JAVA FOR THE STUDY OF EVOLUTION

VOLUME X

TREES

José del Carmen Rodríguez Santamaría
The EvolJava Community

November 20, 2013

Contents

1 Getting a perspective **1**
- 1.1 Randomness as a null hypothesis . . . 1
- 1.2 The challenge of fossils . . . 11
- 1.3 The function paradigm . . . 12
- 1.4 Conclusion . . . 17

2 Java trees **19**
- 2.1 How to draw trees . . . 19
- 2.2 Formats for input . . . 28
- 2.3 Dynamic trees. . . . 41
- 2.4 Recording evolution into a tree . . . 57
- 2.5 Conclusion . . . 73

3 Binary trees **75**
- 3.1 Reusing the general machinery . . . 75
- 3.2 Our approach . . . 76
- 3.3 Our drawing methodology . . . 82
- 3.4 Simulating evolution . . . 116
- 3.5 Correction of the scrolling bug . . . 138
- 3.6 Conclusion . . . 157

4 The nearest neighbor's principle **159**
- 4.1 The Euclidean n-gram distance . . . 159
- 4.2 The nearest neighbor's principle . . . 165
- 4.3 Testing the principle . . . 165
- 4.4 Conclusion . . . 233

5 Phylogenetic trees **235**
- 5.1 Our problem . . . 235
- 5.2 Technicalities . . . 237

5.3 Our solution 368
5.4 Graduation 482
5.5 Conclusion 483

6 Evolution as a very expensive craftsmanship 485
6.1 The mutation model 485
6.2 The function paradigm 486
6.3 Java trees 491
6.4 Tracing the ancestor 492
6.5 Phylogenetic trees 493

Answers to exercises 495

Preface

The series *Java for the Study of Evolution* is directed to scientists that want to manage a serious but not excessively expensive tool to study evolution by direct experimentation under perfectly controlled conditions. These requirements cannot be met in nature but only in simulations and mathematical models. In consequence, the series has three main purposes:

1. To endow the community of **researchers in biology and evolution** with *high level programming*, enabling an accurate study of models and simulations of the most diverse nature.

2. To clearly show how this tool is used to study the fundamental questions of evolution.

3. To suggest that the study of Java could be *very fruitful* for **undergraduates** in biological sciences even more than calculus alone.

This is the tenth volume: Trees. We exhibit the grandiose possibilities of Java to directly represent evolutionary processes by means of various kinds of trees. We study the power of n-gram distances to find direct ancestors along a tree. Besides, we test the power of evolution to attack the problem of finding a phylogenetic tree for a given distance matrix.

Bogotá, Colombia, *José Rodríguez*
November 2013

Introduction

Trees are intrinsically associated with evolution because both instantiate the same father-offspring pairing. So, the study of trees is a must for every student of evolution that wants to pass from a theoretic and journalistic approach to an artisan point of view. In fact, our deal is to make evolution with our own hands and in a self-contained program.

In chapter I, *Getting a perspective*, we get insight: from randomness as a null hypothesis, consigned in the mutation-model, we pass to consider the function paradigm, according to which structures has functions to fulfill. This paradigm encloses the evolutionary theory and every sort of creationism.

Chapter II is dedicated to *Java trees* and how to use them to record an evolutionary process. Chapter III presents *binary trees* as a very useful simplification that is suitable to represent splitting events such as speciation.

Chapter IV, *The nearest neighbor' principle*, studies the possibilities of n-gram distances to indicate the ancestor of an individual whose genotype equals its phenotype: the direct ancestor of a given individual is that specimen of the past generation that is closest to it. Chapter V, *Phylogentic trees*, deals with the *distance matrix to tree problem* which consists in extracting the maximum degree of treehood contained in a distance matrix. When the matrix represents pairwise distances among species, the corresponding tree is a phylogenetic tree. That tree is by its very nature the most robust specific evolutionary theory (the most difficult to falsify) about the origin of considered species. We test the power of Darwinian evolution to solve the distance matrix to tree problem.

Our conclusion is presented in chapter VI: *evolution as a frustration*. By this we mean that our distance matrix to tree problem is too heavy for Darwinian evolution.

Answers to problems conform a fundamental part of the text. Drawing were made with Latexdraw under Ubuntu.

Chapter 1

Getting a perspective

`Discussing the foreground`

1 ***Objective.*** *Every natural explanation of the origin of species must rely on random mutation. So, we formulate and study a model, the* ***mutation-model****, of an evolving population in which reproduction is subjected to mutation and nothing else. Next, we find reasons to consider the* ***functional paradigm*** *as an alternate world-vision to formulate other models. The evolutionary theory of modern science is one of them.*

1.1 Randomness as a null hypothesis

We learned at home how to classify some living beings in different species: dogs, cats, horses, doves and humans. Now, to go from such intuitive capability to sound and solid concepts, we need to become acquainted with various problems. The very first is that a model of mutation per site that happens at random may erode the notion of species.

2 ***An intuitive notion of species.***

We all know that cats breed with cats but not with dogs, and that cats have retractile nails and so are their offspring. Or, the daughter of a cow is good to produce milk in the world that we live in. Besides, cows are too slow to run. With thousand of experiences like this, one learns that likes beget like, that living beings come in species, ensembles of living beings that can be separated by suitable an inheritable characteristics and that do not breed with others but only with specimens in the same group albeit of the other sex.

We see that the very beginning of the discussion about species and speciation is that there are least two species. Nevertheless, modern genetics offers a serious objection against that prerequisite:

Because reproduction is always accompanied with mutation and this is unstoppable and at random, then all possible genotypes are possibly connected by descent. The notion of species is silly in a universe like that because we never would have more than one: it is not excluded that in the long range a cat and a dog might be connected by descent. Nevertheless, and this is a concession, if the number of genotypes is large and if like begets like, then diverse lines of descent might appear as diverse species if observation is restricted to moderate intervals of time and space.

To better understand this very traumatic point, let us make a Java simulation of evolution to show that all phenotypes are related by descent in a universe in which reproduction is tied to random mutation of the genetic material and all genotypes are viable and reproductive.

3 *The mutation-model.*

To capture in a Java simulation the core of the objection, let us define the **mutation-model** as a family of evolving populations that is distinguished by the fact that reproduction is accompanied by mutation at random and all genotypes are viable.

Our task is to prove that all genotypes are connected by descent. To that aim we report the number of generations that are needed to get all possible genotypes in the same line of descent. So, by the characteristics of our problem, we need to consider just one line of descent.

Our evolutionary world has the following characteristics:

1. The population consists in a single string that is synthesized at random. It is n chars long taken from the alphabet $\{A,C,G,T\}$.
2. The string is subjected to evolution with a mutation rate r per site per generation.
3. Strings are at the same time genotypes and phenotypes and all are viable.
4. There is one individual in every generation that can duplicate itself to get only one son and so, there is no selection and no recombination.

4 ***The code follows.***

```
//Program J4 OnlyOneSpecies
/*
Our aim is to show that the concept of species has no
meaning in a world deprived of selection and determined by
duplication and mutation alone in
which all mutants are viable.

The reason is that all possible genotypes are
connected by descent and so we would have only one species.


To test this, we study a simulation with the following
characteristics:

We synthesize at random one string
n chars long taken from the alphabet {A,C,G,T}.
It is subjected  to evolution with a mutation rate r
per site per generation.
Strings are at the same time
genotypes and phenotypes and all are viable.
There is one individual  in every
generation that can duplicate itself to get
only one son and so, there is no
selection and no recombination.
Our task is to prove that all possible  genotypes are
connected by descent.
To that aim we report the number of generations
that are needed
to get  all possible genotypes
in the  only one line of descent.


With this we
prove that the concept of species
has absolutely no  meaning in a world
like the one prescribed:
variations have no limit apart from
those allowed by the working alphabet. So, we
```

```
always have only one possible species making
this concept into a silly definition.
 *
 */


import java.util.Random;




//Main class
public class OnlyOneSpecies
{
 // Definition of the alphabet
 private static String Alphabet = "ATCG";
 //How many symbols in the alphabet
 private static  int lengthAlf   = Alphabet.length();
 //A boolean vector to keep track of generated genotypes
 private static Boolean Indexes[] = new Boolean[2100000];
 //Mutation rate
 private static double m = 0.1;
 //Individual are strings with a constant length
 private static int lengthIndividual;
 //Number of possible genotypes
 private static int N;

//We turn on the random generator:
 private static Random r  = new Random();

//A string of stringLength characters is generated
private static String generateString(int stringLength)
 {
  String  individual = "";
  // Synthesis of a DNA string stringLength bases long
  for(int i= 1; i<= stringLength; i++)
   {
  //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
```

```
   char c = Alphabet.charAt(l);
 individual  = individual +c;
   }
  return individual;
 }


//Returns the index   of the genotype ngram.
//An n-gram is a string with n chars taken from Alphabet.
//Take ngram as a number in base = length of the Alphabet.
//The index is that number in base 10.
//The index is a 1 to 1 association of genotypes
//and numbers 0,1,..., N.
private static int index(int n, String ngram)
{
 char c1 = ' ';
 int p = 0;
 int ind = 0;

 for(int i = 0; i <n; i++)
  {
 c1 = ngram.charAt(i);
 p = Alphabet.indexOf(c1);
 if (p>=0)
  ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
 else
 {
  ind = -1;
 }
  }
 return ind;
}


//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
 String mutant = "";
 for(int i = 0; i < lengthIndividual; i++)
```

```
 {
  double  p = r.nextDouble();
if (p<m)  //mutation
     { //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
   char c = Alphabet.charAt(l);
   mutant = mutant + c;
  }
 else
  { //no mutation
 mutant = mutant + individual.charAt(i);
  }
 }
 return mutant;
}

//Evolution consists in reproduction with only one son
//with mutation. No selection, no recombination.
public static void evolution(String seed)
 {
  boolean done = false;
  int counterGenotypes = 0;
  int counterGenerations = 0;
  String individual = seed;
  while (!(done))
   {
 individual = mutation(individual);
 //System.out.println(individual);
 counterGenerations++;
 //Get the index of individual
 int indexNumber = index(lengthIndividual, individual);
 //If the genotype has been generated do nothing
 if (Indexes[indexNumber] == true) {}
  else //If we have a new genotype:
  {
  //Annotate the new phenotype
   Indexes[indexNumber] = true;
   counterGenotypes++;
   //System.out.println(individual);
```

```
  }
 //If all possible genotypes have been generated
 //end work and report result
 if (counterGenotypes == N) done = true;

   }

  //System.out.println(counterGenerations);


  System.out.println("Number of generations " +
   "to get all genotypes = " + counterGenerations);


 }


public static void main(String[] args)
 {
  //Study with varying length of strings
  for(int j = 1; j < 10; j++)
   {
  lengthIndividual = j;
  //Number of all possible genotypes
  N = (int) Math.pow(lengthAlf, lengthIndividual );
      //No genotype at the beginning
  for(int i = 0; i < Indexes.length; i++)
 Indexes[i]= false;
  System.out.println("Length of strings = "
  + lengthIndividual);
  //An individual is synthesized as a random string
    String seed = generateString(lengthIndividual);
     evolution(seed);
  }
  }
} //End of main class J4 OnlyOneSpecies
```

5 ***Exercise.*** *Run the previous program and play with the code.*

6 ***Exercise.*** *Study the output of the previous program to agree else disagree with the following claims that are related to the results of the previous program:*

1. The necessary number of generations to synthesized all possible genotypes is an exponential function in the length of genotypes. Thus, that time is finite for each length.

2. Therefore, it is fair to conclude that all possible genotypes are connected by descent.

3. Hence, the notion of species is silly.

4. No matter how the program might be modified, to get it more realistic or more powerful or whatever, the conclusion is stable: all possible genotypes are connected by descent in a world with unlimited time and resources if only mutation is at random and not nil.

5. Since the time to visit all possible genotypes is an exponential function in the length of the working strings, it is perfectly sane to admit the notion of species as an statistical concession: during extremely long periods of time, beyond known cosmological constraints, the notion of species is valid and cats are not expected to be in the same line of descent than dogs.

Answer

7 ***A bit of mathematics.***

Java simulations have many virtues but also have their own poison. In fact, the danger of simulations is that when one tries to formulate a law, one is always making a sophism that consist in inferring that all roses are red just because one found tree thousand roses and all of them were red. The remedy is to make simulations but keep an eye on mathematics. The following calculations show that mathematics are indeed necessary to get robust conclusions.

It is immediate to prove that the average time to visit all possible genotype is bounded below by an exponential function in the number of chars of the used alphabet. In effect, let a be the number of chars of the alphabet. We work with strings of constant length, let it be n. Then for each site of the string we have a options and since we have n sites, we have a^n possible phenotypes. So, the minimum possible number of generations to visit all genotypes is the exponential function a^n. Therefore, the average number of generations to visit all genotypes is bounded below by the same function.

Let us prove that there is no absolute guarantee that all strings will be visited in a finite lapse of time. It is just the same that happens with a coin: there is no

guarantee that one will see a head in a finite lapse of time. To see that, let us consider that if the probability of mutation per site per generation is r then the probability of no mutation per site per generation is $1-r$. Now, if a given string has n chars, the probability of getting the same string in the following generation is $(1-r)^n$. So, the probability of getting the very same string without mutation all along a number of T generations is $[(1-r)^n]^T = (1-r)^{nT}$. No matter how large could be n or T, this probability is always non zero. So, we cannot falsify anyhow the proposition that like begets like from generation to generation by mere luck.

Let us see why the probability to have a dog from a cat in this universe is always positive. To see this, let us consider that the dog is encoded by a string d and the cat by string c. For the sake of ease let us consider the case in which d and c are completely different. So, we need a synchronized change in all sites to get from c to d. If the mutation rate per site is r, then the probability to change a char by a different one is r. Therefore, the probability to change a specific char by another specific and different char, one among $a-1$, is $\frac{r}{a-1}$. Hence, the probability to get the asked change is bounded below by $(\frac{r}{a-1})^n$. No matter how low could be r or how large could be n, this probability is always positive. So, we cannot falsify anyhow the claim that cats can beget dogs.

Let us prove now that we can guarantee that most experiments will connect all genotypes by descent in a finite lapse of time. We have seen that the probability of hitting a specific genotype in just one generation is $(\frac{r}{a-1})^n$ or higher. Therefore, the probability of not going into that state in one generation is $c = 1-(\frac{r}{n-1})^n$ or lower, where c is a non negative number less than one. This implies that the probability of not producing that determined string during T generation is c^T or lower. As T runs from 1 to ∞, c^T steadily approaches zero. Thus, one might wait and keep waiting that a time will come when all sites will be visited. The problem is that no one can say when will this happen.

8 ***Challenge.*** *The* ***mutation-model*** *has another mechanism to augment the probability of explaining highly perdurable species: a coherent change in the genotypes while no change is observed in the phenotypes. After all, things can be made in many diverse forms. Make an estimation of that probability.*

9 ***A cornered model.***

We have seen that our **mutation-model** can explain everything in relation with species. It is simply and almighty. Nevertheless, modern science does not rely on

this model. Why? There are two reasons. First, the **mutation-model** is cornered and second, other less cornered models have been proposed.

To understand this jargon, let us imagine that Susan has many suitors but that there are two that specially call her attention because they eagerly love her. The first works as most men do and have some savings. He never has great money at his disposition but he receives enough for a living every month. The second suitor does not work. He likes to play in modern electronic gambling machines. As everyone knows, these machines are programmed so that the owner could win as much as possible and the clients not too much, just the necessary amount to feed the desire to come for more. It happens that the second suitor has been able to earn a rather humble life with these games. Because the relation effort/gain is very favorable for him, he thinks to continue in that way forever.

Let w be the probability that a game may produce a favorable output for the client. This implies that the probability of winning n times is w^n, which is a decreasing function converging to zero. So, Susan cornered the second suitor against a wall and complained: I like you the most, but do you want me to work and never stop working to maintain you? After weeping some three days, she decided to accept the first suitor that has everything needed to be a honorable husband. In fact, as he brings his wage month after month, his fidelity calls for a congratulation.

10 ***Example.***

Let us imagine that the average Joe has probability $t \sim 0$ per day of getting at his work more retarded than his companions because of jamming troubles. Now, John is recurrently arriving the last of all, time over time. One might correctly say that all his delays can be explained by jams at random with probability t^n. Nevertheless, he will be fired because no one will accept that theory. Instead, everyone believes, and does so without further investigation, that he simply likes to sleep and sleep even after the wake up alarm has been triggered. In fact, the probability of arriving the last by chance steadily tends to zero. So, probabilities cornered him against the wall: to invoke jams as the cause of his recurring retarding is no good managing policy: the reason shall be another one.

When John was interrogated, he declared that he uses to visit his father before heading to work. His old father has problems because of age and so John uses to arrange things that his father could stand up and take his breakfast. This explanation produces a satisfactory theory: would it be a motive for the company to make an exception, that explanation would be investigated to see how correct and insuperable could be the mentioned situation. This is an example of a honorable theory: it makes of facts a necessity and of counterexamples a matter of chance.

11 ***Moral.*** *A* ***cornered model*** *is one that explains everything but is dismissed by the very facts it wants to explain because facts are explained by very low probabilities that get still lower when more data are considered. Randomness is the cornered model by excellence. A* ***honorable model*** *explains things as a necessity and in a manner that excites experimentation because it is its failures that are explained by chance.*

Now, the mutation model is cornered because it can explain everything but as more data is considered, say two lines of descent instead of one, the probability of getting crucial facts closes down against zero.

12 ***Exercise.*** *A teenager is spending too much money, in any case more than he receives from his mother. Formulate a cornered model to explain his welfare. Summon him to explain the true reason and verify that it is a honorable model.* *Answer*

13 ***The importance of the mutation-model***

The **mutation-model** is just the null hypothesis of science: randomness is the explanation of everything. Rudimentary as this model may be, it frames very specific and popular proposals to find plausible phylogenies. This will be pursue in the next few chapters.

1.2 The challenge of fossils

It seems that the assertion that the concept of species is a statistical concession in within the **mutation-model** is incontestable. Nevertheless, nature is challenging enough to show us that matters are not that simple. An important objection is posed by archeology.

14 ***The challenge of fossils***

It is accepted in some circles that we have fossils of bacteria billions year old and that they are very similar to those we see around us. Such might be the case of cyanobacteria or “blue-green algae,” whose fossil record extends far back into the Precambrian -nearly 3.5 billion years old (Waggoner and Speer, [13], 1995; Brasier et al, [1], 2006)).

The case of the coelacanth is also very interesting: there is no fossil record of this fish during the last 65 million years, but it has been observed alive very

recently. It is not easy to understand what is behind this reappearance (Casane, Laurenti, [2], 2013).

Please, run the next test: look at the photograph of a mosquito presented by Greenwalt et al ([3], 2013) and make a guess of the lapse of time that have passed since its death. Next, read the age as evaluated by experts. The answer is surprising: where was mutation all over that time?

The controversial theme of living fossils has its own version in this era of molecular biology:

15 ***Conserved sequences***

A **conserved sequence** is a DNA or RNA sequence that appears in almost the same form in different species. The typical example of a conserved sequence is that of the ribosomal DNA, which is present in all species. Its evolutionary importance has motivated the creation of banks and projects dedicated to its study, for instance Silva ([9], 2013), a comprehensive on-line resource for quality checked and aligned ribosomal RNA sequence data.

If conserved sequences are exactly equal one to the others, they are called **ultraconserved sequences** (Reneker et al, [7], 2012).

There is a bank (UCBase, [11], 2013) with 481 ultraconserved sequences longer than 200 bases that share 100% identity among human, mouse and rat.

In conclusion: random mutation is almighty in the sense that it is universal, ubiquitous, permanently and unstoppable. Mutation can explain everything that happens as well as everything that has not happened. So, it can explain evolution wherever it is observed and it can explain ecstasy wherever it is registered. Nevertheless, our everyday experience together with the existence of living fossils, of conserved and ultraconserved sequences are enough reasons to consider that the ecstasy of species is not simply a matter of chance. So, a question naturally arises: Do we have an insight that converts ecstasy into a necessity rather that being explained by extreme low probabilities? It happens that science, creationists and lay people all hold the same view, the function paradigm, which associates functions to physical structures.

1.3 The function paradigm

Eyes are for seeing and ears for hearing. Thus, every one is inborn with the **function paradigm**: structures may have functions that serve a given purpose. We

see that the (intuitive) association of functions to structures is biologically given and so it is inherent to every human culture. In spite of how obvious and natural this paradigm might sound, it is indeed a metaphysical construction. It is the same that happens with gravity: to explain why bodies fall towards the Earth, one must invoke randomness or one might invent gravity or at least believe that it exists. Nevertheless, gravity is not part of direct observations. The same happens with functions: they cannot be weighted or caught. It is through the predictions of formulated models that one can test beliefs. In regard with gravity, one can predict that a stone and a plumb fall in vacuum with the same acceleration. In regard with function, one must resort to design to test whether or not postulated functions correctly fit performance.

16 ***Challenge.*** *Formulate an objective definition of function. Teleonomy, the connection of functions to projects or purposes, must be explicitly included or excluded. Hint: revise Vol II circa SAT.*

In the sequel we will rely on our intuitive conception of function, as well as on that of species.

17 *The problems of the function paradigm*

Our new paradigm is interesting because it poses challenges that tolerate no delay:

From the perspective of the mutation model the origin of life and of species would offer no problem. But these become very terrible puzzles for fans of the function paradigm: where does so huge levels of diversity, complexity, perfection and adaptation come from?

18 *Mutation vs. complexity*

Because we all are loaded with the function paradigm and have a lot of practicing with it, no ones takes seriously the **mutation-model** in which the notion of function is absolutely unnecessary. Instead, the ecstasy of species is an expected fact in the function-paradigm because things to be really functional need to be very complex and this usually means that changes, even tiny, are lethal to the function.

Nevertheless, life on Earth is endowed with mutation. Now, how is the interaction of complexity and mutation?

19 *Evolution as a necessity*

If science is loaded with the functional paradigm over the background of mutation, then evolution in nature is a necessity that unfolds as the result of the interaction of mutation and complexity. So, evolution is the change in genetic content due to the surviving of allowed mutations. Everyone on Earth accepts this as a fact.

20 ***The evolutionary theory in a nutshell***

The incredible similarity among the fish, the pig and the human being is a sufficient reason to propose the Evolutionary Theory: the expected output of the interaction of mutation and complexity is that ecstasy is what we must observe over some few generations while shifts of structure and function is what we must see over geological times. As a byproduct we have the origin of species: ensembles of functions that agglomerate themselves in functional units that re in general incompatible one with another.

A first problem of this theory is the following: How can the interaction of complexity (functional constraint) and mutation explain the origin of the observed extreme perfection of living beings? The answer proposed by the theory is extremely simple: mutants have diverse functional performance and so, some specimens leave more offspring that others. With the cumulation of tiny changes generations over generations functions are improved and perfection appears.

Theories in science are proposals to falsify: mandatory predictions must be formulated to test whether or not they are observed. What do we have here?

21 ***The Evolutionary Theory is obviously false***

We all know that extremely complex problems, such as those that are dealt with by living structures, cannot be solved if not after many trials that leave a track from imperfection to perfection. Therefore it is obvious that evolution is not the universal explanation of species. The reason is that a mandatory prediction of the theory is not backed by nature. Concretely: since mutation in that theory is at random, complex changes must be expected to be accompanied with huge amounts of unsuccessful trials that must be present in every modern population and that should had leaved fossil evidence. But nothing like that is found.

Our discussion has been extremely theoretic, in the air. We need a lab to add a year of reality to our ideas. Such a lab, the best possible one, is here:

22 ***Our official insight***

Our insight is very natural and captures at once the idea that complexity is the air to be breathed in biology:

Our approach is determined by the vision that the genome is software, i.e., the primary function of the genome is to encode for verbal instructions for the continuation of life. As we all know, the development of software is amidst the most complex tasks in the world. Nevertheless, we also know that the capability to design software is not a privative skill of man. Instead, we have made clear in Vol II that evolution, natural evolution, is also a software developer.

Now, the most close reality to every designer is that there is no functional perfection without huge amounts of imperfections. Example, we all know how to make a sheet of paper into a really flying craft, but only until some days ago experts were able of making artificial flying crafts that resemble insects. Now, what is the connection of our papered flying objects and those wonders? They are connected by a human driven evolutionary process that is full in false starts and blind alleys. That is the signature of complexity, which is immanent to software development.

But we have a problem: evolution has no purpose at all. So, we cannot apply to it our experience with purposefully design. Nevertheless, evolution has an emergent and sophisticated aim: to perfection rudimentary functions, specially those that serve the purpose of reproduction and surviving. The mechanism of emergence is the very essence of evolution: if by mere randomness a structure appears that better fits a function, then the possibility exists that it could leave more offspring than its brethren.

That is why the experimental generalization about bugs can be immediately applied to natural evolution: were evolution the cause of the existence of species, the fossil record and modern populations should be plagued with imperfections and malfunctions. Since this is not found, the belief in the origin of most species by Darwinian evolution is just a flaw created by the evolutionary dogma.

Nevertheless, it is not excluded that expected imperfections and malfunctions are not found in some concrete cases because the probability of appearing + registering them might be too low. This may happen when speciation is due to simple changes, say, when it is reduced to the creation of sexual barriers, which can be created by a change in a single gene:

Reproductive isolation may hang on a simple locus, as it happens with bindin, a protein mediating recognition and binding between the sperm and the egg once the sperm has penetrated the egg jelly. Recognizing might be species specific. A high rate of evolution of bindin, i.e., of the creation of sexual barriers, has been found in many species of sea urchin that occupy the same region (Palumbi et al, [6] 2005).

This mechanism about the arising of sexual barriers must be considered together with other known facts:

23 ***Some facts about species and speciation.***

The notion of species comes accompanied with inseparable facts that must be directly or indirectly considered:

1. There are at least two species. Say, cats and dogs.
2. There are descriptors that differentiate cats from dogs and one species from another: cats have round face, dogs have an elongated one.
3. There are descriptors that single out a given species: the activity of making high tech tools to kill one another singles out the human race as a separated group.
4. Every living being can recognize members of its own species: dogs immediately recognize other dogs even if they all are babies.
5. Living beings use special descriptors to discriminate members of the same species. Animals include the use of large sets of pheromones.
6. Members of the same species mate and have viable and reproductive offspring. The purpose of mating is recombination of the genetic material, which is the synthesis of a new material from recombining parts. Mating is a universal phenomenon.
7. One can recombine germ cells from different species with the help of viruses. A zygote is always formed and begins its ontogeny but the time of surviving depends on the nearness of chosen species. The full viability of the zygote is not restricted to germs cells of the same species: one can form a kind of chimeras such as the mule that is a breeding of the horse and the donkey.
8. Living beings have a huge variability: dogs are more different one from another than what cats do. Variability results from both recombination and mutation. Mutation is the unavoidable partial change of the genetic material from parents to offspring.
9. The genetic world is a hot universe. Apart from the warming by recombination and mutation, we have the so called horizontal transfer: viruses can infect diverse species and so they can transport genetic material from one species into another.

10. In spite of variations from parents to offspring, like begets like: cats have retractile nails and so are their offspring.

11. As shown by clone technology, breeding is not essential for reproduction. Thus, recombination or breeding must not be essential to the notion of species.

12. Fecundation hangs on a process of molecular recognizing: the spermatozoon is trapped by a specialized receptor in the ovule that is species discriminating. By the same token, bacteria rely on appropriate molecular receptors to exercise both recognizing and conjugation (the injection of part of the DNA by one bacteria into another. Injected DNA is recombined to that of the host).

13. Sexual barriers may hang on one single locus.

It would take a long time before we all agree on the interpretation of these and related facts about the forging of a robust concept of speciation. At the meantime, we have enough space to exercise our affection for Java.

1.4 Conclusion

Mutation at random at each DNA(RNA) basis is an inherent property of life on Earth. So, we contend that the null hypothesis of science in regard with the origin of species is captured by the **mutation-model**, in which random mutation is all to life, including every fold of morphological or ethological variability and divergence. In effect, our **mutation-model** explains everything by a strike of luck, be something that happened or something that has not been observed. Nevertheless, we live in a social world in which many things happen not by strikes of luck but by necessity, at least during some period of time and under some circumstances, say we love the employment system in which one can earn a life without fluctuating troubles. Therefore, it is normal to look after other models in which usual facts are explained by necessity and exceptions by randomness. Nature has forced us to consider the **function paradigm**: structures may have functions that serve many possible aims and so we can speak of incipient, perfect or imperfect implementations. The evolutionary theory rests on the function paradigm and proposes that species arouse as a natural interaction between function and complexity. That theory is obviously false because a mandatory prediction is not backed by nature: it must exist an abundant presence of imperfection as in modern populations as in the fossil record. Both, the **mutation-model** and the function paradigm allow the proposal of phylogenies, of evolutionary relationships among species that are depicted in evolutionary trees. Their study is the purpose of this volume.

Chapter 2

Java trees

24 ***Purpose.*** *Here we get acquainted with the machinery of Java to deal with trees. We learn to draw trees and to encode trees according to various procedures.*

The Java machinery for trees is really powerful (Oracle, [5], 2013).

2.1 How to draw trees

Let us learn how to instruct Java to draw **trees**.

25 ***Joe and his sons.*** *Joe has three sons: Peter, Paul and James. The corresponding tree is depicted by the next code:*

```
//Program J25 JohnSons
//We learn to draw a tree that represents
//father John and his three sons.

import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

public class JohnSons
```

```
{
  public static void main(String args[])
  {
    JFrame frame = new JFrame("The sons of John");
    //A node is declared
    DefaultMutableTreeNode John = new
     DefaultMutableTreeNode("John is the father");
    //John is made into a root of the tree
    JTree tree = new JTree(John);

    //The sons of John
    //Peter is the first son
    DefaultMutableTreeNode Peter = new
     DefaultMutableTreeNode("Peter is son 1");
    //The node Peter is grafted to node John.
    John.add(Peter);

    DefaultMutableTreeNode Paul = new
     DefaultMutableTreeNode("Paul is son 2");
    John.add(Paul);

    DefaultMutableTreeNode James = new
     DefaultMutableTreeNode("James is son 3");
    John.add(James);

    //Set children visible
    tree.expandPath(new TreePath(John.getPath()));

    //Pane where to draw
    JScrollPane scrollPane = new JScrollPane(tree);
    frame.getContentPane().
              add(scrollPane, BorderLayout.CENTER);
    frame.setSize(300, 150);
    frame.setVisible(true);
  }
}//End of main class J25 JohnSons
```

26 ***Exercise.*** *Run the program and play with the code.*

27 ***Exercise.*** *Add a daughter of Joe to the family.* *Answer*

28 ***Exercise.*** *Imagine the form as Java would implement the possibility to add grandsons to the family. Check your intuition with the next code.*

29 ***Grandsons.*** *This code shows how to add grandsons to the family tree.*

```
//Program J29 JohnSons3
//We learn to draw a tree that represents
//1. Father John and his four sons,
//   a daughter included.
//2. Grandsons

//Observe that the Java tree can be
//expanded and collapsed interactively.

//Expandable nodes, apart from the root, are marked.
//To expand  a node, use double click or
//one click over the expansion-collapsing mark.
//To collapse the branches of a node,
//give one click to the mark.
//To collapse the root, double click.



import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

public class JohnSons3
{
  public static void main(String args[])
  {
    JFrame frame = new JFrame("The sons of John");
    //A node is declared
    DefaultMutableTreeNode John = new
     DefaultMutableTreeNode("John is the father");
```

```
//John is made into a root of the tree
JTree tree = new JTree(John);

//The sons of John
DefaultMutableTreeNode Peter = new
 DefaultMutableTreeNode("Peter is son 1");
//The node Peter is grafted to node John.
John.add(Peter);

DefaultMutableTreeNode Paul = new
 DefaultMutableTreeNode("Paul is son 2");
John.add(Paul);

DefaultMutableTreeNode James = new
 DefaultMutableTreeNode("James is son 3");
John.add(James);

//Node Mary
DefaultMutableTreeNode Mary = new
 DefaultMutableTreeNode("Mary is daughter 1");
//Node Mary is grafted to John
John.add(Mary);

//The grandsons of John:

//Mary has two children, Louise and Mark
DefaultMutableTreeNode Louise = new
  DefaultMutableTreeNode("Louise is daughter of Mary");
Mary.add(Louise);
DefaultMutableTreeNode Mark = new
 DefaultMutableTreeNode("Mark is son of Mary");
Mary.add(Mark);

//Set children visible
tree.expandPath(new TreePath(John.getPath()));

//Pane where to draw
JScrollPane scrollPane = new JScrollPane(tree);
frame.getContentPane().
          add(scrollPane, BorderLayout.CENTER);
```

```
    frame.setSize(300, 150);
    frame.setVisible(true);
  }
}//End of main class 29 JohnSons3
```

30 ***Exercise.*** *Run the program and play with the code.*

31 ***Phylogeny.*** *A* ***phylogenetic tree*** *is a tree that depicts a postulated evolutionary relationship among species. Each phylogenetic tree is a particular evolutionary theory. There is no one single reason under the sun to consider that all species are related by descend but there are thousands of reasons to believe that some species do, say, flies.*

The next diagram shows a phylogenetic tree in which six living species, named after numbers from 1 to 6, are postulated to be related by descend: all species are different now but sometime in the past they shared common ancestors until a single common grandmother is found.

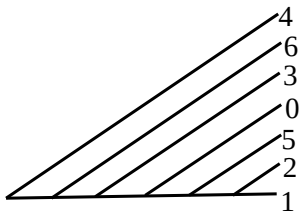


Figure 2.0. A phylogenetic unrooted tree with six living species. The tree is unrooted: the overall ancestor is not defined.

32 ***The Java code for the depicted phylogenic tree above follows:***

```
//Program J32 Phylogeny
//We use a Java tree to represent a phylogeny.

//Observe that the Java tree can be
//expanded and collapsed interactively.

//Expandable nodes, apart from the root, are marked.
//To expand  a node, use double click or
//one click over the expansion-collapsing mark.
//To collapse the branches of a node,
//give one click to the mark.
```

```
//To collapse the root, double click.



import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

public class Phylogeny
{
  public static void main(String args[])
  {
    JFrame frame = new JFrame("The sons of John");
    //A node is declared
    DefaultMutableTreeNode root1 = new
     DefaultMutableTreeNode("Root1");
    //John is made into a root of the tree
    JTree tree = new JTree(root1);


    DefaultMutableTreeNode four = new
     DefaultMutableTreeNode("4");
    root1.add(four);
    DefaultMutableTreeNode root2 = new
     DefaultMutableTreeNode("Root2");
    root1.add(root2);

    DefaultMutableTreeNode six = new
     DefaultMutableTreeNode("6");
    root2.add(six);
    DefaultMutableTreeNode root3 = new
     DefaultMutableTreeNode("Root3");
    root2.add(root3);

    DefaultMutableTreeNode three = new
     DefaultMutableTreeNode("3");
```

```
    root3.add(three);
    DefaultMutableTreeNode root4 = new
     DefaultMutableTreeNode("Root4");
    root3.add(root4);

    DefaultMutableTreeNode zero = new
     DefaultMutableTreeNode("0");
    root4.add(zero);
    DefaultMutableTreeNode root5 = new
     DefaultMutableTreeNode("Root5");
    root4.add(root5);

    DefaultMutableTreeNode five = new
     DefaultMutableTreeNode("5");
    root5.add(five);
    DefaultMutableTreeNode root6 = new
     DefaultMutableTreeNode("Root6");
    root5.add(root6);

    DefaultMutableTreeNode two = new
     DefaultMutableTreeNode("2");
    root6.add(two);
    DefaultMutableTreeNode one = new
     DefaultMutableTreeNode("1");
    root6.add(one);

    //Set children visible
    tree.expandPath(new TreePath(root1.getPath()));


    //Pane where to draw
    JScrollPane scrollPane = new JScrollPane(tree);
    frame.getContentPane().
                add(scrollPane, BorderLayout.CENTER);
    frame.setSize(300, 150);
    frame.setVisible(true);
  }
}//End of main class 32 Phylogeny
```

33 ***Exercise.*** *Run the program and play with the code.*

34 ***Let us try to improve the clarity of the diagram:***

```
//Program J34 Phylogeny2
//We use a Java tree to represent a phylogeny.
//To augment clarity,
//unnecessary names  are not shown.

//Observe that the Java tree can be
//expanded and collapsed interactively.
//Expandable nodes, apart from the root, are marked.
//To expand  a node, use double click or
//one click over the expansion-collapsing mark.
//To collapse the branches of a node,
//give one click to the mark.
//To collapse the root, double click.



import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;

public class Phylogeny2
{
  public static void main(String args[])
  {
    JFrame frame = new JFrame("The sons of John");
    //A node is declared
    DefaultMutableTreeNode root1 = new
     DefaultMutableTreeNode("");
    //John is made into a root of the tree
    JTree tree = new JTree(root1);


    DefaultMutableTreeNode four = new
     DefaultMutableTreeNode("4");
    root1.add(four);
```

```
DefaultMutableTreeNode root2 = new
 DefaultMutableTreeNode("");
root1.add(root2);



DefaultMutableTreeNode six = new
 DefaultMutableTreeNode("6");
root2.add(six);
DefaultMutableTreeNode root3 = new
 DefaultMutableTreeNode("");
root2.add(root3);

DefaultMutableTreeNode three = new
 DefaultMutableTreeNode("3");
root3.add(three);
DefaultMutableTreeNode root4 = new
 DefaultMutableTreeNode("");
root3.add(root4);

DefaultMutableTreeNode zero = new
 DefaultMutableTreeNode("0");
root4.add(zero);
DefaultMutableTreeNode root5 = new
 DefaultMutableTreeNode("");
root4.add(root5);

DefaultMutableTreeNode five = new
 DefaultMutableTreeNode("5");
root5.add(five);
DefaultMutableTreeNode root6 = new
 DefaultMutableTreeNode("");
root5.add(root6);

DefaultMutableTreeNode two = new
 DefaultMutableTreeNode("2");
root6.add(two);
DefaultMutableTreeNode one = new
 DefaultMutableTreeNode("1");
root6.add(one);
```

```
    //Set children visible
    tree.expandPath(new TreePath(root1.getPath()));

    //Pane where to draw
    JScrollPane scrollPane = new JScrollPane(tree);
    frame.getContentPane().
           add(scrollPane, BorderLayout.CENTER);
    frame.setSize(300, 150);
    frame.setVisible(true);
  }
}//End of main class 34 Phylogeny2
```

35 ***Exercise.*** *Run the program and play with the code. Evaluate the improvement of clarity. Agree else disagree with the Author: the Java tree does not match the evolutionary idea that the numbered species exist now and so they must appear at the same vertical line. And besides, the diagram is a little cumbersome to understand.*

36 ***Professional drawing.*** *In spite of the fact that the machinery for Java trees is really powerful, many authors prefer to develop their own drawing facilities. Some of them are high quality software and moreover free. Examples:*

Baobab, figtree (use their names on Internet to prompt a search machine).

We also need a better depicting for our own purposes, so we will develop below our own drawing methods and objects.

2.2 Formats for input

We have seen how to communicate to Java the structure of a tree and how to have it making the corresponding drawing. So, we already know what is going on so, we may proceed to others more convenient forms of formatting the input.

37 ***Hashtables***

The structure of a tree is a recursion to the father-offspring pairing. The generality of pairings can be treated in Java with **hashtables**. So, let us learn to use them.

```
//Program J37   HashtableToTree
/*
 * The structure of a tree is a recursion to
 * the   father-offspring pairing. The generality of
 * pairing can be treated in Java with Hashtables.
 * So, we learn here how  to
 * feed a Java tree with a hashtable.
 */
import java.awt.BorderLayout;
import java.io.Serializable;
import java.util.Hashtable;


import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

public class HashtableToTree
{
  public static void main(String args[])
  {
    JFrame frame = new
      JFrame("HashtableToTree");

    //The root is defined
    DefaultMutableTreeNode Alf = new
      DefaultMutableTreeNode("Alf");
    //Tree declaration
    JTree tree = new JTree(Alf);
   //Hashtable declaration
    Hashtable<String, Object> hashtable = new
     Hashtable<String, Object>();
    //A son of Alf with no children
    hashtable.put("Hans", new String[] {});
    //A second son of Alf with four children
    hashtable.put("Joe", new
      String[] { "Peter", "Paul", "James", "Pamela" });
    //New hashtable for another pairing
```

```
    Hashtable<Serializable, Object> hashtable2 = new
      Hashtable<Serializable, Object>();
   //Mary2 has two sons
    hashtable2.put("Mary2",  new
     String[] { "Louis", "Mark"});

    //Recursion is implemented:
    //Mary2 and her sons is grafted to  Mary.
    //Mary is another daughter of Alf.
    hashtable.put("Mary", hashtable2);
    //The tree is conformed
    JTree.DynamicUtilTreeNode.
                  createChildren(Alf, hashtable);
    //Show the sons of the root = Alf
tree.expandPath(new TreePath(Alf));

    JScrollPane scrollPane = new JScrollPane(tree);
    frame.getContentPane().
         add(scrollPane, BorderLayout.CENTER);
    frame.setSize(300, 500);
    frame.setVisible(true);
  }
}//End of main class 37 hashTableToTree
```

38 ***Exercise.*** *Run and play with the code.*

39 ***Hashtables plus Java trees are all one needs to deal with file directories in Java:***

```
//Program J39 FileDirectory
//The power of Java trees is specially targeted
//to file directories.


import java.awt.BorderLayout;
import java.util.Hashtable;
import java.util.Properties;

import javax.swing.JFrame;
```

```
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

public class FileDirectory
{
  public static void main(String args[])
  {
    JFrame frame = new JFrame("File directory");
    //The root node is declared
    DefaultMutableTreeNode root = new
     DefaultMutableTreeNode("");
    //A tree is declared based on root
    JTree tree = new JTree(root);

   //A helping hashtable is used
    Hashtable<Properties, Properties> hashtable = new
     Hashtable<Properties, Properties>();
    //The properties of the system are read
    Properties props = System.getProperties();
    //Properties are added to the hashtable
    hashtable.put(props, props);

    //The hashtable is grafted to the root
    JTree.DynamicUtilTreeNode.
               createChildren(root, hashtable);
   //Show the sons of the root
   tree.expandPath(new TreePath(root));
    //The pane to draw in
    JScrollPane scrollPane = new JScrollPane(tree);
    frame.getContentPane().
                add(scrollPane, BorderLayout.CENTER);
    frame.setSize(300, 150);
    frame.setVisible(true);
  }
}//End of main class J39 FileDirectory
```

40 ***Exercise.*** *Run and play with the code.*

41 ***Using vectors to input a tree.*** *Vector is a generic class in Java whose in-*

stances represent ordered n-tuples of whatever nature. The following version can represent the pairing father-offspring if the first entry of the vector is the name of the father and the second entry is a linear array with the sons. The virtue of this representation is that one can add sons very easily. Let us see how to use vectors to input the structure of a tree.

```
//Program J41 VectorToTree
/*
 * Vector is a class in Java whose instances
represent  ordered n-tuples of whatever nature.
Vectors can represent the pairing
father-offspring if the first entry of the vector
is the name of the father and
the second entry is a linear array with the sons.
The virtue of this representation is that one can
add sons very easily.
Let us see how to use vectors to
input the structure of a tree.
 */

import java.awt.BorderLayout;
import java.util.Vector;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;


//This inner class represents the
//father-offspring pairing
class PairingVector<E> extends Vector<E>
 {

  private static final long serialVersionUID = 1L;
  //The name of the father
  String name;

  //Constructor1: just the father
  PairingVector(String name)
   {
```

```
    this.name = name;
   }

  //Constructor2: a father  plus sons
  PairingVector(String name, E elements[])
  {
    this.name = name;
    for (int i = 0, n = elements.length; i < n; i++)
    {
      add(elements[i]);
    }
  }

  //Draws automatically  the name of the father
  public String toString()
  {
    return "[" + name + "]";
  }
}

//Main class
public class VectorToTree {
  public static void main(final String args[])
  {
    JFrame frame = new JFrame("VectorToTree");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //A father with sons
    Vector<String> v1 = new
     PairingVector<String>("1", new String[]
     { "11", "12",    "13" });
    //A father<Object> with a name
    Vector<Object> v2 = new PairingVector<Object>("2");
    //A father with sons
    Vector<String> v3 = new
     PairingVector<String>("21", new String[]
     { "211", "212",    "213" });
    //Recursion is implemented:
    //v3 is grafted to v2<Object> = 2
    v2.add(v3);
```

```
    //print name to the console
    System.out.println(v2.toString());
    //Grandpa has two sons: 1 and 2
    Object rootNodes[] = {v1, v2 };
    //The grandpa pairing is officially made
    Vector<Object> rootVector = new
     PairingVector<Object>("Grandpa", rootNodes);
    //A tree is defined based on the Grandpa pairing
    JTree tree = new JTree(rootVector);
    //Pane to draw in
    frame.add(new JScrollPane(tree), BorderLayout.CENTER);
    frame.setSize(300, 500);
    frame.setVisible(true);
  }
} //End of main class J41 VectorToTree
```

42 ***Exercise.*** *Run the previous program and play with the code.*

43 ***Looking for simplicity.***

We have seen powerful machineries in action but the problem is that they are not easy to manage. Can we simplify all this? The next program contains a wonderful idea: Just made an array of lines that consist in a father followed by an array with his sons.

```
//Program J43 EasyPairingToTree
//How to encode a tree?
//Just made an array of lines:
//in each line one must find
//a father followed by an array with  his sons.

/*
 * Slight modification of:
Java Swing, 2nd Edition
By Marc Loy, Robert Eckstein, Dave Wood, James Elliott, Brian Cole
ISBN: 0-596-00408-7
Publisher: O'Reilly

Program:
ObjectTree.java
```

```
*/

import java.awt.BorderLayout;
import java.util.Hashtable;

import javax.swing.JFrame;
import javax.swing.JTree;

public class EasyPairingToTree extends JFrame {


private static final long serialVersionUID = 1L;

 JTree tree;

 //This array contains the paring:
 //the first element of each line is the father
 //the second element is an array with the sons.
  String[][] pairingData = {
 { "Amy" }, { "Brandon", "Bailey" },
 { "Jodi" }, { "Trent", "Garrett", "Paige", "Dylan" },
 { "Donn" },  { "Nancy", "Donald", "Phyllis", "John", "Pat" },
 { "Ron" },  { "Linda", "Mark", "Lois", "Marvin" } };

  //Constructor: pane to draw in
  public EasyPairingToTree()
  {
    super("EasyPairingToTree");
    setSize(400, 700);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
  }

  //The array pairingData is used to build a hashtable,
  //which is used to build a tree
  public void init()
  {
    Hashtable<String, String[]> h = new
     Hashtable<String, String[]>();
    // Build up the hashtable using even entries
    //in the String[][] as  fathers, which are
```

```
    // followed by an array with the sons.
    //i+=2 means i = i+2: data are scanned by pairs
    for (int i = 0; i < pairingData.length; i += 2)
      h.put(pairingData[i][0], pairingData[i + 1]);
    //The hashtable is used to feed a tree
    tree = new JTree(h);
    getContentPane().add(tree, BorderLayout.CENTER);
  }

  public static void main(String args[])
  {
EasyPairingToTree tt = new EasyPairingToTree();
    tt.init();
    tt.setVisible(true);
  }
}//End of main class J43 EasyPairingToTree
```

44 ***Exercise.*** *Run and play with the code.*

45 ***Challenge.*** *Study the code of the previous program, verify that it does not implement recursiveness. Made you own objects to make a full recursive implementation of the wonderful idea offered by that code.*

46 ***Research: editable tree.*** *The following code was downloaded from Internet. It is just an example of how incredible is the power of Java to deal with trees. Study the code and populate it with comments to make it full understandable to other users. Play with the compiled program to make sure that you have grasped all its functions. Insert at the beginning of the code a concise but complete operating manual. Change the name of key referents in such a way that the father-sons pairing becomes apparent adding in that way more transparency to the code.*

```
//Program J46 EditableTree
//A GUI for editing and reediting a tree


/*Source at:

http://www.java2s.com/Tutorial/Java/
0240__Swing/SearchingnodeinaJTree.htm

*/
```

```
//Verified 28/VI/2013

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Enumeration;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;

public class EditableTree extends JFrame {

  /**
 *
 */
private static final long serialVersionUID = 1L;

private DefaultMutableTreeNode m_rootNode = new
           DefaultMutableTreeNode("AA");

  private DefaultTreeModel m_model = new
   DefaultTreeModel(m_rootNode);

  private JTree m_tree = new JTree(m_model);

  private JButton m_addButton = new JButton("Add Node");

  private JButton m_delButton = new JButton("Delete Node");
```

```
  private JButton m_searchButton = new
              JButton("Search Node");

  private JButton m_searchAndDeleteButton = new
  JButton("Search and Delete Node");

  private JTextField m_searchText;

  public EditableTree() {
    DefaultMutableTreeNode forums = new
     DefaultMutableTreeNode("A");
    forums.add(new DefaultMutableTreeNode("B"));
    DefaultMutableTreeNode articles = new
     DefaultMutableTreeNode("E");
    articles.add(new DefaultMutableTreeNode("F"));
    DefaultMutableTreeNode examples = new
     DefaultMutableTreeNode("G");
    examples.add(new DefaultMutableTreeNode("H"));

    m_rootNode.add(forums);
    m_rootNode.add(articles);
    m_rootNode.add(examples);

    m_tree.setEditable(true);
    m_tree.setSelectionRow(0);

    JScrollPane scrollPane = new JScrollPane(m_tree);
    getContentPane().add(scrollPane, BorderLayout.CENTER);

    JPanel panel = new JPanel();

    m_addButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        DefaultMutableTreeNode selNode =
         (DefaultMutableTreeNode) m_tree
            .getLastSelectedPathComponent();
        if (selNode == null) {
          return;
        }
        DefaultMutableTreeNode newNode = new
```

```
        DefaultMutableTreeNode("New Node");
      m_model.insertNodeInto(newNode, selNode,
                          selNode.getChildCount());

      TreeNode[] nodes = m_model.getPathToRoot(newNode);
      TreePath path = new TreePath(nodes);
      m_tree.scrollPathToVisible(path);
      m_tree.setSelectionPath(path);
      m_tree.startEditingAtPath(path);
    }
  });
  panel.add(m_addButton);

  m_delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      DefaultMutableTreeNode selNode =
       (DefaultMutableTreeNode) m_tree
          .getLastSelectedPathComponent();
      removeNode(selNode);
    }
  });
  panel.add(m_delButton);

  JPanel searchPanel = new JPanel();
  searchPanel.setBorder(BorderFactory.
            createEtchedBorder());

  m_searchText = new JTextField(10);
  searchPanel.add(m_searchText);

  m_searchButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      DefaultMutableTreeNode node =
       searchNode(m_searchText.getText());
      if (node != null) {
        TreeNode[] nodes = m_model.getPathToRoot(node);
        TreePath path = new TreePath(nodes);
        m_tree.scrollPathToVisible(path);
        m_tree.setSelectionPath(path);
      } else {
```

```
          System.out.println("Node with string " +
                  m_searchText.getText() + " not found");
        }
      }
    });
    searchPanel.add(m_searchButton);

    m_searchAndDeleteButton.addActionListener(new
             ActionListener() {
      public void actionPerformed(ActionEvent e) {
        DefaultMutableTreeNode node =
         searchNode(m_searchText.getText());
        if (node != null) {
          removeNode(node);
        } else {
          System.out.println("Node with string " +
                    m_searchText.getText() + " not found");
        }
      }
    });
    searchPanel.add(m_searchAndDeleteButton);
    panel.add(searchPanel);
    getContentPane().add(panel, BorderLayout.SOUTH);
    setSize(750, 700);
    setVisible(true);
  }

  public DefaultMutableTreeNode searchNode(String nodeStr) {
    DefaultMutableTreeNode node = null;
    Enumeration<?> e = m_rootNode.breadthFirstEnumeration();
    while (e.hasMoreElements()) {
      node = (DefaultMutableTreeNode) e.nextElement();
      if (nodeStr.equals(node.getUserObject().toString())) {
        return node;
      }
    }
    return null;
  }

  public void removeNode(DefaultMutableTreeNode selNode) {
```

```
    if (selNode == null) {
      return;
    }
    MutableTreeNode parent = (MutableTreeNode)
     (selNode.getParent());
    if (parent == null) {
      return;
    }
    MutableTreeNode toBeSelNode = getSibling(selNode);
    if (toBeSelNode == null) {
      toBeSelNode = parent;
    }
    TreeNode[] nodes = m_model.getPathToRoot(toBeSelNode);
    TreePath path = new TreePath(nodes);
    m_tree.scrollPathToVisible(path);
    m_tree.setSelectionPath(path);
    m_model.removeNodeFromParent(selNode);
  }

  private MutableTreeNode
          getSibling(DefaultMutableTreeNode selNode) {
    MutableTreeNode sibling = (MutableTreeNode)
      selNode.getPreviousSibling();
    if (sibling == null) {
      sibling = (MutableTreeNode) selNode.getNextSibling();
    }
    return sibling;
  }

  public static void main(String[] arg) {
    new EditableTree();
  }
}//End of main class J46 EditableTree
```

2.3 Dynamic trees.

Our trees are intended to serve as transparent expressions of evolutionary processes. So, they cannot be static, frozen entities. Quite to the contrary, we must

deal with reproduction that implies that a simulation must be enabled with the capability of autonomously representing child birth at once.

47 ***The next code shows us how to incorporate the automatic birth of nodes into a dynamic tree.***

```
//Program J47 DynamicTreeDemo
//How to enable automatic child birth in trees.
//This is a prerequisite for simulations of evolution.

/*
 * Operation manual:
 * Select a node, then click over the add button
 * to add a new node to the selected node.
 * Or, click over the delete button to delete the
 * selected node.
 * To delete all nodes with the exception of the root,
 * click over the clear button.

 *
 * The Random_Change button generates
 * a change chosen at random:
 * a new child is added ( at a node chosen at random)
 * else an automatically  selected node is deleted
 * else the tree is cleared.
 *
 */

//============================================================


//This program is a modification to the program with
//equal name downloaded from

/*
 * http://www.java2s.com/
 * Tutorial/Java/0240__Swing/DynamicTree.htm
 *
 * Cited 28/VI/2013.
 *
```

```
 * Modifications made by JR of the Evoljava Community.
 * Modifications have unrestricted license.
 *
 * License of the original source code:
 *
 * Copyright (c) 1995 - 2008 Sun Microsystems, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that
 * the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer.
 *
 *   - Redistributions in binary form must reproduce
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer in the
 *     documentation and/or other materials
 *     provided with the distribution.
 *
 *   - Neither the name of Sun Microsystems nor
 *     the names of its
 *     contributors may be used to endorse or
 *     promote products derived
 *     from this software without specific
 *     prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
```

```
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

//package components;
/*
 * This code is based on an example
 * provided by Richard Stanford,
 * a tutorial reader.
 */

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
```

```
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

public class DynamicTreeDemo extends JPanel
                         implements ActionListener
{

 private static final long serialVersionUID = 1L;

 //Varying numerical suffix to name the new nodes
 private static int newNodeSuffix = 1;
 //Commands for the buttons
 private static String Random_Change_COMMAND =
                          "Random_Change";
 private static String ADD_COMMAND = "add";
 private static String REMOVE_COMMAND = "remove";
 private static String CLEAR_COMMAND = "clear";



  //DynamicTree is an inner service class that comes
  //at the end of this file.
  private DynamicTree treePanel;



  //Constructor of main class: GUI basics
  public DynamicTreeDemo()
  {
    super(new BorderLayout());

    // Create the components.
    treePanel = new DynamicTree();
    populateTree(treePanel);
```

```
    //A new node is added
    JButton addButton = new JButton("Add");
    addButton.setActionCommand(ADD_COMMAND);
    addButton.addActionListener(this);

    //A selected by the User node is removed
    JButton removeButton = new JButton("Remove");
    removeButton.setActionCommand(REMOVE_COMMAND);
    removeButton.addActionListener(this);

    //The tree is cleared out.
    JButton clearButton = new JButton("Clear");
    clearButton.setActionCommand(CLEAR_COMMAND);
    clearButton.addActionListener(this);

    //This button generates a change chosen at random:
    //a new node is added
    //else a selected node is deleted
    //else the tree is cleared.
    JButton Random_ChangeButton = new
     JButton("Random_Change");
    Random_ChangeButton.
    setActionCommand(Random_Change_COMMAND);
    Random_ChangeButton.addActionListener(this);

    // Lay everything out.
    treePanel.setPreferredSize(new Dimension(800, 500));
    add(treePanel, BorderLayout.CENTER);

    //Lay all four buttons in line
    JPanel panel = new JPanel(new GridLayout(0, 4));

    //Add buttons to the panel
    panel.add(Random_ChangeButton);
    panel.add(addButton);
    panel.add(removeButton);
    panel.add(clearButton);
    add(panel, BorderLayout.SOUTH);
  }//End of constructor
```

```
//Initialization of the tree
public void populateTree(DynamicTree treePanel)
{
  String p1Name = new String("Parent 1");
  String p2Name = new String("Parent 2");
  String c1Name = new String("Child 1");
  String c2Name = new String("Child 2");

  DefaultMutableTreeNode p1, p2;

  p1 = treePanel.addObject(null, p1Name);
  p2 = treePanel.addObject(null, p2Name);

  treePanel.addObject(p1, c1Name);
  treePanel.addObject(p1, c2Name);

  treePanel.addObject(p2, c1Name);
  treePanel.addObject(p2, c2Name);
}


public void actionPerformed(ActionEvent e) {
  String command = e.getActionCommand();

  if (ADD_COMMAND.equals(command)) {
    // Add button clicked
    treePanel.addObject("New Node " + newNodeSuffix++);
  } else if (REMOVE_COMMAND.equals(command)) {
    // Remove button clicked
    treePanel.removeCurrentNode();
  } else if (CLEAR_COMMAND.equals(command)) {
    // Clear button clicked.
    treePanel.clear();
  } else if (Random_Change_COMMAND.equals(command)) {
   //A random change is made: add, delete, clear.
    treePanel.randomChange();
  }
  report();
  System.out.println("\nAll nodes in the tree:");
```

```
    visitAllNodes(treePanel.rootNode);
  }

  //The next method is recursive,
  //This variable sets the initial setting
  static int NumberNodes = 0;
  //List all nodes in the tree
  public static int visitAllNodes(TreeNode node) {

    System.out.println(node);
    NumberNodes++;
    if (node.getChildCount() >= 0) {
      for (Enumeration<?> e = node.children();
        e.hasMoreElements();)
       {
        TreeNode n = (TreeNode) e.nextElement();
        visitAllNodes(n);
       }
    }
    return NumberNodes;
  }




  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }
```

```
  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
  private static void createAndShowGUI()
  {
    // Create and set up the window.
    JFrame frame = new JFrame("DynamicTreeDemo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


    // Create and set up the content pane.
    DynamicTreeDemo newContentPane = new DynamicTreeDemo();
    // content panes must be opaque
    newContentPane.setOpaque(true);
    frame.setContentPane(newContentPane);

    // Display the window.
    frame.pack();
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createAndShowGUI();
      }
    });
  }


//=============================================================
//==================Inner service class=======================
//=============================================================
```

```
//This program is a modification to the program with
//equal name downloaded from

/*
 * http://www.java2s.com/
 * Tutorial/Java/0240__Swing/DynamicTree.htm
 *
 * Cited 28/VI/2013.
 *
 * Modifications made by JR of the Evoljava Community.
 * Modifications have unrestricted license.
 *
 * License of the original source code:
 *
 * Copyright (c) 1995 - 2008 Sun Microsystems, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that
 * the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer.
 *
 *   - Redistributions in binary form must reproduce
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer in the
 *     documentation and/or other materials
 *     provided with the distribution.
 *
 *   - Neither the name of Sun Microsystems nor
 *     the names of its
 *     contributors may be used to endorse or
 *     promote products derived
 *     from this software without specific
```

```
 *     prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

// package components;
/*
 * This code is based on an example provided
 * by Richard Stanford, a tutorial
 * reader.
 */
class DynamicTree extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
```

```
private Toolkit toolkit = Toolkit.getDefaultToolkit();
Random r = new Random();

//Constructor
public DynamicTree()
{
   super(new GridLayout(1, 0));

   rootNode = new DefaultMutableTreeNode("Root Node");
   treeModel = new DefaultTreeModel(rootNode);

   tree = new JTree(treeModel);
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);

   JScrollPane scrollPane = new JScrollPane(tree);
   add(scrollPane);
 }


 /** Remove all nodes except the root node. */
 public void clear() {
   rootNode.removeAllChildren();
   treeModel.reload();
 }

 /** Remove the currently selected node. */
 public void removeCurrentNode() {
   TreePath currentSelection = tree.getSelectionPath();

   if (currentSelection != null) {
     DefaultMutableTreeNode currentNode =
      (DefaultMutableTreeNode) (currentSelection
         .getLastPathComponent());
     MutableTreeNode parent =
      (MutableTreeNode) (currentNode.getParent());
     if (parent != null) {
       treeModel.removeNodeFromParent(currentNode);
```

```
        return;
      }
    }

    // Either there was no selection, or the root was selected.
    toolkit.beep();
  }




  /** Remove the currently selected node. */
  public void removeRandomNode()
  {
//To select a node at random,
//we go the graphic of the tree,
//measure it and take its width.
//Next, a random number less that width is thrown
//and the node that is closest to
//the corresponding pixel is selected.
Dimension d = tree.getPreferredScrollableViewportSize();
int i = r.nextInt(d.width);
//System.out.println("dx " + d.height + "dy = "
//+ d.width + "i = " + i);
    TreePath currentSelection =
        tree.getClosestPathForLocation(50, i);

    //Node is removed
    if (currentSelection != null) {
      DefaultMutableTreeNode currentNode =
       (DefaultMutableTreeNode) (currentSelection
          .getLastPathComponent());
      MutableTreeNode parent =
       (MutableTreeNode) (currentNode.getParent());
      if (parent != null) {
        treeModel.removeNodeFromParent(currentNode);
        return;
      }
    }
```

```
    // Either there was no selection, or the root was selected.
    toolkit.beep();
  }

  //A random change to the tree is made
  public void randomChange()
  {
int what = r.nextInt(3);
if (what == 1) removeRandomNode();
if (what == 2) addObjectAtRandom("New Node "
             + DynamicTreeDemo.newNodeSuffix++);
if (what == 3) clear();
  }



  /** Add child to  node selected at random. */
  public DefaultMutableTreeNode
              addObjectAtRandom(Object child) {
    DefaultMutableTreeNode parentNode = null;
    Dimension d = tree.getPreferredScrollableViewportSize();
    int i = r.nextInt(d.width);
    TreePath parentPath =
     tree.getClosestPathForLocation(50, i);

    if (parentPath == null) {
      parentNode = rootNode;
    } else {
      parentNode = (DefaultMutableTreeNode)
               (parentPath.getLastPathComponent());
    }

    return addObject(parentNode, child, true);
  }

  /** Add child to the currently selected node. */
  public DefaultMutableTreeNode addObject(Object child) {
    DefaultMutableTreeNode parentNode = null;
    TreePath parentPath = tree.getSelectionPath();
```

```
    if (parentPath == null) {
      parentNode = rootNode;
    } else {
      parentNode = (DefaultMutableTreeNode)
             (parentPath.getLastPathComponent());
    }

    return addObject(parentNode, child, true);
  }



  public DefaultMutableTreeNode
               addObject(DefaultMutableTreeNode parent,
      Object child) {
    return addObject(parent, child, true);
  }



  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                     new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
               (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
```

```
    }
    return childNode;
  }

  //Listener
  class MyTreeModelListener implements TreeModelListener {
    public void treeNodesChanged(TreeModelEvent e) {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e) {
    }

    public void treeNodesRemoved(TreeModelEvent e) {
    }

    public void treeStructureChanged(TreeModelEvent e) {
    }
  }
}
```

```
}//End of main class J47 DynamicTreeDemo
```

48 ***Exercise.*** *Run the program and play with the code.*

2.4 Recording evolution into a tree

We aim at recording in a tree a simulation of evolution. We will consider that generations are non-overlapping and so, the only generation that is subjected to reproduction is the last one. When ones uses a tree to record the evolutionary process, the last generation includes the **leaves of the tree**, those nodes that have no children. So, we need to identify leaves.

49 ***Code that shows how to enumerate the leaves of a tree.***

```
//Program J49 LeavesOfTree
//This program lists the leaves of the tree.
//This is a prerequisite for simulations of evolution.

/*
 * Operation manual:
 * Select a node, then click over the add button
 * to add a new node to the selected node.
 * Or, click over the delete button to delete the
 * selected node.
 * To delete all nodes with the exception of the root,
 * click over the clear button.

 *
 * The Random_Change button generates
 * a change chosen at random:
 * a new child is added ( at a node chosen at random)
 * else an automatically  selected node is deleted
 * else the tree is cleared.
 *
 * To list leaves. click over the
 * corresponding button.
 */

//============================================================
```

```
//This program is a modification to the program with
//equal name downloaded from

/*
 * http://www.java2s.com/
 * Tutorial/Java/0240__Swing/DynamicTree.htm
 *
 * Cited 28/VI/2013.
 *
 * Modifications made by JR of the Evoljava Community.
 * Modifications have unrestricted license.
 *
 * License of the original source code:
 *
 * Copyright (c) 1995 - 2008 Sun Microsystems, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that
 * the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer.
 *
 *   - Redistributions in binary form must reproduce
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer in the
 *     documentation and/or other materials
 *     provided with the distribution.
 *
 *   - Neither the name of Sun Microsystems nor
 *     the names of its
 *     contributors may be used to endorse or
 *     promote products derived
```

```
 *     from this software without specific
 *     prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

//package components;
/*
 * This code is based on an example
 * provided by Richard Stanford,
 * a tutorial reader.
 */

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

public class LeavesOfTree extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;

 //Varying numerical suffix to name the new nodes
 private static int newNodeSuffix = 1;
 //Commands for the buttons
 private static String Random_Change_COMMAND =
                          "Random_Change";
 private static String ADD_COMMAND = "add";
 private static String REMOVE_COMMAND = "remove";
 private static String CLEAR_COMMAND = "clear";
 private static String ListLeaves_COMMAND = "list";



  //DynamicTree is an inner service class that comes
  //at the end of this file.
  private DynamicTree2 treePanel;
```

```
//Constructor of main class: GUI basics
public LeavesOfTree()
{
  super(new BorderLayout());

  // Create the components.
  treePanel = new DynamicTree2();
  populateTree(treePanel);



  //A new node is added
  JButton addButton = new JButton("Add");
  addButton.setActionCommand(ADD_COMMAND);
  addButton.addActionListener(this);

  //A selected by the User node is removed
  JButton removeButton = new JButton("Remove");
  removeButton.setActionCommand(REMOVE_COMMAND);
  removeButton.addActionListener(this);

  //The tree is cleared out.
  JButton clearButton = new JButton("Clear");
  clearButton.setActionCommand(CLEAR_COMMAND);
  clearButton.addActionListener(this);

  //This button generates a change chosen at random:
  //a new node is added
  //else a selected node is deleted
  //else the tree is cleared.
  JButton Random_ChangeButton = new
   JButton("Random_Change");
  Random_ChangeButton.
  setActionCommand(Random_Change_COMMAND);
  Random_ChangeButton.addActionListener(this);

  JButton ListLeavesButton = new
   JButton("List leaves");
  ListLeavesButton.
```

```
    setActionCommand(ListLeaves_COMMAND);
    ListLeavesButton.addActionListener(this);

    // Lay everything out.
    treePanel.setPreferredSize(new Dimension(1200, 500));
    add(treePanel, BorderLayout.CENTER);

    //Lay all four buttons in line
    JPanel panel = new JPanel(new GridLayout(1, 6));

    //Add buttons to the panel
    panel.add(Random_ChangeButton);
    panel.add(addButton);
    panel.add(removeButton);
    panel.add(clearButton);
    panel.add(ListLeavesButton);


    add(panel, BorderLayout.SOUTH);
  }//End of constructor

  //Initialization of the tree
  private void populateTree(DynamicTree2 treePanel)
  {
    String p1Name = new String("Parent 1");
    String p2Name = new String("Parent 2");
    String c1Name = new String("Child 1");
    String c2Name = new String("Child 2");

    DefaultMutableTreeNode p1, p2;

    p1 = treePanel.addObject(null, p1Name);
    p2 = treePanel.addObject(null, p2Name);

    treePanel.addObject(p1, c1Name);
    treePanel.addObject(p1, c2Name);

    treePanel.addObject(p2, c1Name);
    treePanel.addObject(p2, c2Name);
  }
```

```
public void actionPerformed(ActionEvent e) {
  String command = e.getActionCommand();

  if (ADD_COMMAND.equals(command)) {
    // Add button clicked
    treePanel.addObject("New Node " + newNodeSuffix++);
  } else if (REMOVE_COMMAND.equals(command)) {
    // Remove button clicked
    treePanel.removeCurrentNode();
  } else if (CLEAR_COMMAND.equals(command)) {
    // Clear button clicked.
    treePanel.clear();
  } else if (Random_Change_COMMAND.equals(command)) {
   //A random change is made: add, delete, clear.
    treePanel.randomChange();
  } else if (ListLeaves_COMMAND.equals(command)) {
   //A random change is made: add, delete, clear.
   System.out.println("\nLeaves of the tree:");
    treePanel.ListLeaves();
  }
  report();
  System.out.println("\nAll nodes in the tree:");
  visitAllNodes(treePanel.rootNode);
}

//The next method is recursive,
//This variable sets the initial setting
static int NumberNodes = 0;
//List all nodes in the tree
private static int visitAllNodes(TreeNode node) {

  System.out.println(node);
  NumberNodes++;
  if (node.getChildCount() >= 0) {
    for (Enumeration<?> e = node.children();
      e.hasMoreElements();)
     {
      TreeNode n = (TreeNode) e.nextElement();
```

```
        visitAllNodes(n);
       }
    }
    return NumberNodes;
  }



  //Data about the tree is printed to the console
  private void report()
  {
System.out.println("\nThe root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
  private static void createAndShowGUI()
  {
    // Create and set up the window.
    JFrame frame = new JFrame("Leaves of Tree");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


    // Create and set up the content pane.
    LeavesOfTree newContentPane2 = new LeavesOfTree();
    // content panes must be opaque
```

```
    newContentPane2.setOpaque(true);
    frame.setContentPane(newContentPane2);

    // Display the window.
    frame.pack();
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createAndShowGUI();
      }
    });
  }


//=============================================================
//==================Inner service class=======================
//=============================================================


//This program is a modification to the program with
//equal name downloaded from

/*
 * http://www.java2s.com/
 * Tutorial/Java/0240__Swing/DynamicTree.htm
 *
 * Cited 28/VI/2013.
 *
 * Modifications made by JR of the Evoljava Community.
 * Modifications have unrestricted license.
 *
 * License of the original source code:
 *
 * Copyright (c) 1995 - 2008 Sun Microsystems, Inc.
 * All rights reserved.
```

```
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that
 * the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer.
 *
 *   - Redistributions in binary form must reproduce
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer in the
 *     documentation and/or other materials
 *     provided with the distribution.
 *
 *   - Neither the name of Sun Microsystems nor
 *     the names of its
 *     contributors may be used to endorse or
 *     promote products derived
 *     from this software without specific
 *     prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR
```

```
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

// package components;
/*
 * This code is based on an example provided
 * by Richard Stanford, a tutorial
 * reader.
 */
class DynamicTree2 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 private Toolkit toolkit = Toolkit.getDefaultToolkit();
 Random r = new Random();

 //Constructor
 public DynamicTree2()
 {
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode("Root Node");
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);
```

```
    JScrollPane scrollPane = new JScrollPane(tree);
    add(scrollPane);
  }


  /** Remove all nodes except the root node. */
  private void clear() {
    rootNode.removeAllChildren();
    treeModel.reload();
  }

  /** Remove the currently selected node. */
  private void removeCurrentNode() {
    TreePath currentSelection = tree.getSelectionPath();

    if (currentSelection != null) {
      DefaultMutableTreeNode currentNode =
       (DefaultMutableTreeNode) (currentSelection
          .getLastPathComponent());
      MutableTreeNode parent =
       (MutableTreeNode) (currentNode.getParent());
      if (parent != null) {
        treeModel.removeNodeFromParent(currentNode);
        return;
      }
    }

    // Either there was no selection, or the root was selected.
    toolkit.beep();
  }




  /** Remove the currently selected node. */
  private void removeRandomNode()
  {
//To select a node at random,
```

```
//we go the graphic of the tree,
//measure it and take its width.
//Next, a random number less that width is thrown
//and the node that is closest to
//the corresponding pixel is selected.
Dimension d = tree.getPreferredScrollableViewportSize();
int i = r.nextInt(d.width);
//System.out.println("dx " + d.height + "dy = "
//+ d.width + "i = " + i);
    TreePath currentSelection =
        tree.getClosestPathForLocation(50, i);

    //Node is removed
    if (currentSelection != null) {
      DefaultMutableTreeNode currentNode =
       (DefaultMutableTreeNode) (currentSelection
          .getLastPathComponent());
      MutableTreeNode parent =
       (MutableTreeNode) (currentNode.getParent());
      if (parent != null) {
        treeModel.removeNodeFromParent(currentNode);
        return;
      }
    }

    // Either there was no selection, or the root was selected.
    toolkit.beep();
  }

  //A random change to the tree is made
  private void randomChange()
  {
int what = r.nextInt(3);
if (what == 1) removeRandomNode();
if (what == 2) addObjectAtRandom("New Node "
             + LeavesOfTree.newNodeSuffix++);
if (what == 3) clear();
  }
```

```
/** Add child to  node selected at random. */
private DefaultMutableTreeNode
            addObjectAtRandom(Object child) {
  DefaultMutableTreeNode parentNode = null;
  Dimension d = tree.getPreferredScrollableViewportSize();
  int i = r.nextInt(d.width);
  TreePath parentPath =
   tree.getClosestPathForLocation(50, i);

  if (parentPath == null) {
    parentNode = rootNode;
  } else {
    parentNode = (DefaultMutableTreeNode)
             (parentPath.getLastPathComponent());
  }

  return addObject(parentNode, child, true);
}

/** Add child to the currently selected node. */
private DefaultMutableTreeNode addObject(Object child) {
  DefaultMutableTreeNode parentNode = null;
  TreePath parentPath = tree.getSelectionPath();

  if (parentPath == null) {
    parentNode = rootNode;
  } else {
    parentNode = (DefaultMutableTreeNode)
           (parentPath.getLastPathComponent());
  }

  return addObject(parentNode, child, true);
}



private DefaultMutableTreeNode
             addObject(DefaultMutableTreeNode parent,
    Object child) {
```

```
    return addObject(parent, child, true);
  }



  private DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                    new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
                (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }

  //Lists the leaves of the tree.
  //Leaf =  node without children
  private DefaultMutableTreeNode ListLeaves()
  {
   DefaultMutableTreeNode node = null;
   Enumeration<?> e = rootNode.depthFirstEnumeration();
   while (e.hasMoreElements())
    {
 node = (DefaultMutableTreeNode) e.nextElement();
 if(node.isLeaf())
      {
     System.out.println(node.toString()) ;
```

```
      }
}
   return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener {
    public void treeNodesChanged(TreeModelEvent e) {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e) {
    }

    public void treeNodesRemoved(TreeModelEvent e) {
    }

    public void treeStructureChanged(TreeModelEvent e) {
    }
  }
```

```
}

}//End of main class J49 LeavesOfTree
```

50 ***Exercise.*** *Run the previous program and play with the code.*

51 ***Exercise.*** *Develop a program to record into a tree an evolutionary process under the possible simplest implementation of the* ***mutation-model*** *(just consider mutation).*

52 ***Research.*** *We have introduced the ready to use tree class that comes with Java. The need may arise, nevertheless, to design own version of this class to get complete dominion over everything one does. So, prompt Internet with the key phrases: Java tree class, or Java tree data structure to see how this is done.*

2.5 Conclusion

Trees naturally represent the recursion to the father-offspring pairing: a father that has sons, which also become fathers with more sons, and so on. This relation, also known as a branching process, is so important that it has a special class in Java: the tree class. We have learned various formalisms to communicate Java the structure of a tree and how to make the corresponding diagram. In spite of the fact that we saw a minimum material, it was enough to recognize how powerful is the Java tree class, allowing us to expand and contract subtrees, edit a tree interactively, and to generate nodes automatically such as it is needed in a simulation of evolution, the matter that we spin around.

Chapter 3

Binary trees

`A maximum of two sons`

53 ***Introduction and purpose.*** *As we have seen, trees result from recursion to the father-offspring pairing. On the other hand, the use of recursion turns things to be very complex. So, trees are in general complicated. There is, nevertheless, a subgroup of trees that is astonishingly simple and that at the same time suffices for many evolutionary studies: it is the class of binary trees, which are those in which each node can have a maximum of two children. We will study them in this chapter.*

3.1 Reusing the general machinery

Binary trees also are trees, so the machinery that we got acquainted with in the previous chapter is good to deal with them. This will be explored here but before a word of caution might be appropriate.

54 ***Virtues and pitfalls of binary trees***

The creation of new species is in general a rare event, so one likes to imagine that speciation results when one parent species splits in two new species. So, it seems that binary trees must suffice to deal with evolution in a very natural way.

On the other hand, binary trees ladens our work with a preconception: there cannot be multiple speciation, such as it happens in a radiation. The binary tree approach would be to simulate a radiation by a succession of near in time binary splits. Nevertheless, radiations are not intended to be reduced to such a shower, instead they may represent genuine different events, calling for possibly different theoretical insights. So, we must keep an eye into this possibility every time we

see a shower of splittings. In a case like that we must resort to trees in their most general formulation. And, beware, there will be occasions in which the sufficiency of trees will also be questioned. In that case, we must resort to general graphs.

55 ***Reusing Java trees***

The general Java theory about trees that we have seen is directly applicable to binary trees:

56 ***Exercise.*** *Choose one of the various input methods for general trees that are known to us and reuse it to represent the following binary tree and make Java to depict it.* *Answer*

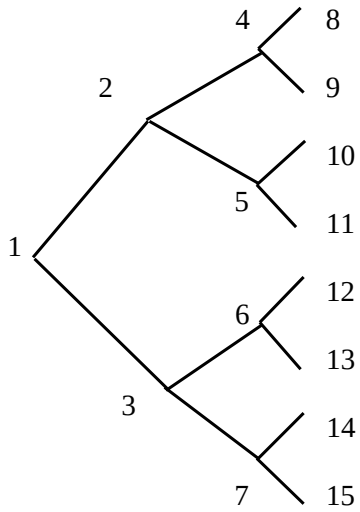


Figure 3.0. A binary tree with numbered nodes.

57 ***Research.*** *The simplicity of binary trees must we dealt with simplicity. So, the very general machinery for Java trees is unnecessary. In consequence, inquire over Internet to see the appropriate classes that come inbuilt in Java for that purpose. Next, developers also like to devise their own appropriate classes and some have offered the code in Internet. Scan the web to see some implementations.*

3.2 Our approach

It sounds very fun to develop our own technology for so nice entities as binary trees. Let us see.

58 ***Our encoding for binary trees***

Let us pursue simplicity to its maximal expression. In that regard, we invite the reader to pay attention to the next form of naming the nodes of a binary tree:

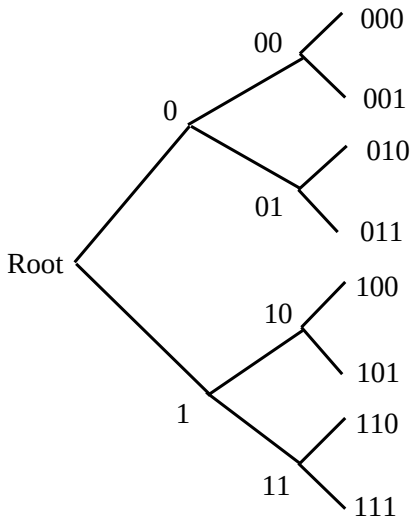


Figure 3.1. Binary numeration of the nodes of a binary tree.

We see that names correspond with binary strings in their full expression, say, 0 is different than 00 and 01 different than 1. From now on, this procedure for numbering the nodes of a binary tree defines their addresses in the tree.

Our notation is transparent: Where are the children of 10? They are in locations 100 and 101. Where is the father of 110? It is in location 11. Where are the brothers of 001? 001 has only one brother that is in position 000.

Our decision: since our notation is so simple and transparent, we need nothing else beyond it to deal with binary trees.

59 ***Our binary notation.*** *If one works with complete binary trees, one needs just to list the name or content of the nodes in the (alphanumeric) order just seen: begin with the name or content of node 0 and end with that of node 1..1. But, it is more realistic to consider the possibility of nodes without children and a lot of others with just one. In attention to those possibilities, we prefer to define our notation as a* ***pairing of the address of a node and its content****. At this moment we need nothing else, so we choose a linear, unidimensional, array with String entries as*

our Java structure to model binary trees. In the next Java code, we see a binary tree with three generations and fourteen nodes -apart from the root. Addresses will be in even position of the array while corresponding contents in the odd ones.

We consider the next example of a tree in our notation:

```
static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
"10", "ba",
"11", "bb",
"000", "aaa",
"001", "aab",
"010", "aba",
"011", "abb",
"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};
```

60 *The following code shows how to make the tree Java Swing package to understand our notation and to draw the corresponding tree.*

```
//Program J60 BinaryInput
/*
 * The representation of binary trees must be
 * as simple as the trees themselves.
 * This program deals with the following notation:
 * each node has an address given by
 * the binary full expansion of its position in the tree,
 * beginning from 0 and 1, the sons of the root,
 * and ending with 1....1,
 * the second son of the last father.
 *
 * The program receives a tree in this binary notation
 * and outputs a drawing of the tree using
 * the Java Tree class (of package Swing).
```

```
 *
 * This is a particular solution for the given example.
 */

import java.awt.BorderLayout;


import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

import javax.swing.tree.TreePath;


//Main class
public class BinaryInput
{
 //This is data for a tree in our binary notation:
 //Even entries are the addresses in the tree of the nodes,
 //odd ones are the names.
static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
"10", "ba",
"11", "bb",
"000", "aaa",
"001", "aab",
"010", "aba",
"011", "abb",
"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};
```

```
private static DefaultMutableTreeNode rootNode =
             new DefaultMutableTreeNode("Root");

    private static DefaultTreeModel treeModel =
                  new DefaultTreeModel(rootNode);

    private static JTree tree = new JTree(treeModel);



 /** Add child to the currently selected node. */
 public DefaultMutableTreeNode addObject(Object child)
  {
   DefaultMutableTreeNode parentNode = null;
   TreePath parentPath = tree.getSelectionPath();

   if (parentPath == null)
    {
     parentNode = rootNode;
    } else
     {
      parentNode = (DefaultMutableTreeNode)
            (parentPath.getLastPathComponent());
     }

   return addObject(parentNode, child, true);
 }



 public static DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
    Object child)
  {
   return addObject(parent, child, true);
  }


 public static DefaultMutableTreeNode
```

```
        addObject(DefaultMutableTreeNode parent,
           Object child, boolean shouldBeVisible)
  {
   DefaultMutableTreeNode childNode =
               new DefaultMutableTreeNode(child);

  if (parent == null)   parent = rootNode;


  // It is key to invoke this on the TreeModel,
  //and NOT DefaultMutableTreeNode
  treeModel.insertNodeInto
    (childNode, parent, parent.getChildCount());

  // Make sure the user can see the lovely new node.
  if (shouldBeVisible)
     tree.scrollPathToVisible
       (new TreePath(childNode.getPath()));

  return childNode;
}


  public static void main(final String args[])
  {
    JFrame frame = new JFrame("Binary Tree in binary notation");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


      //First generation: The sons of the root.
             DefaultMutableTreeNode a =
                      addObject(rootNode,Input[1] );
            DefaultMutableTreeNode b =
                      addObject(rootNode,Input[3] );
     //Second generation
            DefaultMutableTreeNode aa =
                     addObject(a,Input[5] );
            DefaultMutableTreeNode ab =
                     addObject(a,Input[7] );
            DefaultMutableTreeNode ba =
```

```
                addObject(b,Input[9] );
          DefaultMutableTreeNode bb =
                addObject(b,Input[11] );
      //Third and last generation
          addObject(aa,Input[13] );
          addObject(aa,Input[15] );
          addObject(ab,Input[17] );
          addObject(ab,Input[19] );
          addObject(ba,Input[21] );
          addObject(ba,Input[23] );
          addObject(bb,Input[25] );
          addObject(bb,Input[27] );


    new JScrollPane(tree);
    //Pane to draw in
    frame.add(new JScrollPane(tree), BorderLayout.CENTER);
    frame.setSize(400, 500);
    frame.setVisible(true);
  }
} //End of main class J60 BinaryInput
```

61 ***Exercise.*** *Run the previous program and play with the code.*

62 ***Exercise.*** *Generalize the approach of the previous code to get a program that receives whatever input no matter how long it could be if only it is pretty ordered. It also must accept nodes without offspring and nodes with only one son.* *Answer*

63 ***Challenge.*** *Generalize the previous code to get a program that receives whatever input no matter how long or how disordered it could be.*

3.3 Our drawing methodology

Let us investigate the form as we can draw a binary tree if we use our approach and notation.

64 ***Let us draw the structure of a binary tree, just a bush.***

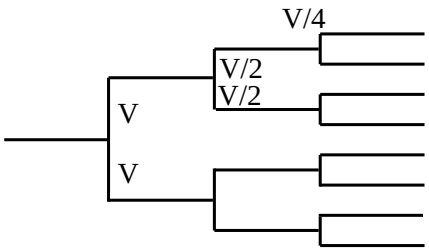


Figure 3.2. We want Java to draw trees like this one.

Moral: A binary tree results from recursion to the pattern of a horizontal U. Horizontal strokes all have the very same length. On its turn, vertical strokes have all the same length over the same generation but the corresponding length for the next generation is divided by two. For this reason the scale must be a power of 2, otherwise approximation problems will be faced. This suffices to instruct Java to draw the structure of a binary tree.

65 ***Let us have Java to draw a bush. The code follows:***

```
//Program J65 BinaryBushDrawing
/*
* The structure of a binary tree is drawn
*
*/

import javax.swing.*;
import java.awt.*;


 public class BinaryBushDrawing extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane;

 private static int numberOfGenerations = 7;
 //Scale must be a power of 2
 private static int scale = 8;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;
```

```
 //Constructor
 public BinaryBushDrawing()
 {
  super(new BorderLayout());
  //Set up the drawing area.
  drawingPane = new DrawingPane();
  //Put the drawing area in a scroll pane.
  JScrollPane scroller = new JScrollPane(drawingPane);
  //scroller.setPreferredSize(new Dimension(700,800));
  add(scroller, BorderLayout.CENTER);
 }



 //The tree is drawn
 public class DrawingPane extends JPanel
  {
   private static final long serialVersionUID = 1L;

  /* a horizontal U  is drawn:
   * center is the center of the vertical stroke,
   * H is the length of the Horizontal stroke.
   * V is the half the length of the vertical stroke
   */
  private void drawU(Graphics g, Dimension center, int H, int V)
  {
int x0 =  center.width;
int y0 =  center.height;
int x1 =  x0;
int y1 =  y0 + V;
g.drawLine(x0, y0, x1,  y1);
g.drawLine(x1, y1, x1 + H,  y1);

y1 =  y0 - V;
g.drawLine(x0, y0, x1,  y1);
g.drawLine(x1, y1, x1 + H,  y1);
  }

  //The binary tree is drawn
```

```
protected void paintComponent(Graphics g)
 {
   super.paintComponent(g);

   g.setColor(Color.red);
   int centerX = 1;
   int centerY = 1;


   //Draw stem
   int x0 =    scale*centerX;
int y0 =   scale* centerY + 200;
int x1 =  x0 + (int) ( scale*horizontalLength);
int y1 =  y0;
  g.drawLine(x0, y0, x1,  y1);

  //Horizontal and basic vertical strokes
  int H = scale * horizontalLength;
int V = scale * verticalLength;

 //For each generation,
 //the upper U is drawn in first place
   Dimension upperCenter = new Dimension(x1,y1);
   Dimension localCenter = new Dimension(0,0);
   for(int j = 1; j < numberOfGenerations + 1; j++)
   {
    //System.out.println("V = " + V);
    //The number of U's in each generation
    //equal half the number of leaves
   double numberOfLeaves =
      Math.pow(2, j);
   int  numberOfU_s = (int) numberOfLeaves /2;
   /*
   System.out.println("Number of leaves = " +
    numberOfLeaves);
   System.out.println("Number of U's = " +
    numberOfU_s);
   */
   localCenter.width = upperCenter.width;
   localCenter.height = upperCenter.height;
```

```
     //The U's are drawn
     for(int i = 0; i < numberOfU_s; i++)
      {
     drawU(g,localCenter, H,V );
     //Center of the U below
        localCenter.height = localCenter.height - 4*V;
      }
     //The center of the upper U of the next generation
     upperCenter.height = upperCenter.height + V;
     upperCenter.width = upperCenter.width + H;
     //Vertical stroke is divided by two
     V = V/2;
     }
   }
}


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane = new BinaryBushDrawing();
   frame.setContentPane(newContentPane);
   frame.setSize(200,300);
   //Display the window.
   frame.setVisible(true);
 }

public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();
```

```
     }
   });
  }
}//End of main class J65 BinaryBushDrawing
```

66 ***Exercise.*** *Run the previous program and play with the code.*

67 ***Let us populate the bush with suitable names.*** *The input is given according to the address - content pairing. The code follows:*

```
//Program J67 BinaryBushDrawing2
/*
* The representation of binary trees must be
* as simple as the trees themselves.
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
* and outputs a drawing of the tree using
* our own methods.
*
*/

import javax.swing.*;


import java.awt.*;




public class BinaryBushDrawing2 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane2;
```

```
 //Scale must be a power of 2
 private static int scale = 16;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
"10", "ba",
"11", "bb",
"000", "aaa",
"001", "aab",
"010", "aba",
"011", "abb",
"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

private static int numberOfNodes = Input.length;
private static int numberOfGenerations =
              Input[numberOfNodes-1].length();

 //Constructor
 public BinaryBushDrawing2()
 {
  super(new BorderLayout());
  //Set up the drawing area.
  drawingPane2 = new DrawingPane2();
```

```
  //Put the drawing area in a scroll pane.
  JScrollPane scroller = new JScrollPane(drawingPane2);
  //scroller.setPreferredSize(new Dimension(700,800));
  add(scroller, BorderLayout.CENTER);
 }



 //The tree is drawn
 public class DrawingPane2 extends JPanel
  {
   private static final long serialVersionUID = 1L;


  /* a horizontal U  is drawn:
   * center is the center of the vertical stroke,
   * H is the length of the Horizontal stroke.
   * V is the half the length of the vertical stroke
   * Up = name of the upper branch
   * Down = name of the lower branch
   */
  private void drawU(Graphics g, Dimension center,
             int H, int V, String Up, String Down)
  {
int x0 =  center.width;
int y0 =  center.height;
int x1 =  x0;
int y1 =  y0 + V;
g.drawLine(x0, y0, x1,  y1);
g.drawLine(x1, y1, x1 + H,  y1);
//String is drawn
g.drawString(Up, x1, y1);


y1 =  y0 - V;
g.drawLine(x0, y0, x1,  y1);
g.drawLine(x1, y1, x1 + H,  y1);
//String is drawn
g.drawString(Down, x1, y1);
  }
```

```
//The binary tree is drawn
protected void paintComponent(Graphics g)
 {
   super.paintComponent(g);

   g.setColor(Color.red);
   int centerX = 1;
   int centerY = 1;
   //The position of first string in Input[]
   int stringPosition = 1;

   //Draw stem
   int x0 =    scale*centerX;
int y0 =   scale* centerY + 400;
int x1 =  x0 + (int) ( scale*horizontalLength);
int y1 =  y0;
  g.drawLine(x0, y0, x1,  y1);

  //Horizontal and basic vertical strokes
  int H = scale * horizontalLength;
int V = scale * verticalLength;

 //For each generation,
 //the upper U is drawn in first place
   Dimension upperCenter = new Dimension(x1,y1);
   Dimension localCenter = new Dimension(0,0);
   for(int j = 1; j < numberOfGenerations + 1; j++)
   {
    //System.out.println("V = " + V);
    //The number of U's in each generation
    //equal half the number of leaves
   double numberOfLeaves =
      Math.pow(2, j);
   int  numberOfU_s = (int) numberOfLeaves /2;
   /*
   System.out.println("Number of leaves = " +
    numberOfLeaves);
   System.out.println("Number of U's = " +
    numberOfU_s);
```

```
     */
     localCenter.width = upperCenter.width;
     localCenter.height = upperCenter.height;
     //The U's are drawn
     for(int i = 0; i < numberOfU_s; i++)
      {
     //drawU(g,localCenter, H,V );
      String Up = Input[stringPosition];
      //Strings are at the odd position of Input
      stringPosition = stringPosition + 2;
      String Down = Input[stringPosition];
      if (stringPosition < numberOfNodes -4 )
           stringPosition = stringPosition + 2;
      drawU(g,localCenter, H,V,Up, Down  );
     //Center of the U below
        localCenter.height = localCenter.height - 4*V;
      }
     //The center of the upper U of the next generation
     upperCenter.height = upperCenter.height + V;
     upperCenter.width = upperCenter.width + H;
     //Vertical stroke is divided by two
     V = V/2;
     }
   }
}


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing2");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane2 = new BinaryBushDrawing2();
   frame.setContentPane(newContentPane2);
   frame.setSize(700,800);
   //Display the window.
   frame.setVisible(true);
 }
```

```
public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
  }
}//End of main class J67 BinaryBushDrawing2
```

68 ***Exercise.*** *Run the previous program and play with the code.*

69 ***Exercise.*** *The tree of the previous program has a polarity up-down. The problem is that what is up according to the code results to appear at the bottom of the display and this is not what one wants to see. Amend the code: up shall mean up in the display and down shall mean down in the display.* *Answer*

70 ***Non evolvable software.*** *The code developed in the previous exercise is good to draw a binary tree and to populate it with strings when every node has its own string. But it is useless to process data in which some nodes have no associated string and so they do not appear in the list. Trying to generalize the code to cover this situation, we developed the following amendments:*

```
//Program J70 BinaryBushDrawing4
/*
* The representation of binary trees must be
* as simple as the trees themselves.
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
```

```
* and outputs a drawing of the tree using
* our own methods.
* The input must be ordered.
*
* The up-down polarity of the drawing is user friendly:
* y -> y - k causes a vertical reflection plus
* a translation.
*
* The present code was developed as a  failing try to
* accept an input with
* incomplete content. So, the complete tree
* is drawn but it lacks corresponding names.
*
*/

import javax.swing.*;


import java.awt.*;




public class BinaryBushDrawing4 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane4;


 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
```

```
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names or contents.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
//"10", "ba",
"11", "bb",
"000", "aaa",
"001", "aab",
"010", "aba",
//"011", "abb",
//"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

 private static int InputLength = Input.length;
 //The number of generation is
 //the length of the last address
 private static int numberOfGenerations =
              Input[InputLength-1].length();
 private static boolean print = true;
 private static boolean print2 = false;
 //Constructor
 public BinaryBushDrawing4()
 {
  super(new BorderLayout());
  //Set up the drawing area.
  drawingPane4 = new DrawingPane4();
  //Put the drawing area in a scroll pane.
  JScrollPane scroller = new JScrollPane(drawingPane4);
  //scroller.setPreferredSize(new Dimension(700,800));
  add(scroller, BorderLayout.CENTER);
 }
```

```
 //The tree is drawn
 public class DrawingPane4 extends JPanel
  {
   private static final long serialVersionUID = 1L;


 //Returns the index   of the binary string s.
 //Take s as a number in base = 2.
 //The index is that number in base 10.
 //The index is a 1 to 1 association
 //of strings of the same length = n
 //and numbers 0,1,..., N = 2^n.
 private   int index( String s)
 {
  int n = s.length();
  char c = ' ';
  int ind = 0;
  int bit = 0;
  for(int i = 0; i <n; i++)
   {
   c = s.charAt(i);
   if (c == '1') bit = 1; else bit = 0;
   ind = (int) (ind + bit*Math.pow(2, n-i-1));
   }
  return ind;
 }

//The fullIndex of s is the position of string s
//in the complete list of all nodes of a binary tree
//in our pairing notation address - content(name).
//fullIndex(s) is the arabic enumeration in the tree of
// s = binary enumeration of a node.
 private int fullIndex(String s)
 {
  int n = s.length();
  int tail = 0;
  for(int k = 0; k < n; k++)
  tail = (int) (tail + Math.pow(2, k));
  return tail + index(s)-1;
```

```
 }



  /* A horizontal U  is drawn:
   * center is the center of the vertical stroke,
   * H is the length of the Horizontal stroke.
   * V is the half the length of the vertical stroke
   * Up = name of the upper branch
   * Down = name of the lower branch
   *
   */
  private void drawU(Graphics g, Dimension center,
             int H, int V, String Up, String Down)
  {
int x0 =  center.width;
int y0 =  center.height;
int x1 =  x0;
int y1 =  y0 + V;
g.drawLine(x0, verticalSize -y0, x1, verticalSize- y1);
g.drawLine(x1, verticalSize - y1,
   x1 + H,  verticalSize - y1);
//String is drawn
g.drawString(Up, x1 + 1, verticalSize - y1-1);


y1 =  y0 - V;
g.drawLine(x0, verticalSize - y0,
   x1,  verticalSize - y1);
g.drawLine(x1, verticalSize - y1,
   x1 + H,  verticalSize - y1);
//String is drawn
g.drawString(Down, x1 + 1, verticalSize - y1-1);
  }

  private void printInform(String a, int b, int c)
  //The binary tree is draw
  {
   System.out.println(a +  " -> "
```

```
                    + b);
 System.out.println(" positionInTree =  "  + c);
}

protected void paintComponent(Graphics g)
 {
   super.paintComponent(g);
   if (print)
    {
     //Test for the full index
     System.out.println("test for the fullIndex");
     for(int v =  0; v < InputLength; v= v+2)
     System.out.println(Input[v] +  " "
       + fullIndex(Input[v]) + " " + Input[v+1]);
    }
   g.setColor(Color.red);
   int centerX = 1;
   int centerY = 1;
   //Input[] has the addresses and names = contents
   int addressPosition = 0;
   //Address in binary notation at Input[];
   String addressBinNotation = Input[addressPosition];
   //Arabic notation of address in the
   //full tree of listed node
   int arabicAddressAtInput = fullIndex( addressBinNotation);
   //The position of   string in Input[]
   int stringPosition = addressPosition + 1;
   //Actual position in the tree
   int positionInTree = 0;

   //Draw stem
   int x0 =    scale*centerX;
int y0 =   scale* centerY + 400;
int x1 =  x0 + (int) ( scale*horizontalLength);
int y1 =  y0;
  g.drawLine(x0, verticalSize - y0,
      x1,  verticalSize - y1);

  //Horizontal and basic vertical strokes
  int H = scale * horizontalLength;
```

```
int V = scale * verticalLength;

 //For each generation,
 //the upper U is drawn in first place
   Dimension upperCenter = new Dimension(x1,y1);
   Dimension localCenter = new Dimension(0,0);
   for(int j = 1; j < numberOfGenerations + 1; j++)
   {
    //System.out.println("V = " + V);
    //The number of U's in each generation
    //equal half the number of leaves
   double numberOfLeaves =
      Math.pow(2, j);
   int  numberOfU_s = (int) numberOfLeaves /2;
   /*
   System.out.println("Number of leaves = " +
    numberOfLeaves);
   System.out.println("Number of U's = " +
    numberOfU_s);
   */
   localCenter.width = upperCenter.width;
   localCenter.height = upperCenter.height;
   //The U's are drawn
   for(int i = 0; i < numberOfU_s; i++)
    {
   String Up = "NothingUp";
   String Down = "NothingDown";
   System.out.println();

      if (print2) printInform(addressBinNotation,
    arabicAddressAtInput, positionInTree);
   //if listed node corresponds with  the current one
   //or with the next in a U,
   //the content must be added to the drawing.
   if ( (arabicAddressAtInput == positionInTree) ||
     (arabicAddressAtInput == positionInTree + 1))
    {
     //Upper strings come in even nodes.

   int remainer = arabicAddressAtInput % 2;
```

```
//If arabicAddressAtInput is even:
  if (remainer == 0 )
   {
  if (print) System.out.println("Even");
 Up = Input[stringPosition];
 if (print) System.out.println("Up = " + Up);
//Is there a string for the lower stroke?
 addressPosition = addressPosition + 2;
 addressBinNotation = Input[addressPosition];
 arabicAddressAtInput =
 fullIndex(addressBinNotation);
     if (print2) printInform(addressBinNotation,
   arabicAddressAtInput, positionInTree);

 positionInTree++;
 if (print2) printInform(addressBinNotation,
          arabicAddressAtInput, positionInTree);
 if (arabicAddressAtInput == positionInTree)
     {
  Down = Input[stringPosition+2];
  if (print2)
  System.out.println("Down = " + Down);
  addressPosition = addressPosition + 2;
         if (print2) printInform(addressBinNotation,
         arabicAddressAtInput, positionInTree);
     }
   }
  else //if arabicAddressAtInput is odd
   {
 if (print) System.out.println("Odd");
Down = Input[stringPosition];
 if (print)
 System.out.println("Down = " + Down);
    addressPosition = addressPosition + 2;
    positionInTree = positionInTree + 1;

   }//EndOf  if (remainer == 0 )

 }
```

```
      //The U is drawn
      drawU(g,localCenter, H,V,Up, Down  );
      //data for the next U
      if (addressPosition < InputLength  )
      {
       addressBinNotation = Input[addressPosition];
       arabicAddressAtInput =
              fullIndex( addressBinNotation);
       stringPosition = addressPosition + 1;
       positionInTree = positionInTree +1;
          if (print) printInform(addressBinNotation,
        arabicAddressAtInput, positionInTree);
      }


     //Center of the U below
        localCenter.height = localCenter.height - 4*V;

      }//End of drawing of U
     //Data for the next generation
     //The center of the upper U of the next generation
     upperCenter.height = upperCenter.height + V;
     upperCenter.width = upperCenter.width + H;
     //Vertical stroke is divided in two
     V = V/2;
     }
   }
}



 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing4");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane4 = new BinaryBushDrawing4();
   frame.setContentPane(newContentPane4);
```

```
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);
 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
  }
}//End of main class J70 BinaryBushDrawing4
```

71 ***Exercise.*** *Run the previous program and play with the code. Play enough to show that the program is broken by data Input with enough vacuums. Challenge: made the necessary amendments that the Author could not.*

72 ***Discussion.*** *Experience shows that software development is just a walk along and evolutionary process described by a simple dictum: there is no software development without huge amount of bugs whose correction creates more bugs. The ensuing work is extremely heavy but there are forms as it could be made still harder: when that happens one says that one is producing not evolvable software. Actually, one dreams of evolvable software as one that can be easily modified with easy to correct bugs. Now, how can one know where one has non evolvable software?*

The case of the previous code shows one instance: there are two tasks. The first one is to draw a bush, the second to add names to predefined nodes. The previous code try to solve the two problems at the same time with intertwining code. The result is an intractable, unmodifiable mesh. The solution is to resort to the function paradigm of software: problems must be divided in simple tasks that must be solved by exclusive methods. Or: one simple task is solved by one exclusive method with specified input and only one output and without global variables (see Vol II and V).

So, the invitation is to face the posed problem with an approach in which drawing the bush and drawing the names of nodes are treated as two separated problems that must be solved by two separated methods.

73 ***Oversight.*** *We know that the genome is software. The astonishing and permanent advances in genetic engineering clearly show that the genome is evolvable. We also know that evolution is a software developer. Is the evolvability of the genome the very reason of considering evolution as a possibly correct explanation of the existence of many species? Certainly not: evolvability is not needed for surviving or reproduction. Unless you filter genomes for evolvability, evolution will produce messy software whose evolvability would be extremely limited, in general. That is another sufficient reason because of which the evolutionary theory of the origin of species is obviously false. Nevertheless, the intuition of the Author adheres to the belief that evolution is the correct explanation of the origin of some species, say of flies.*

But if the evolutionary theory is false, how are we going to explain our existence? One might resort to one of the many versions of creationism. These insights are paradigms in their own and in general are outside the scope of science. Nevertheless, a doctrine to be accepted by people must have some parts that agree with common sense and that one can see in our daily life. These parts are subjected to the scrutiny of science. For instance: many forms of creationism assure that the Creator is smart. Now, intelligence is a fundamental quality that we permanently use in our ordinary life. So, a question immediately arises: how are we going to detect intelligence in the genome, if any? The problem with this question is that it is ill: evolution is a reality and it classifies as software developer. So, evolution is intelligent. Therefore, the question to be correct must take a form like this: How are we going to detect a sign of intelligence in the genome that cannot be explained by natural evolution?

Our proposal is to extrapolate to biology what we see here in our work:

We faced the problem of making a drawing with the correct up-down polarity. But we had a problem: the polarity inherited by the code from natural numbers is opposite to that of the display when it is looked by a human being. The solution is very simple: we need a reflection, to upright the polarity, and then a translation to shift the figure appropriately. This amounts to coherent changes in the code that a posteriori can suggest mathematical guidance and an intelligent creator. Such a suggestion to get some interest must be reinforced by many similar cases. In particular, we recall that we have used in volume V some sophisticated techniques

of numerical analysis to enrich our programs and simulations. Would anyone dare to claim that evolution is capable of doing things like that and in the same style? By contrast, if someone wants to make a scientific point of his or her belief in evolution, he or she just needs to show a huge amount of intermediate solutions filled in bugs and that end at a wonderful solution written in a very clean style.

74 ***A new solution*** *to the problem of adding strings to some but specific nodes can be seen in the next code. Two posed tasks are solved separately one from the other. To understand our rule for interpreting addresses, think of the address 0110. It says: go over the tree up, down, down and up, 4 generations in all. The code follows:*

```
//Program J74 BinaryBushDrawing5
/*
* The representation of binary trees must be
* as simple as the trees themselves.
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
* and outputs a drawing of the tree using
* our own methods.
* The input must be ordered.
*
* The up-down polarity of the drawing is user friendly:
* y -> y - k causes a vertical reflection plus
* a translation.
*
* The present code
* accepts an input with
* incomplete content. So, the complete tree
* is drawn but some nodes may lack
* corresponding strings.
* To achieve this goal in a transparent form,
* tasks are kept separated.
```

```
*
*/

import javax.swing.*;


import java.awt.*;


public class BinaryBushDrawing5 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane5;


 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names or contents.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
//"10", "ba",
"11", "bb",
//"000", "aaa",
//"001", "aab",
"010", "aba",
//"011", "abb",
```

```
//"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

 private static int InputLength = Input.length;
 //The number of generation is
 //the length of the last address
 private static int numberOfGenerations =
              Input[InputLength-1].length();

 //Constructor
 public BinaryBushDrawing5()
 {
  super(new BorderLayout());
  //Set up the drawing area.
  drawingPane5 = new DrawingPane5();
  //Put the drawing area in a scroll pane.
  JScrollPane scroller = new JScrollPane(drawingPane5);
  //scroller.setPreferredSize(new Dimension(700,800));
  add(scroller, BorderLayout.CENTER);
 }



 //The tree is drawn
 public class DrawingPane5 extends JPanel
  {
   private static final long serialVersionUID = 1L;




 /* A horizontal U  is drawn:
  * center is the center of the vertical stroke,
  * H is the length of the Horizontal stroke.
  * V is the half the length of the vertical stroke
  * Up = name of the upper branch
  * Down = name of the lower branch
```

```
  *
  */
 private void drawU(Graphics g, Dimension center,
         int H, int V)
{
int x0 =  center.width;
int y0 =  center.height;
int x1 =  x0;
int y1 =  y0 + V;
g.drawLine(x0, verticalSize -y0, x1, verticalSize- y1);
g.drawLine(x1, verticalSize - y1,
   x1 + H,  verticalSize - y1);


y1 =  y0 - V;
g.drawLine(x0, verticalSize - y0,
   x1,  verticalSize - y1);
g.drawLine(x1, verticalSize - y1,
   x1 + H,  verticalSize - y1);

}




  protected void paintComponent(Graphics g)
   {
     super.paintComponent(g);

     g.setColor(Color.red);
     int centerX = 1;
     int centerY = 1;


     //Draw stem
     int x0 =    scale*centerX;
  int y0 =   scale* centerY + 400;
  int x1 =  x0 + (int) ( scale*horizontalLength);
  int y1 =  y0;
    g.drawLine(x0, verticalSize - y0,
```

```
        x1,  verticalSize - y1);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
  int V = scale * verticalLength;

  //The bush is drawn:

   //For each generation,
   //the upper U is drawn in first place
     Dimension upperCenter = new Dimension(x1,y1);
     Dimension localCenter = new Dimension(0,0);
     for(int j = 1; j < numberOfGenerations + 1; j++)
     {
      //The number of U's in each generation
      //equal half the number of leaves
     double numberOfLeaves =
        Math.pow(2, j);
     int  numberOfU_s = (int) numberOfLeaves /2;
     localCenter.width = upperCenter.width;
     localCenter.height = upperCenter.height;
     //The U's are drawn
     for(int i = 0; i < numberOfU_s; i++)
      {
     drawU(g,localCenter, H,V );
     //Center of the U below
        localCenter.height = localCenter.height - 4*V;
      }
     //The center of the upper U of the next generation
     upperCenter.height = upperCenter.height + V;
     upperCenter.width = upperCenter.width + H;
     //Vertical stroke is divided by two
     V = V/2;
     }  //End of drawing of tree


    //Strings are drawn:

     String path = "";
     char direction;
```

```
    for(int i = 0; i < InputLength; i = i+2)
     {
      //Addresses will be used as paths to the places
      //where strings must be drawn
      path = Input[i];
      String s = Input[i+1];
      //Reinitialization
      Dimension center = new Dimension(x0,y0);
      Dimension target = center;
      V = scale * verticalLength;
      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
     g.drawString(s, target.width + 1,
                    verticalSize - target.height-1);
     }
   }
}//End of   paintComponent



 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing5");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane5 = new BinaryBushDrawing5();
```

```
   frame.setContentPane(newContentPane5);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);
 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
  }
} //End of main class J74 BinaryBushDrawing5
```

75 ***Exercise.*** *Run the previous program and play with the code. Play enough to agree else disagree with the appreciation of the Author: no input can break the code so, it is robust, and moreover, the part dealing with the drawing of strings is exquisitely transparent.*

76 ***From U's to paths.*** *We have used recursion over U's to draw a tree. But the transparent interpretation of addresses that we looked at in the previous program poses a challenge: to use the same interpretation to draw the bush. An implementation of this idea follows:*

```
//Program J76 BinaryBushDrawing6
/*
* The representation of binary trees must be
* as simple as the trees themselves.
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
```

```
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
* and outputs a drawing of the tree using
* our own methods.
* The input must be ordered.
*
* The up-down polarity of the drawing is user friendly:
* y -> y - k causes a vertical reflection plus
* a translation.
*
* The present code
* accepts an input with
* incomplete content. So, the complete tree
* is drawn but some nodes may lack
* corresponding strings.
*
* We rely on a natural interpretation of addresses:
* Address in the tree 0110 means:
* go  up, down, down and up, 4  generations in all.
*
*/



import javax.swing.*;


import java.awt.*;


public class BinaryBushDrawing6 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane6;


 //Vertical  size of the panel
 private static int verticalSize = 800;
```

```
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names or contents.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
//"10", "ba",
"11", "bb",
//"000", "aaa",
//"001", "aab",
"010", "aba",
//"011", "abb",
//"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

 private static int InputLength = Input.length;
 //The number of generation is
 //the length of the last address
 private static int inputhLength =
              Input[InputLength-1].length();

 private static int numberOfGenerations = 5;
 //The terminal  leaves define the complete tree
 private static int numberOfLeaves =
   (int) Math.pow(2, numberOfGenerations);
 //Binary paths to leaves
```

```
private static String[] pathsToLeaves =
                    new String[numberOfLeaves];

//Constructor
public BinaryBushDrawing6()
{
 super(new BorderLayout());
 //Set up the drawing area.
 drawingPane6 = new DrawingPane6();
 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane6);
 //scroller.setPreferredSize(new Dimension(700,800));
 add(scroller, BorderLayout.CENTER);
}



//The tree is drawn
public class DrawingPane6 extends JPanel
 {
  private static final long serialVersionUID = 1L;


  //Paths to leaves are constructed as binary strings
  private void createPathsToLeaves(int numberOfGenerations)
  {
String s;
for(Integer i = 0; i < numberOfLeaves; i++ )
{
 s  = Integer.toBinaryString(i);
 int l = s.length();
 if (l < numberOfGenerations)
 for(int k = l; k < numberOfGenerations; k++)
 s = "0" + s;
 pathsToLeaves[i] = s;
 System.out.println(pathsToLeaves[i]);
}
  }
```

```
//Drawing machinery
protected void paintComponent(Graphics g)
 {
   super.paintComponent(g);
   createPathsToLeaves(numberOfGenerations);
   g.setColor(Color.red);
   //Symbolic initialization
   int centerX = 1;
   int centerY = 1;


   //Draw stem
   int x00 =    scale*centerX;
int y00 =   scale* centerY + 400;
int x11 =  x00 + (int) ( scale*horizontalLength);
int y11 =  y00;
  g.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);

  //Horizontal and basic vertical strokes
  int H = scale * horizontalLength;
int V = scale * verticalLength;


  String path = "";
  char direction;

//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
   for(int i = 0; i < numberOfLeaves; i++)
   {
    //Addresses will be used as paths to  be drawn
    path = pathsToLeaves[i];

    //Reinitialization
    Dimension center = new Dimension(x11,y11);

    V = scale * verticalLength;
```

```
//Path is interpreted
  for(int j = 0; j < path.length(); j++)
   {
 direction = path.charAt(j);
 if (direction == '0') //go up
  {
 int x0 =  center.width;
 int y0 =  center.height;
 int x1 =  x0;
 int y1 =  y0 + V;
 //Draw vertical stroke
 g.drawLine(x0, verticalSize -y0, x1,
 verticalSize- y1);
 //Draw horizontal stroke
 g.drawLine(x1, verticalSize - y1,
    x1 + H,  verticalSize - y1);
 center.height = center.height + V;
  }
 else // if direction == '1'  =  go down
  {
 int x0 =  center.width;
 int y0 =  center.height;
 int x1 =  x0;

 int y1 =  y0 - V;
 //Draw vertical stroke
 g.drawLine(x0, verticalSize - y0,
    x1,  verticalSize - y1);
 //Draw horizontal stroke
 g.drawLine(x1, verticalSize - y1,
    x1 + H,  verticalSize - y1);
 center.height = center.height - V;
  } //End of if + else

 center.width = center.width + H ;

 V = V/2;
   }//End of for j
```

```
     }  //End of for: numberOfLeaves


    //Strings are drawn:


    for(int i = 0; i < InputLength; i = i+2)
     {
      //Addresses will be used as paths to the places
      //where strings must be drawn
      path = Input[i];
      String s = Input[i+1];
      //Reinitialization
      Dimension center = new Dimension(x00,y00);
      Dimension target = center;
      V = scale * verticalLength;
      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
     g.drawString(s, target.width + 1,
                     verticalSize - target.height-1);
     }
   }
}//End of   paintComponent



 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
```

```
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing6");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane6 = new BinaryBushDrawing6();
   frame.setContentPane(newContentPane6);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);
 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
  }
} //End of main class J76 BinaryBushDrawing6
```

77 ***Exercise.*** *Run the previous program and play with the code.*

78 ***Exercise.*** *Improve the style of the previous program. Say, there is an extremely long method. Divide it in two or more pieces, each one with a clear function.* *Answer*

79 ***Exercise.*** *We have defined binary trees as those trees in which each node has a maximum of two children. But all our trees have had exactly 2 children. Enable our drawing machinery to deal with one or no children. In that way, we will be able of simulating evolution in a more interesting way.* *Answer*

3.4 Simulating evolution

Let us test our machinery to see how well it deals with the simulation of the **mutation-model** of evolution (mutation and nothing else).

80 ***The code for a simulation of evolution restricted to binary trees follows.*** *We merge appropriately program, J51 RecordingTree, which records evolution over a Java Tree, with program J78 BinaryBushDrawing7, which has our encoding rules and the drawing machinery.*

```
//Program J80 BinaryEvo
/*
* We run the mutation-model
* restricted over a binary tree.
* We follow our own encoding rules
* and we use our own drawing machinery.
*
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Random;


public class BinaryEvo extends JPanel
                            implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane8;


 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 2;
 private static int verticalLength = 10;
```

```
 //Number of reproductive cycles
 private static int numberOfGenerations = 3;


//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[1000000];
 //Last used position  in Record[]
 private static int lastUsedPosition;


 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";



 private static   String   seed =
    "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   = Alphabet.length();

 private static Random r = new Random();
 //Mutation rate.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;
```

```
//Constructor
public BinaryEvo()
{
 super(new BorderLayout());

 //The tree is cleared out.
 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);
 clearButton.setVisible(true);

 //Set up the drawing area.
 drawingPane8 = new DrawingPane8();
 drawingPane8.add(clearButton);
 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane8);
 //scroller.setPreferredSize(new Dimension(700,800));

 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
  String command = e.getActionCommand();
  //If reset button is clicked over
  if  (RESET_COMMAND.equals(command))
  {
    evolution();
    repaint();
  }
}


//======Evolution====

//The string individual is subjected to mutation
```

```
//with mutation rate m per site per generation
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
 if (p<m)  //mutation
   { //a random integer number
    int l = r.nextInt(lengthAlf);
    //A random char
    char c = Alphabet.charAt(l);
    mutant = mutant + c;
   }
  else
   { //no mutation
  mutant = mutant + individual.charAt(i);
   }
}
return mutant;
}


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
 {
  String fatherPath;
  String childPath;
  //Reproductive leaves are the nodes at the end of Record[]
  //with the max length.
  String lastLeaf = Record[lastUsedPosition-1];
  int maxLength = lastLeaf.length();
for(int i = lastUsedPosition-1; i >= 0; i = i-2)
   {
//Nodes go at even entries
fatherPath = Record[i];
System.out.println("Nodee = " + i  + "  " + fatherPath);
//If node is a reproductive leaf
```

```
 if( fatherPath.length() == maxLength )
   {
//The number of children is within
//0 and 2;
    int actualNumberOfChildren =
     r.nextInt(maxNumbChildren + 1);
    System.out.println("Number of children = "
      + actualNumberOfChildren) ;
    //Leaf undergoes reproduction
    if (actualNumberOfChildren > 0)
    {
     String direction; //0 = up; 1 = down
     int t = r.nextInt(2);
     if ( t == 0 ) direction = "0";
     else direction = "1";
     childPath = fatherPath + direction;
     lastUsedPosition++;
     Record[lastUsedPosition] = childPath;
     System.out.println("Path first = " + childPath);
     lastUsedPosition++;
     Record[lastUsedPosition] = mutation(Record[i+1]);
     if (actualNumberOfChildren == 1)
  {
      //If a second child exists,
      //its direction is contrary to that of the first
      if ( t == 0 ) direction = "1";
     else direction = "0";
      childPath = fatherPath + direction;
     System.out.println("Path second = " + childPath);
     lastUsedPosition++;
     Record[lastUsedPosition] = childPath;
     lastUsedPosition++;
     Record[lastUsedPosition] = mutation(Record[i+1]);

      }
    }
//System.out.println(node.toString()) ;
   }

    }
```

```
  }


   //An evolutionary process under the
   //mutation-model is triggered.
   //Every leaf (node of the last generation) is subjected
   //to reproduction with a number of children
   //equal or less than 2.
   //Reproduction may undergo mutation in each site.
   private static void evolution()
   {
    //Initialization of node 0: path goes up
Record[0] = "0";
//The string of node 0
Record[1] = seed;
//Initialization of node 1: path goes down
    Record[2] = "1";
    //The string of node 1
Record[3] = seed;
    //Last used position in Record[]
    lastUsedPosition = 3;
  //Evolution is run for some reproductive cycles
//Evolution = the mutation-model = just mutation

  for(int gen = 0; gen < numberOfGenerations; gen++)
   {
    System.out.println("gen = " + gen);
  reproductionOfLeaves();
     }
   }

 //===== Inner class with the drawing machinery =======


 public class DrawingPane8 extends JPanel
  {
   private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
```

```
int centerY = 1;
//Coordinates of the origin
int x00 =    scale*centerX;
int y00 =   scale* centerY + 400;
//Target coordinates
int x11 =  x00 + (int) ( scale*horizontalLength);
int y11 =  y00;
//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V = scale * verticalLength;
//Addresses are interpreted as paths
String path = "";
char direction;

//Tree is drawn
private void drawTree(Graphics g)
 {
//The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
     for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to  be drawn
      path = Record[i];
      //Reinitialization
      Dimension center = new Dimension(x11,y11);
      V = scale * verticalLength;
       //Path is interpreted
         for(int j = 0; j < path.length(); j++)
          {
        direction = path.charAt(j);
        int x0 =  center.width;
    int y0 =  center.height;
    int x1 =  x0;
    int y1;
        if (direction == '0') //go up
         {
          y1 =  y0 + V;
          center.height = center.height + V;
         }
```

```
        else // if direction == '1'  =  go down
         {
          y1 =  y0 - V;
          center.height = center.height - V;
         } //End of if + else
      //Draw vertical stroke
    g.drawLine(x0, verticalSize -y0, x1,
    verticalSize- y1);
    //Draw horizontal stroke
    g.drawLine(x1, verticalSize - y1,
       x1 + H,  verticalSize - y1);
        //Move center
        center.width = center.width + H ;
        //Vertical stroke is divided in two
        V = V/2;
       }//End of for j
     }  //End of for: numberOfLeaves
 }


//Strings are drawn
private void drawStrings(Graphics g)
{
    //Strings are drawn:
    for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to the places
      //where strings must be drawn
      path = Record[i];
      String s = Record[i+1];
      //Reinitialization
      Dimension center = new Dimension(x00,y00);
      Dimension target = center;
      V = scale * verticalLength;
      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
```

```
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
     g.drawString(s, target.width + 1,
                    verticalSize - target.height-1);
     }
   }


  protected void paintComponent(Graphics g)
   {
     super.paintComponent(g);
     g.setColor(Color.red);
     //Draw stem
    g.drawLine(x00, verticalSize - y00,
        x11,  verticalSize - y11);
    drawTree(g);
    drawStrings(g);
}//End of   paintComponent
  }


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryEvo");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent BinaryEvo = new BinaryEvo();
   frame.setContentPane(BinaryEvo);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);

 }
```

```
public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();
      evolution();
     }
   });
 }
} //End of main class J80 BinaryEvo
```

81 ***Exercise.*** *Run the previous program and play with the code.*

82 ***Exercise.*** *Improve the code. First: divide that very long method in at least two parts. Second: the parameter* `numberOfgenerations` *is equal to, say, 3, but trees with 4 generations appear. Correct this inconsistency. Third: the code really runs in parallel two simulations of evolution: run just one.* *Answer*

83 ***Research.*** *If one changes the number of generations in the last program, one must accordingly change the vertical stroke of the graphic. A manner to automatically do this must exit. Find one and implement it.*

84 ***Recombination.*** *Let us add the possibility of recombination to the* ***mutation-model*** *of evolution. In the following code, evolution is restricted to a binary tree.*

```
//Program J84 BinaryEvo3
/*
* We run the mutation-model  of evolution
* restricted over a binary tree.
* It includes mutation  and recombination.
*
* We follow our own encoding rules
* and we use our own drawing machinery.
*
*/
```

```
import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Random;


public class BinaryEvo3 extends JPanel
                            implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane10;

 //Number of reproductive cycles
 private static int numberOfGenerations = 5;

 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 2;
 //for 3 generations, vertical length = 6;
//for 4 generations, vertical length = 10;
//for 5 generations, vertical length = 14;
//for 6 generations, vertical length = 25;
//for 7 generations, vertical length = 48;
 private static int verticalLength = 14;

 //Maximal number of nodes in the tree
 private static int N = 10000;
//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
```

```
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[2*N];



 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
    "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   = Alphabet.length();

 private static Random r = new Random();
 //Mutation rate.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;


 //Constructor
 public BinaryEvo3()
 {
  super(new BorderLayout());

  //The tree is cleared out.
  JButton clearButton = new JButton("Reset");
  clearButton.setActionCommand(RESET_COMMAND);
  clearButton.addActionListener(this);
  clearButton.setVisible(true);

  //Set up the drawing area.
  drawingPane10 = new DrawingPane10();
  drawingPane10.add(clearButton);
  //Put the drawing area in a scroll pane.
```

```
 JScrollPane scroller = new JScrollPane(drawingPane10);
 //scroller.setPreferredSize(new Dimension(700,800));

 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
  String command = e.getActionCommand();
  //If reset button is clicked over
  if  (RESET_COMMAND.equals(command))
  {
    evolution();
    repaint();
  }
}


//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
 if (p<m)  //mutation
   { //a random integer number
    int l = r.nextInt(lengthAlf);
    //A random char
    char c = Alphabet.charAt(l);
    mutant = mutant + c;
   }
  else
   { //no mutation
```

```
   mutant = mutant + individual.charAt(i);
    }
 }
 return mutant;
 }

 //Reproduction

 private static String fatherPath;
 private static String childPath;
 private static   int actualNumberOfChildren;
 //Last used position  in Record[]
 private static int lastUsedPosition;



 //Number of reproductive leaves,
 //those leaves of the last generation
 private static int numberOfLeaves = N;
 //Position of leaves in Record[]
 private static  int[] leaves = new int[N];

 //Output leaves and their number
 public static void findLeaves()
 {
  int counter = 0;
  //Reproductive leaves are the nodes at the end of Record[]
  //with the max length.
  String lastLeaf = Record[lastUsedPosition-1];
  int maxLength = lastLeaf.length();
  for(int i = lastUsedPosition-1; i >= 0; i = i-2)
  {
 //Nodes go at even entries
 fatherPath = Record[i];
//System.out.println("Node = " + i  + "  " + fatherPath);
 //If node is a reproductive leaf
 if( fatherPath.length() == maxLength )
   {
leaves[counter] = i;
counter++;
```

```
//System.out.println("Leaf = " + Record[i]);
   }

  }
  numberOfLeaves = counter;
 }

 //Flag for printing recombination
 private static  boolean print= true;

//Record[i+1], the string associates to node i,
//is recombined
//with another chosen at random leaf
public static String recombination(int i)
{
 //String to be recombined
String s = Record[i+1];
String  s1= "";
int recSite = 0;
int m= s.length();
if (m>0)
{
 recSite = r.nextInt(m) ;
 //Recombinant part of s
 s1= s.substring(0,recSite);
}
//A leaf is chosen at random
int k = r.nextInt(numberOfLeaves);
String t = Record[leaves[k]+1];
//Recombinant part of t
String  t2=  t.substring(recSite);
//Recombination
String output = s1+t2;
if (print)
 {
  System.out.println("Recombination");
  System.out.println("recSite = " + recSite);
  System.out.println(s);
  System.out.println(t);
  System.out.println(output);
```

```
}
return output;
}


 //Node at position i is reproduced.
 //Reproduction involves mutation and recombination
 public static void reproduce(int i)
 {
 String direction; //0 = up; 1 = down
     int t = r.nextInt(2);
     if ( t == 0 ) direction = "0";
     else direction = "1";
     childPath = fatherPath + direction;
     lastUsedPosition++;
     Record[lastUsedPosition] = childPath;
     //System.out.println("Path first = " + childPath);
     lastUsedPosition++;
   //The string associated with node i is recombined
     String s = recombination(i);
     //and then passed under mutation
     Record[lastUsedPosition] = mutation(s);
     //Must a second child exist?
     if (actualNumberOfChildren == 2)
  {
      //If a second child exists,
      //its direction is contrary to that of the first
      if ( t == 0 ) direction = "1";
     else direction = "0";
      childPath = fatherPath + direction;
     //System.out.println("Path second = " + childPath);
     lastUsedPosition++;
     Record[lastUsedPosition] = childPath;
     lastUsedPosition++;
     //The string associated with node i is recombined
     s = recombination(i);
     //and then passed under mutation
     Record[lastUsedPosition] = mutation(s);
      }
```

```
 }


 //Terminal leaves, nodes of the last generation, are found
 //and children are generated.
 private static void reproductionOfLeaves()
  {
   findLeaves();
 for(int i = 0; i < numberOfLeaves; i++)
    {
 int position = leaves[i];
 //Nodes go at even entries
 fatherPath = Record[position];
 //The number of children is within
 //0 and 2;
 actualNumberOfChildren =
     r.nextInt(maxNumbChildren + 1);
    /*
    System.out.println("Number of children = "
      + actualNumberOfChildren) ; */
    //Leaf undergoes reproduction
 if (actualNumberOfChildren > 0)
      reproduce(position);
//System.out.println(node.toString()) ;
    }
  }


   //An evolutionary process under the
   //mutation-model is triggered.
   //Every leaf (node of the last generation) is subjected
   //to reproduction with a number of children
   //equal or less than 2.
   //Reproduction may undergo mutation in each site.
   private static void evolution()
   {
    //Initialization of node 0: path goes up
Record[0] = "0";
//The string of node 0
Record[1] = mutation(seed);
```

```
//Initialization of node 1: path goes down
    Record[2] = "1";
    //The string of node 1
    Record[3] =  mutation(seed);
    //Last used position in Record[]
    lastUsedPosition = 3;
  //Evolution is run for some reproductive cycles
//Evolution = the mutation-model = just mutation.
  //Generation 0 was made by hand in the initilization
  for(int gen = 1; gen < numberOfGenerations; gen++)
   {
    //System.out.println("gen = " + gen);
    reproductionOfLeaves();
   }
   }

 //===== Inner class with the drawing machinery =======


 public class DrawingPane10 extends JPanel
  {
   private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00 =    scale*centerX;
int y00 =   scale* centerY + 400;
//Target coordinates
int x11 =  x00 + (int) ( scale*horizontalLength);
int y11 =  y00;
//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V = scale * verticalLength;
//Addresses are interpreted as paths
String path = "";
char direction;

//Tree is drawn
```

```
private void drawTree(Graphics g)
 {
//The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
     for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to  be drawn
      path = Record[i];
      //Reinitialization
      Dimension center = new Dimension(x11,y11);
      V = scale * verticalLength;
       //Path is interpreted
         for(int j = 0; j < path.length(); j++)
          {
        direction = path.charAt(j);
        int x0 =  center.width;
    int y0 =  center.height;
    int x1 =  x0;
    int y1;
        if (direction == '0') //go up
         {
          y1 =  y0 + V;
          center.height = center.height + V;
         }
        else // if direction == '1'  =  go down
         {
          y1 =  y0 - V;
          center.height = center.height - V;
         } //End of if + else
      //Draw vertical stroke
    g.drawLine(x0, verticalSize -y0, x1,
    verticalSize- y1);
    //Draw horizontal stroke
    g.drawLine(x1, verticalSize - y1,
       x1 + H,  verticalSize - y1);
        //Move center
        center.width = center.width + H ;
        //Vertical stroke is divided in two
        V = V/2;
```

```
       }//End of for j
     }  //End of for: numberOfLeaves
 }


//Strings are drawn
private void drawStrings(Graphics g)
{
//Seed is drawn
g.drawString(seed,x00,verticalSize - y00);
//Strings are drawn:
    for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to the places
      //where strings must be drawn
      path = Record[i];
      String s = Record[i+1];
      //Reinitialization
      Dimension center = new Dimension(x00,y00);
      Dimension target = center;
      V = scale * verticalLength;
      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
     g.drawString(s, target.width + 1,
                    verticalSize - target.height-1);
     }
   }
```

```
  protected void paintComponent(Graphics g)
   {
     super.paintComponent(g);
     g.setColor(Color.red);
     //Draw stem
    g.drawLine(x00, verticalSize - y00,
        x11,  verticalSize - y11);
    drawTree(g);
    drawStrings(g);
}//End of   paintComponent
  }


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryEvo");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent BinaryEvo3 = new BinaryEvo3();
   frame.setContentPane(BinaryEvo3);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);

 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();
      evolution();
     }
   });
```

```
 }
} //End of main class J84 BinaryEvo3
```

85 ***Exercise.*** *Run the previous program and play with the code.*

86 ***Sexual dimorphism.*** *One might include at a very low cost sexual dimorphism in our programs if one interprets a path as an indicative of sex. For instance, by assigning 0 to a female and 1 to a male and establishing that males recombine with females but only females reproduce. So, the path 00001 means that a male was born in the fifth generation. Moreover, there is no path such as 011000. Nevertheless this way of programming is not too professional: if we decide that a female may have up to two children, both can be males or females or they could be a couple, or just a male could be born or maybe nothing. The challenge is then to deal with all these possibilities. To solve this problem, we add to the associated string of a node a prefix "$" as a distinctive of maleness and declare that recombination is possible only among beings of different sex and that only females reproduce.*

87 ***Exercise.*** *Develop the code for a simulation of the* ***mutation-model*** *of evolution that is restricted to a binary tree, with mutation, sexual dimorphism and recombination.* *Answer*

88 ***Exercise.*** *Play enough with the solution to the previous exercise to agree else disagree with the following proposition: a population with sexual dimorphism and with only two sons per female is doomed to extinction even with recursion to parthenogenesis.* *Answer*

3.5 Correction of the scrolling bug

We have continuously dragged the problem that our graphics do not scroll and as a consequence we are unable to see a complete tree for a number of generations beyond 4. Let us correct that bug.

89 ***How scrolling works***

To have graphics that scroll, one must fulfill two main conditions. The first is to work with a scrolling pane, a possibility that is furnished by the JscrollPane class such as we have done. The second is that the possibility to have scrolling that is enabled by the aforementioned class must be actualized by a suitable procedure. The logic of design of the required procedure is very natural:

The space needed by the graphic must be calculated in the form of a rectangle or box. Since it begins in (0,0), one needs to give just its width and height. These

two quantities are encoded in a Dimension and is called, say, area. That area is fed into the used instance of the JScrollPane that it could decide whether or not the area fits the assigned size of the JScrollPane. If area is bigger, then one must command the scrolling bars to pop up to enable the User to scroll the graphic. The corresponding code looks like this:

```
 //area = size of the graphic (width, height)
 //area is set as the preferred size of the graphic
 drawingPane.setPreferredSize(area);

 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane.revalidate();
```

90 ***The code where we can learn the scrolling technology follows:***

```
//Program J90 ScrollDemo2
//A scrolling pane that correctly functions.

/*
 * http://docs.oracle.com/javase/tutorial/uiswing
/components/scrollpane.html
 *
 */

/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 *  with or without
 * modification, are permitted provided that
 * the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain
 *   the above copyright
```

```
 *     notice, this list of conditions and
 *     the following disclaimer.
 *
 *   - Redistributions in binary form must
 *   reproduce the above copyright
 *     notice, this list of conditions and
 *     the following disclaimer in the
 *     documentation and/or other materials
 *     provided with the distribution.
 *
 *   - Neither the name of Oracle or the names of its
 *     contributors may be used to endorse or
 *     promote products derived
 *     from this software without specific
 *     prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING
 * IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
```

```
/*
 * This code is based on an example provided by John Vella,
 * a tutorial reader.
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

/* ScrollDemo2.java requires no other files. */
public class ScrollDemo2 extends JPanel
                         implements MouseListener {
/**
 *
 */
private static final long serialVersionUID = 1L;
//indicates area taken up by graphics
    private Dimension area;
    //coordinates used to draw graphics
    private Vector<Rectangle> circles;
    private JPanel drawingPane;

    private final Color colors[] = {
        Color.red, Color.blue, Color.green, Color.orange,
        Color.cyan, Color.magenta,
        Color.darkGray, Color.yellow};
    private final int color_n = colors.length;

    public ScrollDemo2() {
        super(new BorderLayout());

        area = new Dimension(0,0);
        circles = new Vector<Rectangle>();

        //Set up the instructions.
        JLabel instructionsLeft = new JLabel(
            "Click left mouse button to place a circle.");
        JLabel instructionsRight = new JLabel(
         "Click right mouse button to clear drawing area.");
```

```
        JPanel instructionPanel =
         new JPanel(new GridLayout(0,1));
        instructionPanel.setFocusable(true);
        instructionPanel.add(instructionsLeft);
        instructionPanel.add(instructionsRight);

        //Set up the drawing area.
        drawingPane = new DrawingPane12();
        drawingPane.setBackground(Color.white);
        drawingPane.addMouseListener(this);

        //Put the drawing area in a scroll pane.
        JScrollPane scroller = new JScrollPane(drawingPane);
        scroller.setPreferredSize(new Dimension(200,200));

        //Lay out this demo.
        add(instructionPanel, BorderLayout.PAGE_START);
        add(scroller, BorderLayout.CENTER);
    }

    /** The component inside the scroll pane. */
    public class DrawingPane12 extends JPanel {
        /**
 *
 */
private static final long serialVersionUID = 1L;

protected void paintComponent(Graphics g) {
            super.paintComponent(g);

      Rectangle rect;
      for (int i = 0; i < circles.size(); i++) {
       rect = circles.elementAt(i);
       g.setColor(colors[(i % color_n)]);
       g.fillOval(rect.x, rect.y, rect.width, rect.height);
       }
        }
    }

    //Handle mouse events.
```

```
    public void mouseReleased(MouseEvent e) {
        final int W = 100;
        final int H = 100;
        boolean changed = false;
        if (SwingUtilities.isRightMouseButton(e)) {
            //This will clear the graphic objects.
            circles.removeAllElements();
            area.width=0;
            area.height=0;
            changed = true;
        } else {
            int x = e.getX() - W/2;
            int y = e.getY() - H/2;
            if (x < 0) x = 0;
            if (y < 0) y = 0;
            Rectangle rect = new Rectangle(x, y, W, H);
            circles.addElement(rect);
            drawingPane.scrollRectToVisible(rect);

            int this_width = (x + W + 2);
            if (this_width > area.width) {
                area.width = this_width; changed=true;
            }

            int this_height = (y + H + 2);
            if (this_height > area.height) {
                area.height = this_height; changed=true;
            }
        }
        if (changed) {
            //Update client's preferred size because
            //the area taken up by the graphics has
            //gotten larger or smaller (if cleared).
            drawingPane.setPreferredSize(area);

            //Let the scroll pane know to update itself
            //and its scrollbars.
            drawingPane.revalidate();
        }
        drawingPane.repaint();
```

```
    }
    public void mouseClicked(MouseEvent e){}
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    public void mousePressed(MouseEvent e){}

    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {
     //Create and set up the window.
     JFrame frame = new JFrame("ScrollDemo2");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

     //Create and set up the content pane.
     JComponent newContentPane = new ScrollDemo2();
   //content panes must be opaque
     newContentPane.setOpaque(true);
     frame.setContentPane(newContentPane);

     //Display the window.
     frame.pack();
     frame.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable()
         {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}//End of main class J90 ScrollDemo2
```

91 ***Exercise.*** *Run the previous program and play with the code.*

Let us apply this technology to our problems. We begin with an updating of the simplest model.

92 ***The refurnished code with scrolling capabilities for the basic-mutation model follows:***

```
//Program 92 BinaryEvo5
/*
* We run the mutation-model
* restricted over a binary tree.
* We follow our own encoding rules
* and we use our own drawing machinery.
*
* The code has been made more readable
* and more consistent.
* Scrolling works perfectly.
*
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class BinaryEvo5 extends JPanel
                        implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane13;
 JScrollPane scroller;


 //Number of reproductive cycles
 private static int numberOfGenerations = 3;
```

```
 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;


//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[1000000];
 //Last used position  in Record[]
 private static int lastUsedPosition;


 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
    "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();

 private static Random r = new Random();
 //Mutation rate.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
```

```
 private static int maxNumbChildren = 2;
```

```
public BinaryEvo5()
{
 super(new BorderLayout());

 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);

 JPanel instructionPanel = new JPanel(new GridLayout(0,1));
 instructionPanel.setFocusable(true);
 instructionPanel.add(clearButton);

 //Set up the drawing area.
 drawingPane13 = new drawingPane13();
 drawingPane13.setBackground(Color.white);


 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane13);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
 String command = e.getActionCommand();
```

```
 //If reset button is clicked over
 if  (RESET_COMMAND.equals(command))
  {
   evolution();
   repaint();
  }
}


//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
 String mutant = "";
 int lengthIndividual = individual.length();
 for(int i = 0; i < lengthIndividual; i++)
   {
    double  p = r.nextDouble();
    if (p<m)  //mutation
       { //a random integer number
        int l = r.nextInt(lengthAlf);
        //A random char
        char c = Alphabet.charAt(l);
        mutant = mutant + c;
     }
    else
     { //no mutation
      mutant = mutant + individual.charAt(i);
      }
    }
 return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;
```

```
public static void reproduce(int i)
{
 String direction; //0 = up; 1 = down
 int t = r.nextInt(2);
 if ( t == 0 ) direction = "0";
 else direction = "1";
 childPath = fatherPath + direction;
 lastUsedPosition++;
 Record[lastUsedPosition] = childPath;
 //System.out.println("Path first = " + childPath);
 lastUsedPosition++;
 Record[lastUsedPosition] = mutation(Record[i+1]);
 if (actualNumberOfChildren == 1)
  {
   //If a second child exists,
   //its direction is contrary to that of the first
   if ( t == 0 ) direction = "1";
   else direction = "0";
   childPath = fatherPath + direction;
   //System.out.println("Path second = " + childPath);
   lastUsedPosition++;
   Record[lastUsedPosition] = childPath;
   lastUsedPosition++;
   Record[lastUsedPosition] = mutation(Record[i+1]);
  }
 }


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
{
 //Reproductive leaves are the nodes at the end of Record[]
 //with the max length.
 String lastLeaf = Record[lastUsedPosition-1];
 int maxLength = lastLeaf.length();
 for(int i = lastUsedPosition-1; i >= 0; i = i-2)
  {
   //Nodes go at even entries
```

```
   fatherPath = Record[i];
   //System.out.println("Nodee = " + i  + "  " + fatherPath);
   //If node is a reproductive leaf
   if( fatherPath.length() == maxLength )
      {
    //The number of children is within
    //0 and 2;
        actualNumberOfChildren =
         r.nextInt(maxNumbChildren + 1);
        /*
        System.out.println("Number of children = "
          + actualNumberOfChildren) ; */
        //Leaf undergoes reproduction
        if (actualNumberOfChildren > 0)
        {
          reproduce(i);
        }
    //System.out.println(node.toString()) ;
       }

  }
 }


//An evolutionary process under the
//mutation-model is triggered.
//Every leaf (node of the last generation) is subjected
//to reproduction with a number of children
//equal or less than 2.
//Reproduction may undergo mutation in each site.
private static void evolution()
{
 //Initialization of node 0: path goes up
 Record[0] = "0";
 //The string of node 0
 Record[1] = mutation(seed);
 //Initialization of node 1: path goes down
 Record[2] = "1";
 //The string of node 1
 Record[3] =  mutation(seed);
```

```
 //Last used position in Record[]
 lastUsedPosition = 3;
 //Evolution is run for some reproductive cycles
 //Evolution = the mutation-model = just mutation.
 //Generation 0 was made by hand in the initilization
 for(int gen = 1; gen < numberOfGenerations; gen++)
  {
   //System.out.println("gen = " + gen);
   reproductionOfLeaves();
  }
}


/** The component inside the scroll pane. */
public class drawingPane13 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
```

```
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }
```

```
//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area = new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane13.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane13.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
```

```
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
    }
 }



//Graphical  master
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;

  Font font = new Font("Arial", Font.PLAIN, 12);
```

```
  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);

  g2d.setColor(Color.red);
   //Draw stem
  g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
  sizeOfTree(g2d);
  drawTree(g2d);
  drawStrings(g2d);
 }
}


/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("BinaryEvo5");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new BinaryEvo5();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
```

```
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
     createAndShowGUI();
     evolution();
     }
   });
 }
} //End of main class 92 BinaryEvo5
```

93 ***Exercise.*** *Run the previous program and play with the code.*

94 ***Exercise.*** *Include scrolling possibilities in the program J84 BinaryEvo3, which simulates evolution with mutation and recombination.* *Answer*

95 ***Exercise.*** *Include scrolling possibilities in the program J87 BinaryEvo4, which simulates evolution with mutation, recombination and sexual dimorphism.* *Answer*

96 ***Challenge.*** *We have calculated the area of the graphic in the previous programs for the extreme case, in which all possible nodes get realized. But this is not what ordinarily happens. As a consequence, the tree appears as surrounded by large vacuums. Make the necessary amendments to reduce so much padding to just what is needed.*

97 ***Hindsight.*** *We have programs that simulate evolution in three cases: with mutation alone, with mutation + recombination, and with mutation + recombination + sexual dimorphism. Merge all of them in a single class that presents to the User the possibility of choosing which option to run. Unification works are of great esteem and maybe taken as distinctive of both a great intelligence and a exquisite style. From outside, what one sees is that a program makes various tasks that are executed by a code that is rich in regulating instructions (if this and this happens do this else if ...). Now, it is very astonishing that even the most primitive bacteria are full in regulating instructions. Can natural evolution create things like that? Yes, it can. But a price must be paid: huge amounts of bugs. So, if somebody wants the Evolutionary Theory to become a part of science, he or she must accompany his or her claims with reports of the fossil record that point to so much trials that led to an ascending chain of successes.*

98 ***Warning.*** *Our previous programs that simulate evolution have been addressed to work out binary trees assuming that one has no troubles with memory. Nevertheless, one can verify that available memory turns to be insufficient for rather moderate values of the number of generations that are simulated. One might think that the problem will be solved by expanding memory or by recurring to hard disks. It is important to realize that all that will be useless: necessary memory grows as an exponential function while available memory is always a rather humble number. So, aforementioned solutions will cause insignificant improvements. An accepted solution to the memory problem is to restrict reproduction to one child and to record evolution over a moderate population that might be geographically distributed.*

3.6 Conclusion

Binary trees are those in which each node can have two children at maximum. They are simple and we have managed to keep our own Java implementations quite simple. In fact, we encoded binary trees in a vector, even entries for nodes and odd for names or contents. Each node is described by its address in the tree which is a binary string, say 1001, in which a zero means going to the left in the tree and a one is an instruction to going to the right. It has been straightforward to simulate evolution along a **mutation-model** that includes mutation, recombination and sexual dimorphism. Because of simplicity and transparency we are encouraged to use them in the sequel. We also had the opportunity to see, in hindsight, that situations may arise in which various tasks in software development must be solved and it would be desirable to unify them into a single class. Unification works are of great esteem and may be taken as distinctive of both a great intelligence and a exquisite style. From outside, what one sees is that a program makes various tasks that are executed by a code that is rich in regulating instructions (if this and this happens do this else if ...). Now, it is very astonishing that even the most primitive bacteria are full in regulating instructions. Can natural evolution create things like that? Yes, possibly it can. But a price must be paid: huge amounts of bugs. So, if somebody wants the Evolutionary Theory to become a part of science, he or she must accompany his or her claims with reports of the fossil record that point to so many trials that led to an ascending chain of successes. Reports must be also plenty of relevant malformations and malfunctions in living populations.

Chapter 4

The nearest neighbor's principle

```
Who is my father?
```

99 ***Introduction.*** *Trees can represent concrete evolutionary theories about the possible evolutionary relation among species. Such a tree is a* ***phylogentic tree****, while* ***phylogenetic inference*** *is the art of devising phylogenetic trees that could represent optimal proposals that eventually could be the most difficult to falsify. We will introduce and test in this chapter the power of the* ***nearest neighbor's principle*** *to devise phylogenetic trees: my father is that being of the just past generation to whom I am the most like. To measure differences between texts we will rely on the n-gram technology that we worked thoroughly in Vol IX. Our study will be restricted to binary trees.*

4.1 The Euclidean n-gram distance

Our objects of study are strings. They can be DNA, RNA protein sequences or sequences synthesized from letters taken from an arbitrary alphabet. To measure differences we use the n-gram technology. The underpinning idea is easy to grasp:

100 ***The idea***

We begin with an **alphabet**, an ordered set of symbols. A substring n-letters long is called an **n-gram**. A **text** is a string of undefined length, preferentially a long one. For a given text and for a given n, one can record the n-gram frequencies in a vector, following lexicographic order. Dividing by the number of n-grams in the analyzed text, one gets the vector of relative n-gram frequencies. Such a vector is an n-tuple of real numbers, that is, it an element of R^n. To measure the distance between two texts, we calculate their vectors of relative n-gram frequencies and

measure between them the Euclidean distance (a generalization of the form we measure distances over a plane). The resultant number is the **Euclidean n-gram distance** or **n-gram distance**.

The n-gram technology is currently used in Internet for language recognition. It is a direct, clean and powerful verification of the clonal theory about the operating mechanism of the immune system (see Vol IX).

101 ***Review.*** *We will include in programs below some methods that are necessary to calculate n-gram distances. They were taken mainly from Program* `I116 EuclideanDistMatrix2` *of Vol IX . The next code shows the whole dealt.*

```
//Program J101 nGramD

/*
 * This program receives as input two texts.
 Next, for each text it calculates the vector
 of absolute frequencies
 of all possible n-grams,
 then calculates the relative frequencies
 by dividing each absolute frequency
 by the number of n-grams in each studied text.
 When the two vectors of relative n-gram frequencies have
 been obtained, the Euclidean n-gram distance
 between those vectors is calculated and reported.

 */


public class nGramD
{

 //The alphabet need by fasta format.
 private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*- ";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n=2;
```

```
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
 private static double[]  freq1, freq2
                                = new double[N];

 //private static int L = (int) Math.pow(lengthAlf+1, n);



 private static Boolean printAll;


//Returns the   n-gram corresponding to index ind
//This is the inverse operation of index()
private static String indexInverse(int n, int ind)
{
char c1 = ' ';
int p = 0;
String ngram = "";
int k = 0;
int indl = ind;

//System.out.println("n = " + n);
for(int i = n-1; i > -1; i--)
{
//System.out.println("**** \nIndex = " + indl);
k = ( int ) (Math.pow(lengthAlf, i));
//System.out.println("k = " + k);
if( k <= indl)
{
p = indl / k;
//System.out.println("p = " + p);
if (p < Alphabet.length())
c1 = Alphabet.charAt(p);

indl = indl - p*k;
```

```
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

}
return ngram;
}


  //Returns the index   of ngram in F[] vectors.
  //An n-gram is a string with n chars taken from Alphabet.
  //Take it as a number in base = length of the Alphabet.
  //The index is that number in base 10.
  private static int index(int n, String ngram)
  {
   char c1 = ' ';
   int p = 0;
   int ind = 0;
   boolean correct = true;
   for(int i = 0; (i <n) & correct; i++)
    {
   c1 = ngram.charAt(i);
   p = Alphabet.indexOf(c1);
   if (p>=0)
    ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
   else
   {
    ind = -1;
    //Detecting the presence of an alien symbol
    correct = false;
   }
    }
   if (printAll)
   System.out.println( "ngram = " + ngram + " index = "
                                              + ind);
   return ind;
  }

  /*
```

```
This procedure records in F the
frequency of appearance of n-grams in the given Text.
Each n-gram is given an index, a number that
identifies it in the  vector F.
At the beginning, every entry of F  is set to 0.
The index of each n-gram of the Text is
calculated and the corresponding entry in F  is
incremented by one.
*/
private static double[] nGramsFrequency(int n,
       StringBuffer Text)
{
 //Zeroed initialization
 double[] F = new double[N];
 int nText = Text.length();
 System.out.println("Length  of working text = " + nText);
 int start = 0;
 int ind = 0;
 //Number of possible n-grams
 int nPossibleNGrams = nText-n+1;
 for(int c=0; c < nPossibleNGrams; c++)
  {
   String ngram =  Text.substring(start, start+n);
   ind = index(n,ngram);
   //if a given char of an n-gram is not in
   //the alphabet, its index is negative:
   //you must enlarge the alphabet else ignore it.
   //We ignore it.
   if (ind >= 0)   F[ind] = F[ind] + 1;
   start = start +1;
  }
 for(int i = 0; i < N; i++)
 F[i] = F[i]/nPossibleNGrams;
 double sum = 0;
 for(int i = 0; i < N; i++)
 sum = sum + F[i];
 //System.out.println("Frequencies add up to "+ sum);
 return F;
}
```

```
  //The Euclidean distance among two vectors of integer type
  //(declared as an object intVector3) is calculated
  private static double euclideanDistance(int N,
                double[] F1, double[] F2)
  {
   double sum = 0;
   double d = 0;
   String nGram = "";
   if (printAll)
  System.out.println("\nRelative frequency " +
  "vectors to compare");
   for(int i = 0; i < N; i++)
    {
    double a = F1[i]-F2[i];
  sum = sum + a*a;
  d = Math.sqrt(sum);
  if (printAll)
  {
   nGram = indexInverse(n, i);
   System.out.println( i + "  " +  nGram + "  " +
                     F1[i] + "  " + F2[i]);
  }
    }
   return d;
  }


 public static void main(String[] args)
  {
//To print additional information, set to true
printAll = true;
System.out.println( "\nLetters per n-gram = " + n );
System.out.println( "Lenght of the complete alphabet = " +
                                       lengthAlf);
System.out.println( "Number of n-grams = " + N);
StringBuffer s1 = new StringBuffer("THIS IS THE FIRST TEXT");
StringBuffer s2 = new StringBuffer("THIS IS THE SECOND TEXT");
```

```
System.out.println("Text 1 = " + s1.toString());
System.out.println("Text 2 = " + s2.toString());
System.out.println("\n" +  n + "-grams of  text 1:");
freq1 = nGramsFrequency(n, s1);
System.out.println("\n" +  n + "-grams of   text 2:");
freq2 = nGramsFrequency(n, s2);
Double dist = euclideanDistance(N, freq1,freq2);
System.out.println(n + "-distance = " + dist);

   }

}//End of main class J101 nGramD
```

4.2 The nearest neighbor's principle

How are we going to use the n-gram distance to propose a robust phylogeny? We will rely on a very simple answer to a very natural question:

102 ***Who is my father?***

To answer to this question we rely on the **nearest neighbor's principle**: my father is the man of the past generation that is closest to me in the relative n-gram frequency space endowed with the Euclidean distance.

Our first duty is to run a test: can we use the neatest neighbor's principle to successfully single out the ancestors of a given leaf in an evolutionary tree?

If we pass the test, we might proceed to the calculation of phylogenies.

4.3 Testing the principle

Science is built upon two main ingredients. First: the proposal of principles that must be mechanistic, nice and simple enough to awake in researchers the desire of testing it. Second: suitable tests that calculate predictions and compare them with actual data. Now, we have a principle but where are the tests?

103 ***A test under the mutation-model***

Let us test the nearest neighbor's principle endowed with the n-gram distance over an evolutionary process with just point mutation, i.e., our background is the **mutation-model**. Our immediate aim is to see whether or not we can use this principle to single out the direct ancestor of the latest generated leaf.

```
//Program 103 TestNearest

/*
* We test the power of the
* nearest neighbor's principle to single out
* the direct ancestor of a leaf over an evolutionary process
* that is restricted to mutation  over a binary tree.

* Nearest neighbor's principle:
* My father is that man whose distance to me
* is the least among the members of the last generation.
* We choose the n-gram distance to measure differences.
*
* An n-gram is a substring n chars long.
* For a given text we calculate the vector of relative n-gram
* frequencies. The n-gram distance between two texts is
* the Euclidean distance (a generalization of the form
* we measure the distance in the plane) between
* the two vectors of relative n-gram frequencies
* that are points of R^n.
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class TestNearest extends JPanel
                         implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane16;
 JScrollPane scroller;


 //Number of reproductive cycles
 private static int numberOfGenerations = 6;
```

```
 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;


//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[1000000];
 //Last used position  in Record[]
 private static int lastUsedPosition;


 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
  "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet =
 "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();

 private static Random r = new Random();

 //MUTATION RATE.
 private static double m = 0.1;
 //Max number of children per leaf.
```

```
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;



/*The alphabet needed by fasta format.
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
                   */

//letters per n-gram
private static  int n=2;
//Number of n-grams
private static  int N = (int) Math.pow(lengthAlf, n);


//The frequency of appearance of n-grams in the text.
//If n>3,  the assigned memory must be expanded
private static double[]  freq1, freq2
                                = new double[N];

//private static int L = (int) Math.pow(lengthAlf+1, n);



private static Boolean printMore = false;




public TestNearest()
{
 super(new BorderLayout());

 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);

 JPanel instructionPanel = new JPanel(new GridLayout(0,1));
```

```
 instructionPanel.setFocusable(true);
 instructionPanel.add(clearButton);

 //Set up the drawing area.
 drawingPane16 = new drawingPane16();
 drawingPane16.setBackground(Color.white);


 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane16);
 scroller.setPreferredSize(new Dimension(500,1000));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
 String command = e.getActionCommand();
 //If reset button is clicked over
 if  (RESET_COMMAND.equals(command))
  {
   evolution();
   repaint();
  }
}


//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
 String mutant = "";
 int lengthIndividual = individual.length();
 for(int i = 0; i < lengthIndividual; i++)
```

```
   {
    double  p = r.nextDouble();
    if (p<m)  //mutation
       { //a random integer number
        int l = r.nextInt(lengthAlf);
        //A random char
        char c = Alphabet.charAt(l);
        mutant = mutant + c;
     }
    else
     { //no mutation
      mutant = mutant + individual.charAt(i);
      }
    }
 return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;

public static void reproduce(int i)
{
 String direction; //0 = up; 1 = down
 int t = r.nextInt(2);
 if ( t == 0 ) direction = "0";
 else direction = "1";
 childPath = fatherPath + direction;
 lastUsedPosition++;
 Record[lastUsedPosition] = childPath;
 //System.out.println("Path first = " + childPath);
 lastUsedPosition++;
 Record[lastUsedPosition] = mutation(Record[i+1]);
 if (actualNumberOfChildren == 1)
  {
   //If a second child exists,
   //its direction is contrary to that of the first
   if ( t == 0 ) direction = "1";
```

```
   else direction = "0";
   childPath = fatherPath + direction;
   //System.out.println("Path second = " + childPath);
   lastUsedPosition++;
   Record[lastUsedPosition] = childPath;
   lastUsedPosition++;
   Record[lastUsedPosition] = mutation(Record[i+1]);
  }
 }


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
{
 //Reproductive leaves are the nodes at the end of Record[]
 //with the max length.
 String lastLeaf = Record[lastUsedPosition-1];
 int maxLength = lastLeaf.length();
 for(int i = lastUsedPosition-1; i >= 0; i = i-2)
  {
   //Nodes go at even entries
   fatherPath = Record[i];
   //System.out.println("Nodee = " + i  + "  " + fatherPath);
   //If node is a reproductive leaf
   if( fatherPath.length() == maxLength )
      {
    //The number of children is within
    //0 and 2;
        actualNumberOfChildren =
         r.nextInt(maxNumbChildren + 1);
        /*
        System.out.println("Number of children = "
          + actualNumberOfChildren) ; */
        //Leaf undergoes reproduction
        if (actualNumberOfChildren > 0)
        {
          reproduce(i);
        }
    //System.out.println(node.toString()) ;
```

```
       }

   }
 }


//An evolutionary process under the
//mutation-model is triggered.
//Every leaf (node of the last generation) is subjected
//to reproduction with a number of children
//equal or less than 2.
//Reproduction may undergo mutation in each site.
private static void evolution()
{
 //Initialization of node 0: path goes up
 Record[0] = "0";
 //The string of node 0
 Record[1] = mutation(seed);
 //Initialization of node 1: path goes down
 Record[2] = "1";
 //The string of node 1
 Record[3] =  mutation(seed);
 //Last used position in Record[]
 lastUsedPosition = 3;
 //Evolution is run for some reproductive cycles
 //Evolution = the mutation-model = just mutation.
 //Generation 0 was made by hand in the initilization
 for(int gen = 1; gen < numberOfGenerations; gen++)
  {
   //System.out.println("gen = " + gen);
   reproductionOfLeaves();
  }
}

//=========================================
//============ n-Gram methods ============
//=========================================


//Returns the   n-gram corresponding to index ind
```

```
//This is the inverse operation of index()
private static String indexInverse(int n, int ind)
{
char c1 = ' ';
int p = 0;
String ngram = "";
int k = 0;
int indl = ind;

//System.out.println("n = " + n);
for(int i = n-1; i > -1; i--)
{
//System.out.println("**** \nIndex = " + indl);
k = ( int ) (Math.pow(lengthAlf, i));
//System.out.println("k = " + k);
if( k <= indl)
{
p = indl / k;
//System.out.println("p = " + p);
if (p < Alphabet.length())
c1 = Alphabet.charAt(p);

indl = indl - p*k;
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

}
return ngram;
}


 //Returns the index   of ngram in F[] vectors.
 //An n-gram is a string with n chars taken from Alphabet.
 //Take it as a number in base = length of the Alphabet.
 //The index is that number in base 10.
 private static int index(int n, String ngram)
 {
  char c1 = ' ';
```

```
 int p = 0;
 int ind = 0;
 boolean correct = true;
 for(int i = 0; (i <n) & correct; i++)
  {
c1 = ngram.charAt(i);
p = Alphabet.indexOf(c1);
if (p>=0)
 ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
else
{
 ind = -1;
 //Detecting the presence of an alien symbol
 correct = false;
}
  }
 if (printMore)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

/*
This procedure records in F the
frequency of appearance of n-grams in the given Text.
Each n-gram is given an index, a number that
identifies it in the  vector F.
At the beginning, every entry of F  is set to 0.
The index of each n-gram of the Text is
calculated and the corresponding entry in F  is
incremented by one.
*/
private static double[] nGramsFrequency(int n,
       StringBuffer Text)
{
 //Zeroed initialization
 double[] F = new double[N];
 int nText = Text.length();
 //System.out.println("Length  of working text = " + nText);
 int start = 0;
```

```
  int ind = 0;
  //Number of possible n-grams
  int nPossibleNGrams = nText-n+1;
  for(int c=0; c < nPossibleNGrams; c++)
   {
    String ngram =  Text.substring(start, start+n);
    //System.out.println("String ngram = " + ngram);
    ind = index(n,ngram);
    //if a given char of an n-gram is not in
    //the alphabet, its index is negative:
    //you must enlarge the alphabet else ignore it.
    //We ignore it.
    if (ind >= 0)   F[ind] = F[ind] + 1;
    start = start +1;
   }
  for(int i = 0; i < N; i++)
  F[i] = F[i]/nPossibleNGrams;
  double sum = 0;
  for(int i = 0; i < N; i++)
  sum = sum + F[i];
  //System.out.println("Frequencies add up to "+ sum);
  return F;
 }



 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
               double[] F1, double[] F2)
 {
  double sum = 0;
  double d = 0;
  String nGram = "";
  if (printMore)
System.out.println("\nRelative frequency " +
"vectors to compare");
  for(int i = 0; i < N; i++)
   {
```

```
  double a = F1[i]-F2[i];
sum = sum + a*a;
d = Math.sqrt(sum);
if (printMore)
{
 nGram = indexInverse(n, i);
 System.out.println( i + "  " +  nGram + "  " +
                  F1[i] + "  " + F2[i]);
}
   }
  return d;
 }



//=========================================
//========== Ancestral tracer ==========
//=========================================



private static int memory = N;
//Number of nodes of the last generation
private static int numberOfFatherNodes = 0;
//Position of nodes of the previous generation in Record[]
private static  int[] PreviousGen = new int[memory];

//Nodes of the previous generation is found
//and the difference of each one of them
//to the  node under test is measured.
public static void
           findPreviousGeneration(int position)
{
int counter = 0;
//Number of bits in the node under test
int lengthThis = Record[position].length();
//The previous generation consists in those  nodes
//whose length = lengthThis-1
```

```
for(int i = position; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  String previous = Record[i];
  //If node belongs in the previous generation
  if( previous.length() == lengthThis - 1)
 {
  PreviousGen[counter] = i;
  counter++;
  //System.out.println("Previous = " + Record[i]);
 }
}
numberOfFatherNodes = counter;
}


//The n-gram distance of the node under test to
//nodes of the previous generation is measured.
//That node with minimum distance is single out.
public static int   findDirectAncestor(int position)
{
 double minDistance = 1000000000;
 int ancestorPosition = 0;
 String address = "";
 StringBuffer s1 =
 new StringBuffer(Record[position+1]);
 //System.out.println("String under test = " + s1.toString());
 freq1 = nGramsFrequency(n, s1);

 for(int i = 0; i < numberOfFatherNodes; i++)
 {
  int j = PreviousGen[i];
  StringBuffer s2 =
 new StringBuffer(Record[j+1]);
  //System.out.println("\n" +  n + "-grams of   text 2:");
  freq2 = nGramsFrequency(n, s2);
  Double dist = euclideanDistance(N, freq1,freq2);
  //System.out.println(n + "-distance = " + dist);
  if (dist < minDistance)
   {
```

```
  minDistance = dist;
  ancestorPosition = j;
   }
  }
 System.out.println("Direct ancestor       = "
               + Record[ancestorPosition] );
 /*
 System.out.println("Direct ancestor = "
                    + Record[ancestorPosition+1]);*/
return ancestorPosition;
}

//A node is addressed by position.
//Its ancestors are found by means of the n-gram technology.
//One can verify the answer directly:
//the ancestor of 11101 is 1110 and
//that of 1010100 is 101010. And so on.
private static void findAncerstors(int position)
{
 findPreviousGeneration(position);
 findDirectAncestor(position);
}



//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
public class drawingPane16 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
```

```
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
```

```
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }


//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));
```

```
 //area = size of the graphic (width, height)
 Dimension area = new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane16.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane16.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
```

```
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
    }
 }

//Graphical  master
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;

  Font font = new Font("Arial", Font.PLAIN, 12);

  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);

  g2d.setColor(Color.red);
   //Draw stem
  g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
  sizeOfTree(g2d);
  drawTree(g2d);
  drawStrings(g2d);
  System.out.println("\nThe direct ancestor of  " +
                     Record[lastUsedPosition-1] + " is: ");
  findAncerstors(lastUsedPosition-1);
 }
}



/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
```

```
{
 //Create and set up the window.
 JFrame frame = new JFrame("TestNearest");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new TestNearest();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
     createAndShowGUI();
     evolution();
     System.out.println( "\nLetters per n-gram = " + n );
     System.out.println( "Lenght of the complete" +
      " alphabet = " +   lengthAlf);
     System.out.println( "Number of n-grams = " + N);
     }
   });
 }
}//End of main class 103 TestNearest
```

104 ***Exercise.*** *Run the previous program and play with the code. In particular, use the reset button of the GUI of the previous program to verify else reject the conclusion of the Author: the nearest neighbor's principle endowed with the n-gram distance perfectly singles out the direct ancestor of a given leaf.*

105 ***Exercise.*** *Go to the code of the previous program, look for the parameter that controls the mutation rate and give to it a high value, say, 0.8. Rerun the program. What do you observe? How do you explain that result?* Answer

106 ***Challenge.*** *Make a research to test the following proposal: under the mutation-model, when all mutations are viable, a small mutation rate over many generations is equivalent to a high mutation rate in one generation. Therefore and for any value of the mutation rate, the nearest neighbor's principle endowed with the n-gram distance would be good to single out the ancestors of a given individual but only up to certain number of generations. Then, the probability of failing augments as generations run forward. Next, discuss two possible applications of this result:*

First: evolution that relies on mutation alone rules in the longterm the existence of living fossils.

Second: the ***mutation-model*** *predicts that in the long term the descendants of dogs will be similar to cats, so convergent evolution becomes an evolutionary necessity. Since the first prediction is false, it is natural to adhere to the* ***function paradigm****.*

107 ***Exercise.*** *Generalize the code of the previous program J103 TestNearest to singled out in line all the ancestors of a given leaf.* Answer

108 ***Exercise.*** *Add interactive capabilities to the previous program. In response to the clicking of the User over a given leaf, all ancestors must get revealed using a distinct color for the strings and/or for the strokes of the tree.* Answer

109 ***Challenge.*** *Use the previous program as a tool to teach the use of the nearest neighbor's principle in phylogenetic inference.*

110 ***Challenge.*** *Use the previous program to study the role of a high mutation rate as a hindrance against phylogenetic inference.*

111 ***Challenge.*** *Reuse the previous program to study the role of the join effect of mutation rate, and reproduction and population size (more precisely, the size of the population that reproduce) over many generations as a hindrance against phylogenetic inference.*

112 ***Challenge.*** *Test the nearest nighbor's principle for trees in general. You must use the Java tree class.*

113 ***Testing under the functional paradigm.***

We have been working under the mutation-model and with binary trees. We claim that the nearest neighbor's principle can correctly single out in line all the ancestors of a given node when both the mutation rate and the number of generation are moderate. But our aim is to apply this principle to the genomes of living species. Is that allowed?

The problem is that no one would see at living species from the perspective of the mutation-model. Rather, everyone is forced by biology to look at life from the perspective of the function paradigm, which claims that the genome is software. More explicitly, the genome contains verbal instructions that need a code to be deciphered and whose execution enables life to be continued and to reproduce. See Vol II.

So, a question naturally arises: will the nearest neighbor's principle stand the test of evolution under the functional paradigm? Things will be more interesting if we get a lab to discuss this question.

114 ***Language as a model.***

An extremely dare approach to the function paradigm would be to research AL, artificial life, which for us means the simulation in Java of complete folds of life to study precisely how it unfolds under given construction and circumstances. Another also extreme approach is genetic programming which is the use of evolution to design software. We have not touched on AL but we made in Vol II some experiments on genetic programming and concluded that it is not a problem to be solved in a day, neither in programming time nor in running time.

In general what we have done is more related to models, to parables, to abstractions. Our present problem is another occasion to show that models are good because they allow us to make a preliminary analysis of ideas under study.

To test the power of the n-gram distance + the nearest neighbor's principle under the function paradigm, we propose to use languages as a natural testbed. The reason is that a human language is a programming language that conveys software for humans. Every text written by a human being transmits some information, a message, a meaning, instructions to do something. So, it is a very natural model of the genome. We can simulate an evolutionary process under the function paradigm if one poses as seed a phrase with meaning. Next, for the nodes of the next generation one adds variations by meaning. And so on. The following code shows one

such evolutionary tree. It is evolutionary because it was actually an evolutionary process that was guided by semantic association in the head of the Author.

115 ***The code for retrodicting the ancestors of nodes under the functional paradigm modeled on the English language.*** *The next code shows a semantic association evolutionary tree of texts in English and runs the nearest neighbor's principle to test how well it can unveil ancestors.*

```
//Program 115 TestNearest4

/*
* We test the power of the
* nearest neighbor's principle to single out
* the   ancestors of a leaf over
* an evolutionary process of English texts
* that has been invented by the Author.
*
* Nearest neighbor's principle:
* My father is that man whose distance to me
* is the least among the members of the last generation.
* We choose the n-gram distance to measure differences.
*
* An n-gram is a substring n chars long.
* For a given text we calculate the vector of relative
* n-gram frequencies. The n-gram distance between two
* texts is the Euclidean distance (a generalization
* of the form  we measure the distance in the plane)
* between the two vectors of relative n-gram frequencies
* that are points of Rˆn.
*/


import java.awt.*;
import java.awt.event.*;


import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
```

```
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;


public class TestNearest4 extends JPanel
                          implements
                          MouseListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane19;
 JScrollPane scroller;

 private static   String   seed =
 "if x= a, write something";

//This is data for an evolutionary process of
//English texts that has been invented by the Author.
 private static String[] Input =
 {
  "0", "if x= 2, write two",
  "1", "if s = e, write equal",
  "00", "if x= 3, write three",
  "01", "if x= 8, write eight",
  "10", "if s = evol, write evolution",
  "11", "if s = different, write n-gram distance",
  "000", "if x= 13, write three",
  "001", "if x= 3, write thirteen",
  "010", "if x= 81, write one",
  "011", "if x= 88, write double eight",
  "100", "if s = evol, write evolutionism",
  "101", "if s = evolution, write darwin",
  "110", "if s = gram, write kilogram",
  "111", "if s = gram, write grammar",
  "0000", "if x= 133, write thirty three",
  "0001", "if x= 13, write bad luck",
  "0010", "if x= 33, write double three",
  "0011", "if x= 3, write lonesome three",
  "0100", "if x= 8181, write twice 81",
  "0101", "if x= 18, write  eighteen",
```

```
  "0110", "if x= 888, write triple eight",
  "0111", "if x = 88, write double eight",
  "1000", "if s = evolution, write  creation",
  "1001", "if s = evol, write evolutionist",
  "1010", "if s = evolution, write wallace",
  "1011", "if s = evolution, write lamarck",
  "1100", "if s = grammar, write Chomsky",
  "1101", "if s = gram, write kilo",
  "1110", "if s = gram, write spanish grammar",
  "1111", "if s = gram, write german grammar",
 };


 //Number of reproductive cycles
 private static int numberOfGenerations = 4;

 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;



//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.

 private static int numberOfLeaves =
 (int) Math.pow(2, numberOfGenerations);

 private static String[] Record =  new String[4*numberOfLeaves];
 //Last used position  in Record[]
```

```
 private static int lastUsedPosition;



 // Definition of the alphabet
 private static String Alphabet =
 "abcdefghijklmnopqrstuvwxyz10";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();




/*The alphabet needed by fasta format.
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
                   */

//letters per n-gram
private static  int n=3;
//Number of n-grams
private static  int N = (int) Math.pow(lengthAlf, n);


//The frequency of appearance of n-grams in the text.
//If n>3,  the assigned memory must be expanded
private static double[]  freq1, freq2
                               = new double[N];


//Y-coordinates of points where strings are drawn
private static int[] yCoordinate = new int[4*numberOfLeaves];
//Position in Record[] of ancestral nodes
private static int[] Ancestors = new int[numberOfGenerations];
//The color to paint
private static Color color = Color.red;

//Reset = true for evolution anew.
```

```
//Reset = false when ancestors are to be found.
private static Boolean reset = false;
private static int numberOfAncestors;
private static Boolean printMore1 = false;
private static Boolean printMore2 = true;
//Position in Record[] of clicked node.
private static int place;


public TestNearest4()
{
 super(new BorderLayout());


 JPanel instructionPanel = new JPanel(new GridLayout(0,1));

 //Set up the instructions.
 JLabel instructionsLeft = new JLabel(
     "Click left mouse button to reveal ancestors.");
 JLabel instructionsRight = new JLabel(
  "Click right mouse button to reset. ");
 JLabel advertisment = new JLabel(
   "Nearest neighbord's principle + n-gram distance.");

 instructionPanel.setFocusable(true);
 instructionPanel.add(instructionsLeft);
 instructionPanel.add(instructionsRight);
 instructionPanel.add(advertisment);
 instructionsRight.setEnabled(false);

 //Set up the drawing area.
 drawingPane19 = new drawingPane19();
 drawingPane19.setBackground(Color.white);
 drawingPane19.addMouseListener(this);

 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane19);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
```

```
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Find the closest leaf to the spot of clicking.
private static int findClosestLeaf(int y)
 {
  int distance = 1000000;
  int position = 0;
  for(int i = 0; i < numberOfLeaves; i++)
   {
//pixels from clicked spot to a y-coordinate of a string
int pixels = Math.abs(y-yCoordinate[i]);
//The closest leaf is found
if ( pixels < distance )
  {
   distance = pixels;
   position  = i;
  }
   }
  return position;
 }



public void mouseClicked(MouseEvent e) { }



public void mousePressed(MouseEvent e) { }



public void mouseReleased(MouseEvent e)
{
 //Reset?
 if (SwingUtilities.isRightMouseButton(e))
  {
   //????
```

```
  }
  //Unveil ancestors?
 if (SwingUtilities.isLeftMouseButton(e))
  {
//Coordinates of clicked spot
int x = e.getX();
int y = e.getY();
//System.out.println( x + " " + y);
//Closest node is found
int yPlace = findClosestLeaf(y);
//System.out.println("================ " + yPlace);
//System.out.println("Clicked Position = " + yPlace);
//System.out.println("Path = " + Record[2*yPlace]);
reset = false;
//place = position in Record[] of the clicked node
//Place is a global variable used by
//the drawing machinery
    place = 2*yPlace;
    repaint();
  }

}



public void mouseEntered(MouseEvent e) { }



public void mouseExited(MouseEvent e) { }

//======Evolution====




//Input is read
private static void readInput()
{
  int l = Input.length;
```

```
  for (int i = 0; i < l; i++)
  Record[i] = Input[i];
  lastUsedPosition = l-1;
}



//=========================================
//============= n-Gram methods ============
//=========================================


//Returns the   n-gram corresponding to index ind
//This is the inverse operation of index()
private static String indexInverse(int n, int ind)
{
char c1 = ' ';
int p = 0;
String ngram = "";
int k = 0;
int indl = ind;

//System.out.println("n = " + n);
for(int i = n-1; i > -1; i--)
{
//System.out.println("**** \nIndex = " + indl);
k = ( int ) (Math.pow(lengthAlf, i));
//System.out.println("k = " + k);
if( k <= indl)
{
p = indl / k;
//System.out.println("p = " + p);
if (p < Alphabet.length())
c1 = Alphabet.charAt(p);

indl = indl - p*k;
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;
```

```
}
return ngram;
}


 //Returns the index   of ngram in F[] vectors.
 //An n-gram is a string with n chars taken from Alphabet.
 //Take it as a number in base = length of the Alphabet.
 //The index is that number in base 10.
 private static int index(int n, String ngram)
 {
  char c1 = ' ';
  int p = 0;
  int ind = 0;
  boolean correct = true;
  for(int i = 0; (i <n) & correct; i++)
   {
 c1 = ngram.charAt(i);
 p = Alphabet.indexOf(c1);
 if (p>=0)
  ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
 else
 {
  ind = -1;
  //Detecting the presence of an alien symbol
  correct = false;
 }
   }
  if (printMore1)
  System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
  return ind;
 }

 /*
 This procedure records in F the
 frequency of appearance of n-grams in the given Text.
 Each n-gram is given an index, a number that
 identifies it in the  vector F.
```

```
At the beginning, every entry of F  is set to 0.
The index of each n-gram of the Text is
calculated and the corresponding entry in F  is
incremented by one.
*/
private static double[] nGramsFrequency(int n,
       StringBuffer Text)
{
 //Zeroed initialization
 double[] F = new double[N];
 int nText = Text.length();
 //System.out.println("Length of working text = " + nText);
 int start = 0;
 int ind = 0;
 //Number of possible n-grams
 int nPossibleNGrams = nText-n+1;
 for(int c=0; c < nPossibleNGrams; c++)
  {
   String ngram =  Text.substring(start, start+n);
   //System.out.println("String ngram = " + ngram);
   ind = index(n,ngram);
   //if a given char of an n-gram is not in
   //the alphabet, its index is negative:
   //you must enlarge the alphabet else ignore it.
   //We ignore it.
   if (ind >= 0)   F[ind] = F[ind] + 1;
   start = start +1;
  }
 for(int i = 0; i < N; i++)
 F[i] = F[i]/nPossibleNGrams;
 double sum = 0;
 for(int i = 0; i < N; i++)
 sum = sum + F[i];
 //System.out.println("Frequencies add up to "+ sum);
 return F;
}
```

```
 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
               double[] F1, double[] F2)
 {
  double sum = 0;
  double d = 0;
  String nGram = "";
  if (printMore1)
System.out.println("\nRelative frequency " +
"vectors to compare");
  for(int i = 0; i < N; i++)
   {
double a = F1[i]-F2[i];
sum = sum + a*a;

if (printMore1)
{
nGram = indexInverse(n, i);
 System.out.println( i + "  " +  nGram + "  " +
                   F1[i] + "  " + F2[i]);
 }
}

  //Innocuous bug corrected
  d = Math.sqrt(sum);


  return d;
 }



//=========================================
//=========== Ancestral tracer ===========
//=========================================



private static int memory = N;
```

```
//Number of nodes of the last generation
private static int numberOfFatherNodes = 0;
//Position of nodes of the previous generation in Record[]
private static  int[] PreviousGen = new int[memory];

//Nodes of the previous generation is found
//and the difference of each one of them
//to the  node under test is measured.
public static void
           findPreviousGeneration(int position)
{
int counter = 0;
//Number of bits in the node under test
int lengthThis = Record[position].length();
//The previous generation consists in those  nodes
//whose length = lengthThis-1

//System.out.println("Previous generation" );
for(int i = position; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  String previous = Record[i];
  //If node belongs in the previous generation
  if( previous.length() == lengthThis - 1)
 {
  PreviousGen[counter] = i;
  counter++;
  //System.out.println("Previous = " + Record[i]);
 }
}
numberOfFatherNodes = counter;
}


//The n-gram distance of the node under test to
//nodes of the previous generation is measured.
//That node with minimum distance is single out.
public static int   findDirectAncestor(int position)
{
 double minDistance = 1000000000;
```

```
 int ancestorPosition = 0;
 StringBuffer s1 =
 new StringBuffer(Record[position+1]);
 /*
 System.out.println("String under   test = "
 + s1.toString());
 */
 freq1 = nGramsFrequency(n, s1);

 for(int i = 0; i < numberOfFatherNodes; i++)
 {

  int j = PreviousGen[i];
  /*
  System.out.println("\n Member of previous generation:"
  + PreviousGen[i] + Record[j+1]);
  */
  StringBuffer s2 =
 new StringBuffer(Record[j+1]);
  //System.out.println("\n  text 2:" + s2);
  //System.out.println("\n" +  n + "-grams of   text 2:");
  freq2 = nGramsFrequency(n, s2);
  Double dist = euclideanDistance(N, freq1,freq2);
  //System.out.println(n + "-distance = " + dist);
  if (dist < minDistance)
   {
  minDistance = dist;
  ancestorPosition = j;
   }
  }
return ancestorPosition;
}

 //print ancestors
private static void  printAncestors(int position)
{
 //Number of bits in the node under test
 //= number of generations
 int lengthThis = Record[position].length();
 if (printMore2)
```

```
  for (int i = 0; i < lengthThis; i++)
   {
 System.out.println("Direct ancestor       = "
            + Record[Ancestors[i]] );
   }
}


//A node is addressed by position.
//Its ancestors are found by means of the n-gram technology.
//One can verify the answer directly:
//the ancestor of 11101 is 1110 and
//that of 1010100 is 101010. And so on.
private static void findAncerstors(int position)
{
 //Number of bits in the node under test
 //= number of generations
 int lengthThis = Record[position].length();
 numberOfAncestors = lengthThis;
 int ancestorPosition = position;
 Ancestors[0] = ancestorPosition;
 for (int i = 0; i < lengthThis-1; i++)
  {
findPreviousGeneration(ancestorPosition);
ancestorPosition = findDirectAncestor(ancestorPosition);
Ancestors[i+1] = ancestorPosition;
  }
 printAncestors(position);
}



//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
public class drawingPane19 extends JPanel
{
```

```
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
```

```
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }


//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
```

```
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area =
 new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane19.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane19.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 color = Color.red;
 int counter = 0;
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 yCoordinate[counter] = verticalSize - y00;
 //counter++;
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
```

```
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                     verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
    }
  numberOfLeaves = counter;
 }

//Those strings that correspond with the ancestors
//are redrawn in another color.
private void drawAncestors(Graphics2D g2d, Color color)
{
g2d.setColor(color);
//Seed is drawn
g2d.drawString(seed,x00,verticalSize - y00);
//Strings are drawn:
for(int i = 0; i < numberOfAncestors; i++)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[Ancestors[i]];
 String s = Record[Ancestors[i]+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
```

```
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                     verticalSize - target.height-1);

  }
}

//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
 g2d.setColor(Color.red);
 //Draw stem
 g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
 sizeOfTree(g2d);
 drawTree(g2d);
 drawStrings(g2d);


 if (reset)//Evolution anew
   {
 if (printMore2)
     System.out.println("\nThe direct ancestor of  " +
       Record[lastUsedPosition-1] + " is: ");
 findAncerstors(lastUsedPosition-1);
 color = Color.blue;
 drawAncestors(g2d,color);
}
```

```
  else //draw ancestors
   {
if (printMore2)
  System.out.println("\nThe direct ancestor of  " +
                     Record[place] + " is: ");
findAncerstors(place);
color = Color.blue;
drawAncestors(g2d,color);
   }
}

//Graphical  master : it is called automatically
//under starting and by repaint()
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  //Font to draw strings
  Font font = new Font("Arial", Font.PLAIN, 12);
  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
  drawBasic(g2d);
 }


}//End of inner class drawingPane19



/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("TestNearest");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new TestNearest4();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
 readInput();
 reset = true;
     createAndShowGUI();
     System.out.println( "\nLetters per n-gram = " + n );
     System.out.println( "Lenght of the complete" +
                       " alphabet = " + lengthAlf);
     System.out.println( "Number of n-grams = " + N);
     }
   });
 }

}//End of main class 115 TestNearest4
```

116 ***Exercise.*** *Run the previous program and play with the code. Notice that there are instances of leafs whose ancestors can correctly be retrodicted by the program and that there are instances of leafs whose ancestors are incorrectly found by the previous program.*

117 ***Important question.*** *We have discovered that our algorithm is inadequate for finding the ancestors of a node in an evolutionary process that is guided by the*

function paradigm. One reason for the failure is that evolution under the function paradigm might include changes that represent sufficiently large n-gram distances that breaks the idea that my father is that man that is closest to me in n-gram space. Nevertheless, our technology has shown to work in some cases. So, can we imagine a world in which our algorithm is expected to work perfectly?

118 ***Smart answer: genetics***

A gen is a portion of DNA (RNA) that encodes for the primary structure of an enzyme or protein. Genes are named after the function of the proteins they encode for. One approach to genetics is **Cladistics (Understanding Evolution, [12], 2013)** . It is one of the most successful ways as one can suggest phylogenetic relations: it classifies organisms according to the presence or absence of characters, say to have wings or not, and with their modifications. In our case. to have a gen or not. So, one can naturally form a tree beginning with beings with low complexity to include more and more genetic characters.

The natural aspiration is that a well documented cladistic tree, with sufficient important characters, amounts to a robust evolutionary theory in regard with ancestors relationships and splitting patterns. Cladistics is a very complex art (Lipscomb, [4], 1998).

We adhere to molecular cladistics that looks at DNA directly to declare the presence or absence of a gen. In fact, every time we speak of a gen, we are saying that we are guided by the function paradigm because a gen is a DNA text whose function is to be an instruction for the synthesis of a poly-peptide that could become an enzyme or a structural molecule or a receptor in the cellular wall. So, the proteome is a very good place for a cladistic project.

Our intuition tells us that our n-gram technology is perfect for detecting ancestors in a cladistic world. The challenge is therefore to test such an idea. To do that at once, we model the function paradigm over the English language and encode a fancy evolutionary tree to see whether or not our technology allows us to retrodict ancestors.

119 ***Example.*** *The following is an example of a cladistic tree with super-gens modeled over the English language. Our purpose is to see whether or not our technology correctly retrodicts the ancestors of a node. We try out various encodings to see which is the best. The code follows:*

```
//Program 119 TestNearest5

/*
```

```
* We test the power of the
* nearest neighbor's principle + n-gram technology
* to single out    the   ancestors of a leaf over
* a fictitious evolutionary process of English texts
* that was invented to illustrate the stand of  cladistics.
*
* Cladistics classify organisms according to the presence
* or absence of characters, say wings or lungs,
* and of modifications to them. So, cladistics
* by their very nature belong in the function paradigm.
* We  consider here presence else
* absence of a given set of characters, a situation
* that can be encoded, say,  by binary numbers.
* Nevertheless, we try various encodings to see
* whether or not someone does it, if any:
*
* Our bit notation is like this: 1001111. It says that
* we study 7 characters from 0 to 6 and that the
* given organism lacks characters 1 and 2.
* The spoken bit notation is the very same bit notation
* but numbers replaced by the denoted characters:
* protolife+nolegs+lungs+noplacenta+end.
* The natural notation mentions a character if and only if
* it is present: protolife+lungs+end.
* If in that notation appears noCharacter1,
* it means that we have a regulating signal that turns off
* the  information for Character1.
*
* Nearest neighbor's principle:
* My father is that man whose distance to me
* is the least among the members of the last generation.
* We choose the n-gram distance to measure differences.
*
* An n-gram is a substring n chars long.
* For a given text we calculate the vector of relative
* n-gram frequencies. The n-gram distance between two
* texts is the Euclidean distance (a generalization
* of the form  we measure the distance in the plane)
* between the two vectors of relative n-gram frequencies
* that are points of R^n.
```

```
*/


import java.awt.*;
import java.awt.event.*;


import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;


public class TestNearest5 extends JPanel
                        implements
                        MouseListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane20;
 JScrollPane scroller;


 // Re-definition of the alphabet (includes 0 and 1)
 private static String Alphabet =
 "abcdefghijklmnopqrstuvwxyz10";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();



 private static   String   seed =
 "protolife";


 private static String[] Input =
 {
  "0", "protolife+0+end",
  "1", "protolife+1+end",
```

```
 "00", "protolife+00+end",
 "01", "protolife+01+end",
 "10", "protolife+10+end",
 "11", "protolife+11+end",
 "000", "protolife+000+end",
 "001", "protolife+001+end",
 "010", "protolife+010+end",
 "011", "protolife+011+end",
 "100", "protolife+100+end",
 "101", "protolife+101+end",
 "110", "protolife+110+end",
 "111", "protolife+111+end",
 "0000", "protolife+0000+end",
 "0001", "protolife+0001+end",
 "0010", "protolife+0010+end",
 "0011", "protolife+0011+end",
 "0100", "protolife+0100+end",
 "0101", "protolife+0101+end",
 "0110", "protolife+0110+end",
 "0111", "protolife+0111+end",
 "1000", "protolife+1000+end",
 "1001", "protolife+1001+end",
 "1010", "protolife+1010+end",
 "1011", "protolife+1011+end",
 "1100", "protolife+1100+end",
 "1101", "protolife+1101+end",
 "1110", "protolife+1110+end",
 "1111", "protolife+1111+end",
};



//Invented cladistic evolutionary process=
//protolife ->> + legs >> + lungs >> + placenta  >>  + hair
private static String[] Input2 =
{
 "0", "protolife+nolegs+end",
 "1", "protolife+legs+end",
 "00", "protolife+nolegs+nolungs+end",
 "01", "protolife+nolegs+lungs+end",
```

```
 "10", "protolife+legs+noplacenta+end",
 "11", "protolife+legs+lungs+end",
 "000", "protolife+nolegs+nolungs+noplacenta+end",
 "001", "protolife+nolegs+nolungs+placenta+end",
 "010", "protolife+nolegs+lungs+noplacenta+end",
 "011", "protolife+nolegs+lungs+placenta+end",
 "100", "protolife+legs+nolungs+noplacenta+end",
 "101", "protolife+legs+nolungs+placenta+end",
 "110", "protolife+legs+lungs+noplacenta+end",
 "111", "protolife+legs+lungs+placenta+end",
 "0000", "protolife+nolegs+nolungs+noplacenta+nohair+end",
 "0001", "protolife+nolegs+nolungs+noplacenta+hair+end",
 "0010", "protolife+nolegs+nolungs+placenta+nohair+end",
 "0011", "protolife+nolegs+nolungs+placenta+hair+end",
 "0100", "protolife+nolegs+lungs+noplacenta+nohair+end",
 "0101", "protolife+nolegs+lungs+noplacenta+hair+end",
 "0110", "protolife+nolegs+lungs+placenta+nohair+end",
 "0111", "protolife+nolegs+lungs+placenta+hair+end",
 "1000", "protolife+legs+nolungs+noplacenta+nohair+end",
 "1001", "protolife+legs+nolungs+noplacenta+hair+end",
 "1010", "protolife+legs+nolungs+placenta+nohair+end",
 "1011", "protolife+legs+nolungs+placenta+hair+end",
 "1100", "protolife+legs+lungs+noplacenta+nohair+end",
 "1101", "protolife+legs+lungs+noplacenta+hair+end",
 "1110", "protolife+legs+lungs+placenta+nohair+end",
 "1111", "protolife+legs+lungs+placenta+hair+end",
};



private static   String   seed3 =
"protolife";

//Invented cladistic evolutionary process=
//protolife ->> + legs >> + lungs >> + placenta  >>  + hair
private static String[] Input3 =
{
 "0", "protolife+end",
 "1", "protolife+legs+end",
 "00", "protolife+end",
```

```
 "01", "protolife+lungs+end",
 "10", "protolife+legs+end",
 "11", "protolife+legs+lungs+end",
 "000", "protolife+end",
 "001", "protolife+placenta+end",
 "010", "protolife+lungs+end",
 "011", "protolife+lungs+placenta+end",
 "100", "protolife+legs+end",
 "101", "protolife+legs+placenta+end",
 "110", "protolife+legs+lungs+end",
 "111", "protolife+legs+lungs+placenta+end",
 "0000", "protolife+end",
 "0001", "protolife+hair+end",
 "0010", "protolife+placenta+end",
 "0011", "protolife+placenta+hair+end",
 "0100", "protolife+lungs+end",
 "0101", "protolife+lungs+hair+end",
 "0110", "protolife+lungs+placenta+end",
 "0111", "protolife+lungs+placenta+hair+end",
 "1000", "protolife+legs+end",
 "1001", "protolife+legs+hair+end",
 "1010", "protolife+legs+placenta+end",
 "1011", "protolife+legs+placenta+hair+end",
 "1100", "protolife+legs+lungs+end",
 "1101", "protolife+legs+lungs+hair+end",
 "1110", "protolife+legs+lungs+placenta+end",
 "1111", "protolife+legs+lungs+placenta+hair+end",
};


//Fancy cladistic evolutionary process=
//protolife ->> + eyes >> + lungs >> + placenta  >>  + hair
private static String[] Input4 =
{
 "0", "protolife+end",
 "1", "protolife+eyes+end",
 "00", "protolife+end",
 "01", "protolife+lungs+end",
 "10", "protolife+eyes+end",
 "11", "protolife+eyes+lungs+end",
```

```
 "000", "protolife+end",
 "001", "protolife+placenta+end",
 "010", "protolife+lungs+end",
 "011", "protolife+lungs+placenta+end",
 "100", "protolife+eyes+end",
 "101", "protolife+eyes+placenta+end",
 "110", "protolife+eyes+lungs+end",
 "111", "protolife+eyes+lungs+placenta+end",
 "0000", "protolife+end",
 "0001", "protolife+hair+end",
 "0010", "protolife+placenta+end",
 "0011", "protolife+placenta+hair+end",
 "0100", "protolife+lungs+end",
 "0101", "protolife+lungs+hair+end",
 "0110", "protolife+lungs+placenta+end",
 "0111", "protolife+lungs+placenta+hair+end",
 "1000", "protolife+eyes+end",
 "1001", "protolife+eyes+hair+end",
 "1010", "protolife+eyes+placenta+end",
 "1011", "protolife+eyes+placenta+hair+end",
 "1100", "protolife+eyes+lungs+end",
 "1101", "protolife+eyes+lungs+hair+end",
 "1110", "protolife+eyes+lungs+placenta+end",
 "1111", "protolife+eyes+lungs+placenta+hair+end",
};

//Fancy cladistic evolutionary process=
//protolife ->> + eyes >> + lungs >> + placenta  >>  + hair
private static String[] Input5 =
{
 "0", "protolife+end",
 "1", "protolife+eyes+end",
 "00", "protolife+end",
 "01", "protolife+lungs+end",
 "10", "protolife+eyes+end",
 "11", "protolife+eyes+lungs+end",
 "000", "protolife+end",
 "001", "protolife+placenta+end",
 "010", "protolife+lungs+end",
 "011", "protolife+lungs+placenta+end",
```

```
  "100", "protolife+eyes+end",
  "101", "protolife+eyes+placenta+end",
  "110", "protolife+eyes+lungs+end",
  "111", "protolife+eyes+lungs+placenta+end",
  "0000", "protolife+end",
  "0001", "protolife+hair+end",
  "0010", "protolife+placenta+end",
  "0011", "protolife+placenta+hair+end",
  "0100", "protolife+lungs+end",
  "0101", "protolife+lungs+hair+end",
  "0110", "protolife+lungs+placenta+end",
  "0111", "protolife+lungs+placenta+hair+end",
  "1000", "protolife+eyes+end",
  "1001", "protolife+eyes+hair+end",
  "1010", "protolife+eyes+placenta+end",
  "1011", "protolife+eyes+placenta+hair+end",
  "1100", "protolife+eyes+lungs+end",
  "1101", "protolife+eyes+lungs+hair+end",
  "1110", "protolife+eyes+lungs+placenta+end",
  "1111", "protolife+eyes+lungs+placenta+hair+end",
 };


 //Number of reproductive cycles
 private static int numberOfGenerations = 4;

 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;



//Evolution is recorded into a tree.
```

```
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.

 private static int numberOfLeaves =
 (int) Math.pow(2, numberOfGenerations);

 private static String[] Record =  new String[4*numberOfLeaves];
 //Last used position  in Record[]
 private static int lastUsedPosition;
```

```
/*The alphabet needed by fasta format.
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
                   */

//letters per n-gram
private static  int n=1;
//Number of n-grams
private static  int N = (int) Math.pow(lengthAlf, n);


//The frequency of appearance of n-grams in the text.
//If n>3,  the assigned memory must be expanded
private static double[]  freq1, freq2
                               = new double[N];


//Y-coordinates of points where strings are drawn
private static int[] yCoordinate = new int[4*numberOfLeaves];
//Position in Record[] of ancestral nodes
private static int[] Ancestors = new int[numberOfGenerations];
```

```
//The color to paint
private static Color color = Color.red;

//Reset = true for evolution anew.
//Reset = false when ancestors are to be found.
private static Boolean reset = false;
private static int numberOfAncestors;
private static Boolean printMore1 = false;
private static Boolean printMore2 = true;
//Position in Record[] of clicked node.
private static int place;


public TestNearest5()
{
 super(new BorderLayout());


 JPanel instructionPanel = new JPanel(new GridLayout(0,1));

 //Set up the instructions.
 JLabel instructionsLeft = new JLabel(
     "Click left mouse button to reveal ancestors.");
 JLabel instructionsRight = new JLabel(
  "Click right mouse button to reset. ");
 JLabel advertisment = new JLabel(
   "Nearest neighbord's principle + n-gram distance.");

 instructionPanel.setFocusable(true);
 instructionPanel.add(instructionsLeft);
 instructionPanel.add(instructionsRight);
 instructionPanel.add(advertisment);
 instructionsRight.setEnabled(false);

 //Set up the drawing area.
 drawingPane20 = new drawingPane20();
 drawingPane20.setBackground(Color.white);
 drawingPane20.addMouseListener(this);

 //Put the drawing area in a scroll pane.
```

```
 JScrollPane scroller = new JScrollPane(drawingPane20);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Find the closest leaf to the spot of clicking.
private static int findClosestLeaf(int y)
 {
  int distance = 1000000;
  int position = 0;
  for(int i = 0; i < numberOfLeaves; i++)
   {
//pixels from clicked spot to a y-coordinate of a string
int pixels = Math.abs(y-yCoordinate[i]);
//The closest leaf is found
if ( pixels < distance )
  {
   distance = pixels;
   position  = i;
  }
   }
  return position;
 }



public void mouseClicked(MouseEvent e) { }



public void mousePressed(MouseEvent e) { }



public void mouseReleased(MouseEvent e)
{
```

```
 //Reset?
 if (SwingUtilities.isRightMouseButton(e))
  {
   //????
  }
  //Unveil ancestors?
 if (SwingUtilities.isLeftMouseButton(e))
  {
//Coordinates of clicked spot
int x = e.getX();
int y = e.getY();
//System.out.println( x + " " + y);
//Closest node is found
int yPlace = findClosestLeaf(y);
//System.out.println("================ " + yPlace);
//System.out.println("Clicked Position = " + yPlace);
//System.out.println("Path = " + Record[2*yPlace]);
reset = false;
//place = position in Record[] of the clicked node
//Place is a global variable used by
//the drawing machinery
    place = 2*yPlace;
    repaint();
  }

}



public void mouseEntered(MouseEvent e) { }



public void mouseExited(MouseEvent e) { }

//======Evolution====
```

```
//Input is read
private static void readInput()
{
  int l = Input.length;
  for (int i = 0; i < l; i++)
  Record[i] = Input[i];
  lastUsedPosition = l-1;
}



//=========================================
//============= n-Gram methods ============
//=========================================


//Returns the   n-gram corresponding to index ind
//This is the inverse operation of index()
private static String indexInverse(int n, int ind)
{
char c1 = ' ';
int p = 0;
String ngram = "";
int k = 0;
int indl = ind;

//System.out.println("n = " + n);
for(int i = n-1; i > -1; i--)
{
//System.out.println("**** \nIndex = " + indl);
k = ( int ) (Math.pow(lengthAlf, i));
//System.out.println("k = " + k);
if( k <= indl)
{
p = indl / k;
//System.out.println("p = " + p);
if (p < Alphabet.length())
c1 = Alphabet.charAt(p);

indl = indl - p*k;
```

```
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

}
return ngram;
}


 //Returns the index   of ngram in F[] vectors.
 //An n-gram is a string with n chars taken from Alphabet.
 //Take it as a number in base = length of the Alphabet.
 //The index is that number in base 10.
 private static int index(int n, String ngram)
 {
  char c1 = ' ';
  int p = 0;
  int ind = 0;
  boolean correct = true;
  for(int i = 0; (i <n) & correct; i++)
   {
 c1 = ngram.charAt(i);
 p = Alphabet.indexOf(c1);
 if (p>=0)
  ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
 else
 {
  ind = -1;
  //Detecting the presence of an alien symbol
  correct = false;
 }
   }
  if (printMore1)
  System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
  return ind;
 }

 /*
```

```
This procedure records in F the
frequency of appearance of n-grams in the given Text.
Each n-gram is given an index, a number that
identifies it in the  vector F.
At the beginning, every entry of F  is set to 0.
The index of each n-gram of the Text is
calculated and the corresponding entry in F  is
incremented by one.
*/
private static double[] nGramsFrequency(int n,
       StringBuffer Text)
{
 //Zeroed initialization
 double[] F = new double[N];
 int nText = Text.length();
 //System.out.println("Length  of working text = " + nText);
 int start = 0;
 int ind = 0;
 //Number of possible n-grams
 int nPossibleNGrams = nText-n+1;
 for(int c=0; c < nPossibleNGrams; c++)
  {
   String ngram =  Text.substring(start, start+n);
   //System.out.println("String ngram = " + ngram);
   ind = index(n,ngram);
   //if a given char of an n-gram is not in
   //the alphabet, its index is negative:
   //you must enlarge the alphabet else ignore it.
   //We ignore it.
   if (ind >= 0)   F[ind] = F[ind] + 1;
   start = start +1;
  }
 for(int i = 0; i < N; i++)
 F[i] = F[i]/nPossibleNGrams;
 double sum = 0;
 for(int i = 0; i < N; i++)
 sum = sum + F[i];
 //System.out.println("Frequencies add up to "+ sum);
 return F;
}
```

```
 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
              double[] F1, double[] F2)
 {
  double sum = 0;
  double d = 0;
  String nGram = "";
  if (printMore1)
System.out.println("\nRelative frequency " +
"vectors to compare");
  for(int i = 0; i < N; i++)
   {
double a = F1[i]-F2[i];
sum = sum + a*a;

if (printMore1)
{
nGram = indexInverse(n, i);
 System.out.println( i + "  " +  nGram + "  " +
                  F1[i] + "  " + F2[i]);
 }
}

  //Innocuous bug corrected
  d = Math.sqrt(sum);


  return d;
 }



//=========================================
//=========== Ancestral tracer ===========
//=========================================
```

```
private static int memory = N;
//Number of nodes of the last generation
private static int numberOfFatherNodes = 0;
//Position of nodes of the previous generation in Record[]
private static  int[] PreviousGen = new int[memory];

//Nodes of the previous generation is found
//and the difference of each one of them
//to the  node under test is measured.
public static void
           findPreviousGeneration(int position)
{
int counter = 0;
//Number of bits in the node under test
int lengthThis = Record[position].length();
//The previous generation consists in those  nodes
//whose length = lengthThis-1

//System.out.println("Previous generation" );
for(int i = position; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  String previous = Record[i];
  //If node belongs in the previous generation
  if( previous.length() == lengthThis - 1)
 {
  PreviousGen[counter] = i;
  counter++;
  //System.out.println("Previous = " + Record[i]);
 }
}
numberOfFatherNodes = counter;
}


//The n-gram distance of the node under test to
//nodes of the previous generation is measured.
```

```
//That node with minimum distance is single out.
public static int   findDirectAncestor(int position)
{
 double minDistance = 1000000000;
 int ancestorPosition = 0;
 StringBuffer s1 =
 new StringBuffer(Record[position+1]);
 /*System.out.println("String under   test = "
 + s1.toString());*/

 freq1 = nGramsFrequency(n, s1);

 for(int i = 0; i < numberOfFatherNodes; i++)
 {

  int j = PreviousGen[i];
  /*System.out.println("\n Member of previous generation:"
  + PreviousGen[i] + Record[j+1]);*/
  StringBuffer s2 =
 new StringBuffer(Record[j+1]);
  //System.out.println("\n  text 2:" + s2);
  //System.out.println("\n" +  n + "-grams of   text 2:");
  freq2 = nGramsFrequency(n, s2);
  Double dist = euclideanDistance(N, freq1,freq2);
  //System.out.println(n + "-distance = " + dist);
  if (dist < minDistance)
   {
  minDistance = dist;
  ancestorPosition = j;
   }
  }
return ancestorPosition;
}

 //print ancestors
private static void  printAncestors(int position)
{
 //Number of bits in the node under test
 //= number of generations
 int lengthThis = Record[position].length();
```

```
 if (printMore2)
  for (int i = 0; i < lengthThis; i++)
   {
 System.out.println("Direct ancestor       = "
            + Record[Ancestors[i]] );
   }
}


//A node is addressed by position.
//Its ancestors are found by means of the n-gram technology.
//One can verify the answer directly:
//the ancestor of 11101 is 1110 and
//that of 1010100 is 101010. And so on.
private static void findAncerstors(int position)
{
 //Number of bits in the node under test
 //= number of generations
 int lengthThis = Record[position].length();
 numberOfAncestors = lengthThis;
 int ancestorPosition = position;
 Ancestors[0] = ancestorPosition;
 for (int i = 0; i < lengthThis-1; i++)
  {
findPreviousGeneration(ancestorPosition);
ancestorPosition = findDirectAncestor(ancestorPosition);
Ancestors[i+1] = ancestorPosition;
  }
 printAncestors(position);
}



//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
public class drawingPane20 extends JPanel
```

```
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
```

```
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }


//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
```

```
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area =
 new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane20.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane20.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 color = Color.red;
 int counter = 0;
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 yCoordinate[counter] = verticalSize - y00;
 //counter++;
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
```

```
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
    }
  numberOfLeaves = counter;
 }

//Those strings that correspond with the ancestors
//are redrawn in another color.
private void drawAncestors(Graphics2D g2d, Color color)
{
g2d.setColor(color);
//Seed is drawn
g2d.drawString(seed,x00,verticalSize - y00);
//Strings are drawn:
for(int i = 0; i < numberOfAncestors; i++)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[Ancestors[i]];
 String s = Record[Ancestors[i]+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
```

```
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);

  }
}

//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
 g2d.setColor(Color.red);
 //Draw stem
 g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
 sizeOfTree(g2d);
 drawTree(g2d);
 drawStrings(g2d);


 if (reset)//Evolution anew
   {
 if (printMore2)
     System.out.println("\nThe direct ancestor of  " +
       Record[lastUsedPosition-1] + " is: ");
 findAncerstors(lastUsedPosition-1);
 color = Color.blue;
 drawAncestors(g2d,color);
```

```
}
  else //draw ancestors
   {
if (printMore2)
  System.out.println("\nThe direct ancestor of  " +
                     Record[place] + " is: ");
findAncerstors(place);
color = Color.blue;
drawAncestors(g2d,color);
   }
}

//Graphical  master : it is called automatically
//under starting and by repaint()
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  //Font to draw strings
  Font font = new Font("Arial", Font.PLAIN, 12);
  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
  drawBasic(g2d);
 }


}//End of inner class drawingPane20



/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("TestNearest");
```

```
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new TestNearest5();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
 readInput();
 reset = true;
     createAndShowGUI();
     System.out.println( "\nLetters per n-gram = " + n );
     System.out.println( "Lenght of the complete" +
                       " alphabet = " + lengthAlf);
     System.out.println( "Number of n-grams = " + N);
     }
   });
 }

}//End of main class 119 TestNearest5
```

120 ***Exercise.*** *Run the previous program and play with the code. Run the various examples of encoding and bring forth your own ones to confidently agree else disagree with the conclusion of the Author that he will adopt as certain: the function paradigm as restricted to cladistics in black or white is a very easy lab for our technology: a very small divergence is sufficient to enable ancestor recognition*

with certainty.

4.4 Conclusion

We have tested the power of the nearest neighbor's principle to unveil the ancestors of a given node in an evolutionary tree. That principle reads: my father is that being of the just past generation to whom I am the most like. As we needed to measure differences between texts, we used the **n-gram** technology that we worked thoroughly in Vol IX. Our study was restricted to binary trees and to two paradigms: the mutation-model (mutation is alone and all mutations are viable) and the function paradigm (structures have a function, a meaning). We found that our principle works nicely under the mutation-model when the mutation rate per site per generation is 0.1 or less. But it is in general not useful neither under the function paradigm nor when the mutation rate is high, say, 0.8. The reason in both cases is that evolution might include changes in one generation that represent sufficiently large n-gram distances that break the idea that my father is that man that is closest to me. So, what is common under the mutation model is in general not viable under the function paradigm and vice verse, what is common under the function paradigm can be attained under the mutation model but only as and extremely improbable coincidence. In spite of the general incompatibility with the function paradigm, we found enough evidence to consider that our technology is really excellent to trace back ancestors in a black and white genetic cladistic world, one in which evolution goes from simplicity to complexity by adding more and more functional genes. These conclusions were reached thanks to rather simple simulations of evolution of texts in English. In fact, human languages are a very natural testbed to study the function paradigm because the function of a human language, as well as that of the genome, is to be software, i.e., to convey verbal instructions to make things.

Chapter 5

Phylogenetic trees

`Robust particular evolutionary theories`

121 ***Introduction and purpose.*** *The nearest neighbor's principle endowed with the n-gram distance was shown in the previous chapter to be perfect to single our the direct ancestor along an evolutionary process under certain constrains. Concretely, under the mutation model when mutation rate is low or under the function paradigm in a genetic cladistic world. These results encourage us to pursue in this chapter a most difficult construction: to propose the most robust Evolutionary Theory in the form of a* ***unrooted phylogenetic tree*** *that relates a group of living species. The tree must be unrooted, lacking a well defined direction of time, because our distance matrices are symmetric and so they cannot decide which is the father and which the son. Besides, we suppose that speciation is a rare splitting event so that it can be fairly represented by a binary tree.*

5.1 Our problem

We lack at present an accepted definition of species. So, a tentative definition of species for sexual beings is the following: it is a maximal group of living beings that can mate and reproduce. Or, constructively, begin with a family and join every specimen that eventually would produce progeny if it mated with a member in that family. The resultant group is a species. This definition is evolutionary: if mating is impossible in within members of two groups or if it does not lead to reproduction, to the appearing of viable offspring, then the two groups will diverge in time. Divergence will eventually lead to morphological, biochemical and ethological separation. One can theoretically load this definition with Evolutionism as follows: one can assume by dogma that all species arouse by this mechanism and

interpret the world accordingly. Else, one can realize that evolution is a reality and so to produce functional complexity it must pass over too many trials that must be evidenced by malfunctions and malformations as in the fossil record as in extant populations. Since this prediction is not supported by nature, the Evolutionary Theory is obviously false. These is certain for both Darwinian and Lamarckian flavors.

Nevertheless, there should be no surprise if evolution turns to be the correct explanation of the origin of many groups of species, say, of flies. Therefore, the following problem makes sense in our community: can we forward a particular evolutionary theory in the form of an evolutionary tree to postulate an evolutionary interrelatedness of a given group of species that live just now?

Actually, we do not mind anyone theory, but one that could be the most difficult to falsify. Our optimism has been fired by the positive results found in the previous chapter: with low mutations rates under the mutation-model as well as for the genetic cladistics that incarnates the function paradigm, our n-gram technology promises to correctly find the direct ancestor of a given species by analyzing DNA strings, proteomes or texts that describe species in the same language.

122 *The evolutionary tree problem*

In concrete terms, we will deal with the next optimization problem:

The distance matrix to tree problem: given a matrix of pairwise n-gram distances among objects that may represent living species, find the unrooted tree that best fits it.

Now, we are very proud of our n-gram technology because it does not need those so difficult and slippery alignment procedures that cause so much suffering to practitioners of standard approaches. Nevertheless, as we will see just below, we also have our own challenges. To begin with, we have no idea of the complexity of the tree space over which we will work. This means that it is mandatory to choose at the beginning a very general procedure to eventually pass next to possible sophistications. Then, which is our plan?

We will develop a general machinery under very simple but clear concepts. So, we combine technical and possibly local problems with the vision of an architect.

Notice that we cannot use the nearest neighbor's principle because we have the species that live at presently but we lack their direct ancestors. In the distance approach, ancestors are not synthesized but only the form of the tree that postulates their evolutionary relationship together with the distances in withing neighboring nodes.

5.2 Technicalities

To propose a solution to the distance matrix to tree problem we need to solve various technical questions in order to restate our problem in very precise and clear terms that it could be solved automatically by a Java program.

So, let us begin.

123 ***A new depiction for binary trees.***

For the problem of finding the evolutionary tree that best fits a distance matrix, emphasis is on distances. So, we will depict trees in a way to include such information: the length of the links represent the distance between adjacent nodes and so lengths are anymore uniform. In fact, under the function paradigm it is expected that such distances may have a very large variability: diverse habitats and diverse functional constraints are enough to create very different rates of divergence.

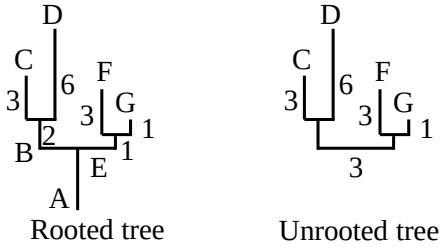


Figure 5.0. The notation for trees in this chapter: every end of a line represents a node, a species. Every vertex where tree lines meet together represents a splitting: a species splits in two. Numbers along links indicate distances between connected nodes. Lengths are best imagined to be connected with vertical strokes while horizontal ones have no length. In a rooted tree, there is a direction of time: the root at the bottom indicates the general ancestor and evolution goes upwards.

In an unrooted tree there is no time direction. A distance matrix is symmetric and therefore cannot include information about the parent-offspring relationship. So, the tree that is inferred from a distance matrix alone is unrooted.

124 ***Entangled terms.*** *Clarity would be enhanced if we make a distinction between the next two terms: the topology of a tree and a tree. By* ***topology of a tree*** *we refer to the form of the tree, in which one depicts how the nodes are linked. By a* ***tree*** *we mean a topology plus lengths of links. Nevertheless, we will use the word tree to mean both concepts, a topology and a topology with lengths. The correct meaning will in general clear from the context.*

125 ***From trees to distance matrices.***

Suppose we are given a depiction of a tree in the notation above. We can use that tree to fill in a distance matrix if only we obey the following rule: the distance between a given pair of nodes equals the sum of the distances along the direct path that connect them.

126 ***Example:*** *The unrooted tree above generates the next distance matrix for the set of nodes* $\{B,C,D,E,F,G\}$*:*

$$D_T = \begin{pmatrix} & B & C & D & E & F & G \\ B & 0 & 5 & 8 & 2 & 6 & 4 \\ C & 5 & 0 & 9 & 5 & 9 & 7 \\ D & 8 & 9 & 0 & 8 & 12 & 10 \\ E & 2 & 5 & 8 & 0 & 4 & 2 \\ F & 6 & 9 & 12 & 4 & 0 & 4 \\ G & 4 & 7 & 10 & 2 & 4 & 0 \end{pmatrix}$$

This procedure outputs the ***associated matrix*** D_T *to the tree* T*, the unrooted tree of previous figure.*

127 Distance matrices associated to leaves

Our general interest is to propose a phylogenetic tree for a set of species that live around us. They will appear in a tree as its leaves, i.e., as terminal nodes apart from the root. We define D_L *the associated distance matrix restricted to leaves as the restriction of* D_T *to the leaves of* T*. Say, for the unrooted tree above, the leaves are* $\{C,D,F,G\}$ *and the corresponding distance matrix is:*

$$D_L = \begin{pmatrix} & C & D & F & G \\ C & 0 & 9 & 9 & 7 \\ D & 9 & 0 & 12 & 10 \\ F & 9 & 12 & 0 & 4 \\ G & 7 & 10 & 4 & 0 \end{pmatrix}$$

128 ***Distance between a pair of matrices***

If we have two matrices M and N with the same number of rows and columns and with entries M_{ij} and N_{ij}, the **absolute distance** between them is defined as the Euclidean distance:

$$d(M,N) = \sum |M_{ij} - N_{ij}|$$

129 ***Example.*** *Let M and N be the two following matrices:*

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$N = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

The distance is

$$d = |1-5| + |2-6| + |3-7| + |4-8| = 4 \times 4 = 16$$

130 ***Technical restating of to the distance matrix to tree problem***

We are given a set of species L that live together with us and whose genomes have been sequenced or whose ethologies or anatomies or physiologies have been described and appear in appropriate texts. The n-grams distance between corresponding pair of texts is consigned in matrix D_L. Our problem is to find the best fitting tree T to D_L. The next procedure will solve the problem:

1. We construct all possible trees with the given set of leaves L and to each link we associate a length.

2. Next, each tree generates a distance matrix among leaves.

3. Last, we must look for the tree with associated distance matrix D such that $d(D,D_L)$, the absolute distance between matrix D and the target matrix D_L, is a minimum.

We can run an immediate test: we must end with the same tree if we go forth from a tree to a distance matrix and back to the best fitting tree. Formally:

131 ***Requirement:*** *The best fitting to D_L, the associated distance matrix to a tree T with set of leaves L, must be T itself. By abuse of language, we will say the best fitting to D_T must be T itself.*

Now, evolutionary tress are iconic to evolution. Therefore and by our honor, this problem must be solved by evolution, which is a general method for solving problems. Is our choice wise? We think that it is and to justify our decision a reason may suffice: the microbial world relies on evolution to permanently threaten our lives. But in spite of all our intelligent efforts to the contrary, dangerous microbes go dancing rampantly. So evolution is worth of the most high respect.

As usual, we will carefully test our code for correctness. So, from the very beginning we will attack the posed requirement: we generate a tree at random, calculate the associated distance matrix which becomes the target matrix to find next the best fitting tree to this matrix. We must end with the target matrix.

So, let us proceed to develop a rather heavy duty machinery.

132 ***Encoding a tree into a string.***

The following is an expedite form of encoding a binary tree into a string. Let us introduce it in two steps. In first place we abstract the **topology** or scheme of the tree. To fix ideas, let us consider the unrooted tree above:

We begin by listing the leaves. Next, we pay attention to those pairs of leaves that form part of a U: such is the case of leaves C and D. The corresponding U is denoted as (C,D). By the same token (F,G) indicates that the two leaves F and G form part of a U. Next, the two U's can be associated into a super-U

((C,D),(F,G))

This notation renders information about the **topology** of the tree, i.e., of the graph of the tree without lengths of links.

This is (part of) the NEWICK representation of phylogentic trees (Salemi, [8] 2009). We find this notation simple and powerful. So, we will use it in our Java programs.

133 ***Can we generate all possible trees?***

We need to make sure that we can generate all possible topologies and trees for unrooted trees. So, let us begin with the simplest tree that has three leaves A,B and C. They conform a unique tree that can be denoted variously:

(A,(B,C)) = (B, (C,A)) = (C, (A,B))

Let us consider now n = 4 leaves A, B, C and D. All possible unrooted trees are:

((A,B), (C,D))
((A,C), (B,D))
((A,D),(B,C))

Let us notice that each leaf has three possibilities to form a U in the tree. That is why we have tree different trees. Now, they can be generated by recursion to level 3, i.e., departing from ((A,B),C). To that aim, we follow the next rule: to generate a new tree, a letter L is replaced by (L,D). This rules produces 3 trees.

(((A,D),B),C) = ((A,D),(B,C))
((A, (B,D)),C) = ((B,D),(A,C)) = ((A,C), (B,D))
((A,B), (C,D))

What does happen with a set of 5 leaves A, B, C, D, E?

Trees generated from (((A,D),B),C) by a replacing a letter L by (L,E):

((((A,E),D),B),C) = ((((A,E),D), (B,C)): D is alone
(((A,(D,E)),B),C) = (((A,(D,E)), (B,C)): A is alone
(((A,D), (B,E)),C): C is alone
(((A,D),B), (C,E)): B is alone

We see that we lack the case in which E, the new leaf, is alone. This means that the rule of replacing a letter or node L by a pair with the new leaf, (L,E), is insufficient to generate all trees. The correct protocol is given by the **rule of the gardener**: graphs are inserted into links (extracted from Swofford et al, 2009, [10] pag 278). So, when we include a new leaf, we get a new tree for each link. Therefore, we need to change our notation to include links. This is done as follows:

nodes are denoted by letters, say A, and links by a pair of letters, say, AB links A and B together.

The tree ((A,B),C) has three leaves and an internal node, let it be I. So, it has three links AI, BI, CI.

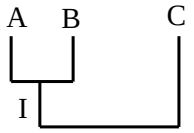


Figure 5.1. The tree ((A,B),C) has an internal node, I, and three links: AI, BI and CI.

What does happen if we graft leaf D to tree ((A,B),C)?

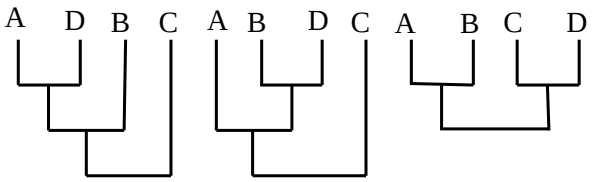


Figure 5.2. Leaf D can be grafted into tree ((A,B),C) in three different forms.

When we graft leaf D into each link we get:

((A,B),C) = ((AI,B),C) = (((A,D),B),C) = ((A,D),(B,C))
((A,B),C) = ((A,BI),C) = ((A, (B,D)),C) = ((B,D),(A,C))
((A,B),C) = ((A,B),CI) = = ((A,B), (C,D))

Let us use the gardeners's rule over the tree (((A,D),B),C) to insert leaf E. So, lets find the links of this tree:

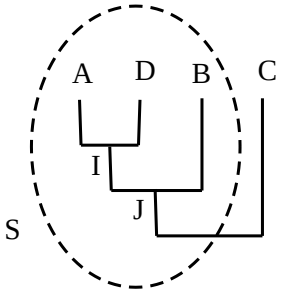


Figure 5.3. The tree (((A,D),B),C) has two internal nodes, I and J. Node I generates three links AI,DI and BI. Node J generates 2 links SJ and CJ.

The unrooted subtree S = (A,D),B) has an internal node, let it be I. So, we have here three links succinctly denoted ((AI,DI),BI). On other hand, the composite structure S is linked to C through an internal node, let it be J. So, (S,C) corresponds with links (SJ,CJ), from which we get other two links: 5 in total that indeed give rise to 5 inequivalent trees.

As a rule, if we graft a new leaf to a link, we add two more links. This allows us to count how many trees we have:

For 3 leaves we get 1 tree with three links. A new leaf can be added in 3 ways, so we get 3 trees with four leaves, each one with five links. Therefore, for each topology with 4 leaves there are 5 possible derived trees with five leaves, each one with seven links. Hence, for 5 leaves we have $1 \times 3 \times 5 = 15$ trees at all. By the same token, for 6 leaves we get $1 \times 3 \times 5 \times 7 = 105$ trees. For 7 leaves: there are $1 \times 3 \times 5 \times 7 \times 9 = 945$ trees. For $n = 14$ wee will have over 5×10^{12} trees.

We see that we face a combinatorial explosion. In other words, we are assuming that we have no restrictions about computer time or memory.

134 *The generative grammar*

We are trying to invent a set of automatic rules or a **generative grammar** to list all trees by recursion.

We have arrived to the conclusion that links are what matter. Thus, let us consider the tree ((((A,E),D),B),C): can we rewrite it as ((((AI,EI),DI)I,BI)I,CI) to make explicit its links? This guess is pretty compatible with the expected 7 derived trees we must have. Let us consider now the tree (((A,D), (B,E)),C) which also must have 7 derived trees: it we rewrite it as (((AI,DI)I, (BI,EI)I)I,CI), we would get 8 links. But this cannot be: we must be more careful.

Let us consider the next succession of trees that represent an evolutionary branch: we add new nodes D,E and F in tandem to the tree ((A,B),C):

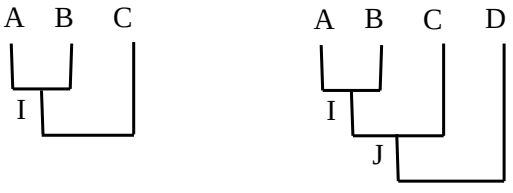


Figure 5.4. We see to the left the tree ((A,B),C) that also can be represented as ((AI,BI),CI), where the letter I stands for the internal node that glues everything. Links are explicitly represented by a pair of letter, say, AI is the link that joins A and I. To the right we see the tree (((A,B),C),D) = (((AI,BI)J,CJ),DJ)

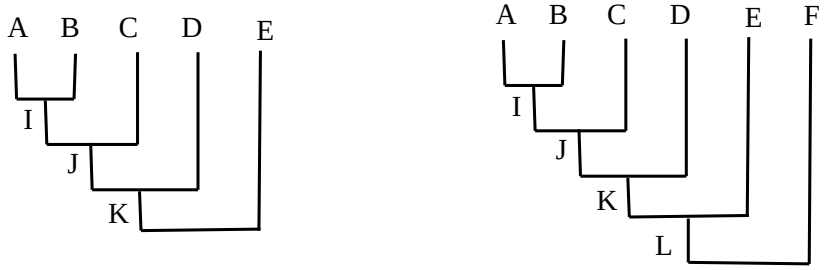


Figure 5.5. We see to the left the tree ((((A,B),C),D),E) = ((((AI,BI)J,CJ)K,DK),EK) To the right we see the tree (((((A,B),C),D),E),F) = (((((AI,BI)J,CJ)K,DK)L,EL),FL)

We get the following succession of trees:
((A,B),C) = ((AI,BI),CI)
(((A,B),C),D) = (((AI,BI)J,CJ),DJ)
((((A,B),C),D),E) = ((((AI,BI)J,CJ)K,DK),EK)
(((((A,B),C),D),E),F) = (((((AI,BI)J,CJ)K,DK)L,EL),FL)

This process can be generated automatically if we impose the following rule:
to append the leaf F at the new internal node L to (((S)K,DK),EK) where S denotes a subtree S, replace the whole string by ((((S)K,DK)L,EL),FL). This rule is denoted as

$(((S)K,DK),EK) -> ((((S)K,DK)L,EL),FL).$

Let us consider now another evolutionary branch: we graft the same new nodes but at different links:



Figure 5.6. We see to the left the tree ((A,B),C) = ((AI,BI),CI). To the right we see the tree (((A,D),B),C) = (((AJ,DJ)I,BI),CI).

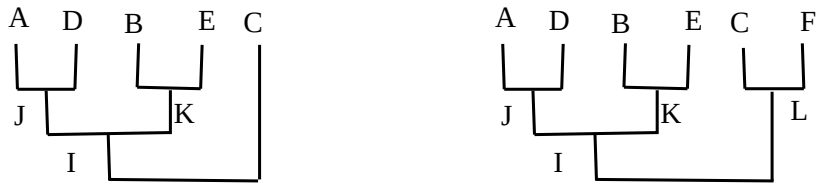


Figure 5.7. We see to the left the tree (((A,D),(B,E)),C) = (((AJ,DJ)I,(BK,EK)I),CI). And to the right the tree (((A,D),(B,E)),(C,F)) = (((AJ,DJ)I,(BK,EK)I),(CL,FL)I).

This second process renders:
((A,B),C) = ((AI,BI),CI).
(((A,D),B),C) = (((AJ,DJ)I,BI),CI).
(((A,D),(B,E)),C) = (((AJ,DJ)I,(BK,EK)I),CI).
(((A,D),(B,E)),(C,F)) = (((AJ,DJ)I,(BK,EK)I),(CL,FL)I).

This process can be generated automatically by the following rule:
If we have a link AI that connect leaf A and internal node I, replace the letter that represents the leaf by (AN,FN). Succinctly:

$AI -> (AN,FN)I$

If we compare the two given examples, we find that the first rule is by far more complicated than the second. So, let us try a simplification: (((S)K,DK),EK) also can be denoted as ((S)K, (DK,EK)) in the very same way as ((AI,BI),CI) can be denoted as (AI,(BI,CI)). If we use the notation ((S)K, (DK,EK)), then the rule

$(((S)K,DK),EK) -> ((((S)K,DK)L,EL),FL).$

can be replaced by the simpler rule:

$((S)K, (DK,EK)) -> ((S)K, (DK,(EL,FL)K))$

which is also covered by our former rule

$AI -> (AN,FN)I$

This is a simplification in the only case in which the string (((S)K,DK),EK) does not appear. Otherwise we need the additional rule:

(((S)K,DK),EK) $->$ ((S)K, (DK,EK))

which is also complicated.

Let us pass now to study the next graphic that unveils a possible problem of our notation:

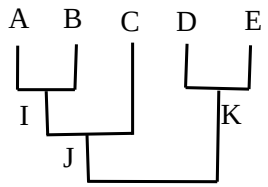


Figure 5.8. Shall this tree be denoted as (((AI,BI)J,CJ)K, (DK,EK)J)?

If we explicitly write all links, we must denote the tree of this graphic as (((AI,BI)J,CJ)K, (DK,EK)J). The problem is that it has 8 letters that denote links while the tree has only 7 links. The remedy is to identify the redundant letter and to follow the notation for the simple tree ((A,B),C) = (A,(B,C)). Its links are denoted naturally as ((AI,BI),CI) = (AI,(BI,CI)) and not as ((AI,BI)I,CI) or (AI,(BI,CI)I). If we follow the natural notation, the tree of the graphic must be denoted as (((AI,BI)J,CJ)K, (DK,EK)) where we have deleted the final J because it is redundant: the three K's already show us that there is a central node K to which three entities are linked: leaves D and E and substree ((AI,BI)J,CJ).

We are ready now to propose a procedure to generate automatically all possible trees from the simple tree ((A,B),C). It has two rules. To fix ideas, let us imagine that we need to insert the leaf F to the new internal node N:

1. If we have a link AI that connects leaf A and internal node I, replace the letter that represents the leaf by (AN,FN). This rule is denoted as

AI $->$ (AN,FN)I

2. If we have a link (S)I that connects a subtree S and internal node I, replace the whole substring by ((S)N, FN)I. This rule is denoted as

(S)I $->$ ((S)N, FN)I

Rule 2 is a generalization of rule 1. So, most possibly we will end with rule two only.

135 ***Definition.*** *A set of substitution rules, such as our rule 1 and 2 just mentioned, is called a* ***generative grammar*** *or a* ***grammar*** *for short.*

136 ***Exercise.*** *Test the correctness and sufficiency of our aforementioned grammar by generating all 5 possible trees when leaf E is added to the new node K in the tree (((AI,BI)J,CJ),DJ). Make enough drawings to make sure of your conclusions.*

Answer:
136, page 247. The tree (((AI,BI)J,CJ),DJ) reads: nodes A and B are linked to internal node I. Substructure (AI,BI) is linked to internal node J as much as C and D. This tree has five non redundant links, so its offspring has 5 members. We can generate at once four trees if we use rule 1:

AI $->$ (AN,FN)I

for leaf A, new internal node N and old internal node I:

1. ((((AK,EK)I,BI)J,CJ),DJ) generated by AI $->$ (AK,EK)I
2. (((AI,(BK,EK)I)J,CJ),DJ) generated by BI $->$ (BK,EK)I
3. (((AI,BI)J,(CK,EK)J),DJ) generated by CJ $->$ (CK,EK)J
4. (((AI,BI)J,CJ),(DK,EK)J) generated by DJ $->$ (DK,EK)J

We lack a tree that results from grafting new node E into the internal link IJ. If we calculate it with our mind, the result is:

((((AI,BI)K,EK)J,CJ),DJ)

This tree can be generated by rule two: (S)I $->$ ((S)N, FN)I in the form

(S)J $->$ ((S)K, EK)J for S = (AI,BI)

We consider that we have enough ideas to attempt an automatic generation of all possible trees. Let us begin the code development with the implementation of rule 1.

137 ***The code for the implementation of rule 1 of the generative grammar of trees follows. Rule 1: AI $->$ (AN,FN)I. Old link AI that connects leaf A to internal node I is replaced by (AN,FN)I, where F is a new leaf and N a new internal node. Newick's notation.***

```
//Program J137 Topologies
//This program implements rule 1
//to generate trees by the insertion of new branch FN:
//aI -> (aN,FN)I;
//a = leaf in tree;
//I = internal node already in tree;
//N = next Internal node to be inserted;
//F = nextLeaf;
//All   possible trees are generated from tree ((AI,BI),IC)



public class Topologies
{

 //Link's notation for trees
 private static   String   parent =
 "((AI,BI),CI)";
 private static  String leaves =  "ABCDEFGRSTUVWXYZ";
 private static String internalNodes = "IJKLMNOP";


 private static boolean charIsInternalNode(char c)
 {
  String s = "" + c;
  if (internalNodes.contains(s)) return true;
  else return false;
 }

//Link notation -> Newick's notation
 // (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
 String tree = "";
//length of string newTree
```

```
int l = newTree.length();
 for(int i = 0; i < l; i++ )
  {
   //newTree is scanned char by char
  char c = newTree.charAt(i);
  //Is c an internal node to be deleted?
  if (charIsInternalNode(c));
   else //add char c to output
    tree = tree + c;
  }
 return tree;
}

private static void generateTrees(String oldTree)
{
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = oldTree.length();
 for(int i = 0; i < l; i++ )
  {
    //oldTree is scanned char by char
char I = oldTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = oldTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')');
  else //if I links to a leaf
   {
```

```
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
          + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree = oldTree.replace(link, replacement);
   }
 System.out.println(newTree + " -> "
                      + simplify(newTree));
}
  }
}


public static void main(String[] args)
{
 System.out.println("Parent tree = \n" + parent
 + " -> "   + simplify(parent));
 System.out.println("Offspring :  "   );
 generateTrees(parent);
}

}//End of main class J137 Topologies
```

138 ***Exercise.*** *Run the previous program and play with the code.*

139 ***Exercise.*** *Add to the previous program an implementation of rule 2: If we have a link (S)I that connects a subtree S and internal node I, replace the whole substring by ((S)N, FN)I, for new leaf F and new internal node N. In short*

$$(S)I -> ((S)N, FN)I$$

To test your code, begin with tree (((AI,BI)J,CJ)K,(DK,EK)J) and generate all 7 trees that result from inserting branch FN with new leaf F and internal node L. *Answer*

140 ***The tree of trees.*** *The data structure of the output of our generative program is that of a tree: with begin with a simple tree and grow complexity by inserting*

one branch at a time. This process creates various children from a parent, which is exactly the structure of a tree. This structure will be referred as the ***tree of trees****. Let us reuse previous software to depict such a tree. We cannot use our binary trees because children are always more than two. Nevertheless, a Java tree will do it. The appropriate guide is program J51 RecordingTree. The code follows:*

```
//Program J140 TreeOfTrees
//Beginning with tree ((AI,BI),CI)
//we generate all possible topologies.
//The resultant data structure is that of a tree.
//It is shown here.

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Enumeration;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;



public class TreeOfTrees extends JPanel
                         implements ActionListener
{

 private static final long serialVersionUID = 1L;
```

```
//Commands for the buttons
private static String RESET_COMMAND = "Reset";


private static   String   seed =
   "((AI,BI),CI)";

private static  String leaves =  "ABCDEFGRSTUVWXYZ";
private static String internalNodes = "IJKLMNOP";

private static boolean print = false;


//Evolution is run for some generations
private static int nGen = 3;

 //DynamicTree is an inner service class that comes
 //at the end of this file.
 private static DynamicTree4 treePanel;


 //Constructor of main class: GUI basics
 public TreeOfTrees()
 {
   super(new BorderLayout());

   // Create the components.
   treePanel = new DynamicTree4();


   //The tree is cleared out.
   JButton clearButton = new JButton("Reset");
   clearButton.setEnabled(false);
   clearButton.setActionCommand(RESET_COMMAND);
   clearButton.addActionListener(this);

   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(1200, 800));
   add(treePanel, BorderLayout.CENTER);
```

```
    //Lay all four buttons in line
    JPanel panel = new JPanel(new GridLayout(1, 6));

    //Add buttons to the panel

    panel.add(clearButton);
    add(panel, BorderLayout.SOUTH);
  }//End of constructor



  public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if  (RESET_COMMAND.equals(command)) {
      // Clear button clicked.
      treePanel.clear();
      evolution();
    } //else if (Random_Change_COMMAND.equals(command)) {
     //A random change is made: add, delete, clear.
     // treePanel.clear();
   // }
    report();
  }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
```

```
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame = new JFrame("TreeOfTrees");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees newContentPane = new TreeOfTrees();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
//length of string newTree
int l = newTree.length();
```

```
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //add char c to output
   tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);
```

```
  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //An evolutionary process under the
  //mutation-model is triggered.
```

```
  //Every leaf (node of the last generation) is subjected
  //to reproduction with a number of children
  //equal or less than 4.
  //Reproduction may undergo mutation in each site.
  private static void evolution()
  {

//Evolution is run for nGen generations
for(int gen = 0; gen < nGen; gen++)
 {
  System.out.println("gen = " + gen);
treePanel.reproductionOfLeaves();
 }
  }


  public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application’s GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        createAndShowGUI();
        evolution();
      }
    });
  }


//=============================================================
//===================Inner service class=======================
//=============================================================


//This program is a modification to the program with
//equal name downloaded from

/*
```

```
 * http://www.java2s.com/
 * Tutorial/Java/0240__Swing/DynamicTree.htm
 *
 * Cited 28/VI/2013.
 *
 * Modifications made by JR of the Evoljava Community.
 * Modifications have unrestricted license.
 *
 * License of the original source code:
 *
 * Copyright (c) 1995 - 2008 Sun Microsystems, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms,
 * with or without
 * modification, are permitted provided that
 * the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer.
 *
 *   - Redistributions in binary form must reproduce
 *     the above copyright
 *     notice, this list of conditions and the
 *     following disclaimer in the
 *     documentation and/or other materials
 *     provided with the distribution.
 *
 *   - Neither the name of Sun Microsystems nor
 *     the names of its
 *     contributors may be used to endorse or
 *     promote products derived
 *     from this software without specific
 *     prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 * HOLDERS AND CONTRIBUTORS "AS
```

```
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

// package components;
/*
 * This code is based on an example provided
 * by Richard Stanford, a tutorial
 * reader.
 */
class DynamicTree4 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
```

```
//counter of generations
int counterGen = 0;


//Constructor
public DynamicTree4()
{
   super(new GridLayout(1, 0));

   rootNode = new DefaultMutableTreeNode(seed);
   treeModel = new DefaultTreeModel(rootNode);

   tree = new JTree(treeModel);
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);

   JScrollPane scrollPane = new JScrollPane(tree);
   add(scrollPane);
 }


 /** Remove all nodes except the root node. */
 public void clear() {
   rootNode.removeAllChildren();
   treeModel.reload();
 }




 /** Add child to the currently selected node. */
 public DefaultMutableTreeNode addObject(Object child) {
   DefaultMutableTreeNode parentNode = null;
   TreePath parentPath = tree.getSelectionPath();

   if (parentPath == null) {
     parentNode = rootNode;
   } else {
```

```
      parentNode = (DefaultMutableTreeNode)
             (parentPath.getLastPathComponent());
    }

    return addObject(parentNode, child, true);
  }



  public DefaultMutableTreeNode
               addObject(DefaultMutableTreeNode parent,
      Object child) {
    return addObject(parent, child, true);
  }



  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                     new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
               (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                   (new TreePath(childNode.getPath()));
    }
    return childNode;
  }
```

```
 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
 System.out.println("counterGen = " + counterGen );

 DefaultMutableTreeNode node = null;
Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
System.out.println("ParentTree = " +  parentTree );

//===============

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
```

```
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree, i,
              nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                      + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...

//System.out.println(node.toString()) ;
   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
```

```
  class MyTreeModelListener implements TreeModelListener {
    public void treeNodesChanged(TreeModelEvent e) {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e) {
    }

    public void treeNodesRemoved(TreeModelEvent e) {
    }

    public void treeStructureChanged(TreeModelEvent e) {
    }
  }
}

}//End of main class J140 TreeOfTrees
```

141 ***Exercise.*** *Run the previous program and play with the code. Run various tests for correctness. For instance, change the number of generations and count the number of trees as a function of the number of leaves. Revise that the strings*

that encode for trees are indeed well formed.

We have done our best to develop a correct program but the price to check the syntax of an expression is high. Now, we hope that to see the generated trees will ease that task. This means that we need a translator from Newick's notation to ours.

142 ***Translating from Newick's notation.***

Our binary notation for trees is very good for recording an unfolding evolutionary process and for drawing the ensuing trees. On the other hand, Newick's notation is very appropriate for automatic generation of the topologies of trees. Our task is therefore to connect our drawing facility to the Newick's notation.

To begin with, let us consider the simplest tree: ((A,B),C). This notation means that A and B are siblings and that they have an ancestor, I, that is also shared by C. That is why we also can denote this tree as ((AI,BI),CI), in where each link is explicitly written as a couple of letters. The problem is that this tree is unrooted while our drawing machinery is addressed for rooted trees. The solution is to graft a root R to, say, link CI:

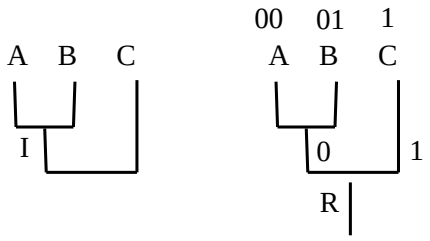


Figure 5.9. The tree ((A,B),C) is unrooted. It is transformed into a rooted tree if a root R is grafted to link IC. In this new rooted tree, every node has an easy to understand binary address, as it is shown to the right: for each splitting, assign a zero to the left and a 1 to the right.

If a root is grafted to the most lowest link, our trees always can be transformed into binary trees in which an assignation of binary addresses is immediate: for each splitting, assign a zero to the left and a 1 to the right.

143 ***Exercise.*** *Invent a form to automatically assign binary addresses to the leaves of a tree encoded in the Newick's notation. Compare it with that of the Author.* *Answer*

Although we have invented some rules to translate from Newick's to binary notation, they must be submitted to test and most possibly we will need to correct them. Therefore, we need a generator of arbitrary trees:

144 ***Exercise. Make a generator of arbitrary trees, i.e., of trees generated at random. Output in Newick's notation.*** *Answer*

With all needed tools at hand, let us develop now the code to make the required translation:

145 ***The following code translates a string that encodes for a tree in Newick's notation to the binary one that is used by us.***

```
//Program J145 NwckToBin
//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.


import java.util.Random;

public class NwckToBin
{

 //Link's notation for trees
 private static   String   parent =
 "((AI,BI),CI)";
 private static  String leaves =  "ABCDEFGHUVWXYZ";
 private static String internalNodes = "IJKLMNOPQRST";
 //Next leaf to be grafted
 private static  int nextLeafIndex = 2;
 //Next internal node to be inserted
 private static  int nextIntNodeIndex = 0;
```

```
 private static  Random r = new Random();

 private static boolean print = false;


 //Detect internal nodes, different than leaves
 private static boolean charIsInternalNode(char c)
 {
  String s = "" + c;
  if (internalNodes.contains(s)) return true;
  else return false;
 }

//Simplification from Link notation to Newick's notation:
 // (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
 String tree = "";
//length of string newTree
int l = newTree.length();
 for(int i = 0; i < l; i++ )
  {
   //newTree is scanned char by char
  char c = newTree.charAt(i);
  //Is c an internal node to be deleted?
  if (charIsInternalNode(c));
   else //add char c to output
    tree = tree + c;
  }
 return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
        char F, char N)
```

```
{
 String newTree = "";
 char I = parentTree.charAt(i);
 if (print)
 System.out.println("i = " + i + " I = " + I);
 //Subtree S of link (S)I must be read
 String S = "";
 //Counter of left parenthesis
 int counter = 0;
 String pl = "" + '(';
 String pr = "" + ')';
 //Position to be scanned
 int position = i;
 boolean go = true;
 //All right parentheses to the near left of I are captured
 while (go)
  {
   position = position -1;
   String a = "" + parentTree.charAt(position);

   if (print)
   System.out.println("a = " + a + " p = " + pr);
   if (a.equals(pr) )
   {
counter++;
S = a + S;
   }
   else go = false;
  }//End of while
 if (print)
 System.out.println("counter = " + counter);
 //The complete subtree is captured:
 //Parentheses must balanced
 while (counter > 0)
  {
   String a = "" + parentTree.charAt(position);
   if (a.equals(pr))   counter++;
   if (a.equals(pl))  counter--;
   S = a + S;
   if (print)
```

```
   System.out.println("Subtree = " + S);
   position = position -1;
   }
 //Replacement rule 2:
 //(S)I -> ((S)N,FN)I.  N= nextIntNode;
 //F = nextLeaf
 //Target to be replaced
 String target  =   S  + I;
 String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(target, replacement);
 //If I links to a complex structure
 return newTree;
}

//From parentTree, a tre in Newick's notation,
//a Child  tree is generated by a generative grammar.
//A random election is made among all possibilities.
private static String generateChild(String parentTree)
{
 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);
 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = parentTree.length();
 //counter of links = 0;
 int counter = 0;
 //Links are counted
 for(int i = 0; i < l; i++ )
  {
   //parentTree is scanned char by char
```

```
   char I = parentTree.charAt(i);
//Is I an internal node that denotes a link?
if (charIsInternalNode(I)) counter++;
  }
 //A link is chosen
 int k = r.nextInt(counter);
 System.out.println("Random number = " + k);
 //new branch is grafted at k
 counter = 0;


 for(int i = 0; i < l; i++ )
  {
    //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  if (counter == k)
   {
//Check for the char before I
char a = parentTree.charAt(i-1);
//If I links to a complex structure
if (a == ')') newTree = useRule2(parentTree, i,
               nextLeaf, nextIntNode);
else //if I links to a leaf
 {
  //Link to be replaced
  link = "" + a + I;
  //Replacement rule 1:
  //aI -> (aN,FN)I;   N= nextIntNode;
  //F = nextLeaf
 replacement = "(" + a + nextIntNode + ","
          + nextLeaf + nextIntNode + ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(link, replacement);
 }
System.out.println(newTree + " -> "
             + simplify(newTree)
             + " -> " + toBinary(newTree));
```

```
 i = l;//End work
   }
  counter++;
}
  }

 return newTree;

}


//A tree at random is generated
private static String generateTree(String parentTree)
{
 int limit = 5; //less than 11.
 for(int i = 0; i < limit; i++ )
  {
   parent = generateChild(parent);
  }
 return parent;
}


//Translation from Newick's to quasi-binary notation
private static String toBinary(String tree)
{

 String regulators = "(,)";
 //Output
 String treeToBi = "";

 String address = "";
 String thisChar = "";
 String nextChar = "";
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
```

```
  {
   thisChar = "" + tree.charAt(i);
   if ( i < l-1) nextChar = "" + tree.charAt(i+1);
   else nextChar = "end";
   //Is thisChar a regulator?
   if (regulators.contains(thisChar))
 {
   treeToBi =  treeToBi + thisChar;

   //This char = (
   if (thisChar.contentEquals("("))
   {
address = address + '0';
if (print)
System.out.println("Address = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
 {
 treeToBi =  treeToBi + address;
 }
   }

 //This char = ,
   if (thisChar.contentEquals(","))
   {
if (print)
System.out.println("Address at ,   = " + address);
int m = address.length();
//replace last 0 by a 1
address = address.substring(0, m-1) + '1';
if (print)
  System.out.println("Address at , 1 = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
  {
 treeToBi =  treeToBi + address;
  }
   }
```

```
 //This char = )
  if (thisChar.contentEquals(")"))
   {
int m = address.length();
//Go one level down the tree
    address = address.substring(0, m-1);
   }
 } //End of   if (regulators.contains(c))
   }//End of for(int i...
 return treeToBi;
}



public static void main(String[] arg)
{
 System.out.println("Parent tree = \n" + parent
 + " -> "    + simplify(parent));
 String tree = generateTree(parent);
 String bTree = toBinary(tree);
 System.out.println("Tree in Newick's notation: \n"
         + tree);
 System.out.println("Tree in quasi-binary notation: \n"
               + bTree);
}

}//End of main class J145 NwckToBin
```

146 ***Exercise.*** *Run the previous program and play with the code. Make your best to confidently agree else disagree with the assessment of the Author: the translating grammar is perfect. In case of disagreement, develop your own correct code.*

147 ***Exercise:*** *Add to the previous code a visor of trees, i.e., merge a previous program that allowed binary trees to be visualized with this generator of trees. Play with the new program enough to make sure that its code is correct. Such certainty is our very aim all around.* *Answer*

We can now merge this program with a previous one:

148 ***The code that generates all trees with the possibility to select one of them for visualization follows.***

```
//Program J148 TreeOfTrees2
//Beginning with tree ((A,B),C) = ((AI,BI),CI)
//we generate all possible topologies of trees.
//The resultant data structure is that of a tree.
//It is shown here.

//Trees are generated
//and one can select one of them
//for visualization.

//Topology = the graph of the tree without lengths

//Formats = Link and Newick's notations.
//You can learn these notations by looking at drawings.



import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;
```

```
import java.awt.*;



import javax.swing.JComponent;


public class TreeOfTrees2 extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;



 //Everything is built by recusion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
 "ABCDEFGHIJKLMNOPRSTUVWXYZ";
 private static String internalNodes =
 "abcdefghijklmnopqrstuvwxyz";

 private static boolean print = false;


 //Number of leaves that must go in all trees
 private static int numberOfLeaves = 6;


 private static String[] Record =  new String[10000];
 //Last filled position  in Record[]
 private static int lastUsedPosition;
 private static String selectedTree = "";

  //DynamicTree is an inner service class that comes
  //at the end of this file.
```

```
  private static DynamicTree5 treePanel;
  private static String VISUALIZE_COMMAND = "Visualize";

  //Constructor of main class: GUI basics
  public TreeOfTrees2()
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructionsLeft = new JLabel(
      "Select  a tree and click the Visualize button.");

   instructionPanel.setFocusable(true);
   instructionPanel.add(instructionsLeft);
   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree5();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(1200, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
     //Selected tree must be unveiled
     selectedTree = treePanel.unveilTree();
     if (print)
      System.out.println("tree just generated"
                + selectedTree);
  //Translation from one format into another
```

```
      //output = a vector with binary address and contents
      translate(selectedTree);
      DrawTree y = new DrawTree();
      y.createAndShowGUI();
 }
  }

//======================

  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame
     = new JFrame("Visual generator of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees2 newContentPane = new TreeOfTrees2();
  // content panes must be opaque
  newContentPane.setOpaque(true);
```

```
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //add char c to output
   tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
```

```
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
```

```
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //Trees are generated
  private static void generateTrees()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }


//=============== Translation in within formats ============
```

```
//Translation from Newick's to quasi-binary notation
private static String toBinary(String tree)
{
 String regulators = "(,)";
 //Output
 String treeToBi = "";
 //The binary address of leaf
 String address = "";
 //The name or content of the tree
 String content = "";
 String thisChar = "";
 String nextChar = "";
 String previousChar = "";
 String prepreviousChar = "";
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if ( i < l-1) nextChar = "" + tree.charAt(i+1);
   else nextChar = "end";
   if ( i > 0) previousChar = "" + tree.charAt(i-1);
   else previousChar = "start";
   //Is thisChar a regulator?
   if (regulators.contains(thisChar))
 {
   treeToBi =  treeToBi + thisChar;

   //This char = (
   if (thisChar.contentEquals("("))
   {
address = address + '0';
if (print)
System.out.println("Address = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
 {
```

```
  content = ""+ nextChar;
  treeToBi =  treeToBi + address + content;

}
   }


 //This char = ,
   if (thisChar.contentEquals(","))
   {
if (print)
System.out.println("Address at ,   = " + address);
int m = address.length();
//replace last 0 by a 1
address = address.substring(0, m-1) + '1';
if (print)
  System.out.println("Address at , 1 = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
  {

 content = ""+ nextChar;

 treeToBi =  treeToBi + address + content;

  }
   }


 //This char = )
  if (thisChar.contentEquals(")"))
   {
  prepreviousChar = "" + tree.charAt(i-2);
if ( (internalNodes.contains(previousChar)
& (leaves.contains(prepreviousChar))))
{
 content = "" + tree.charAt(i-2);
```

```
}
int m = address.length();
//Go one level down the tree
    address = address.substring(0, m-1);
   }


 } //End of   if (regulators.contains(thisChar))
   }//End of for(int i...
 return treeToBi;
}


//bTree in complex Newick's notation
//is used to generate Record[]
//with address and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
```

```
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}

//Tree is translated
//from link notation to binary.
//Output in Record[]
public static void translate(String tree)
{
 System.out.println("Tree to be translated = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 makeInput(bTree);
 if (print)
 {
  System.out.println("\nTree in Newick's notation: \n"
         + tree);
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }
 if (print)
 System.out.println("lastUsedPosition = "
                     + lastUsedPosition);
}




public static void main(String[] args)
  {
```

```
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        createAndShowGUI();
        generateTrees();
      }
    });
  }




//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree5 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree5()
```

```
{
   super(new GridLayout(1, 0));

   rootNode = new DefaultMutableTreeNode(seed);
   treeModel = new DefaultTreeModel(rootNode);

   tree = new JTree(treeModel);
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);

   JScrollPane scrollPane = new JScrollPane(tree);
   add(scrollPane);
}


/** Selected tree is unveiled */
public String unveilTree()
{
  DefaultMutableTreeNode selectedNode = null;
  TreePath selectedPath = tree.getSelectionPath();

  if (selectedPath == null) {
   selectedNode = rootNode;
  } else {
   selectedNode = (DefaultMutableTreeNode)
           (selectedPath.getLastPathComponent());
  }

  String tree =  selectedNode.toString();
  System.out.println("\nSelected tree = "   + tree);
  return tree;
}


//Add a node
public DefaultMutableTreeNode
             addObject(DefaultMutableTreeNode parent,
    Object child, boolean shouldBeVisible) {
```

```
    DefaultMutableTreeNode childNode =
                new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
```

```
System.out.println("ParentTree = " +  parentTree );

//===============

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
              i,   nextLeaf,
nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
```

```
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                       + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e) {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));
```

```
      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e) {
    }

    public void treeNodesRemoved(TreeModelEvent e) {
    }

    public void treeStructureChanged(TreeModelEvent e) {
    }
  }
}//End of DynamicTree5


//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.

//Resultant trees are drawn.




class DrawTree extends JPanel
```

```
{
private static final long serialVersionUID = 1L;
private JPanel drawingPane22;
JScrollPane scroller;



//=====================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength;



  String  NwTree = "";



 boolean print = false;



//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
```

```
//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];



//Set up the instructions.
JLabel instructionsLeft;

public DrawTree()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (Newick's notation) = " +  simplify(selectedTree));

instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);

//Set up the drawing area.
drawingPane22 = new drawingPane22();
drawingPane22.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane22);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//========================================
//============= Painting class ============
```

```
//=========================================


/** The component inside the scroll pane. */
 class drawingPane22 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;




//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
```

```
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane22.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane22.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;
//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;
}



//Tree is drawn
private void drawTree(Graphics2D g2d)
{
//The tree is drawn:
```

```
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree=  "
                                + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
```

```
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not drawn: our tree is unrooted

yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
```

```
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
sizeOfTree(g2d);
drawTree(g2d);
drawStrings(g2d);



}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of inner class drawingPane22



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
```

```
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame
  = new JFrame("Tree (Link notation) = " + selectedTree);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawTree();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of main class 147 DrawTree

}//End of main class J148 TreeOfTrees2
```

149 ***Exercise.*** *Run the previous program and play with the code. Play enough to get certainty on the correctness or incorrectness of the code.*

We have developed a code to generate in Newick's notation all possible topologies without repetition and to correctly draw the selected one. So, we are ready to take the next step in our purpose of solving the problem of finding the best tree that fits a distance matrix. So, let us think of adding lengths to the links of our trees.

We face at once the following trouble: our depiction of trees was addressed to faithfully represent the Newick's notation in which leaves are seen but inner nodes are not. Now, we need to see inner nodes because we must assign lengths to inner links also.

150 ***Exercise.*** *Add to the previous program the visualization of inner nodes. You must solve the problem by hand over a suitable example and then make the corresponding implementation. Play enough with your new code to accept the needed conclusion: the code is correct.* *Answer*

151 ***Trees with lengths of links.***

Since we are done with the topology of trees, we are ready now to include the length of links as in notation as in drawings. This extension in the Newick's notation is as follows:

The length associated to a link is appended after a colon. Say, C:3 means that the link of C to the tree has length 3. Analogously, D:6 means that the length associated to the link of D is 6. This convention is also part of the Newick's notation for phylogenetic trees.

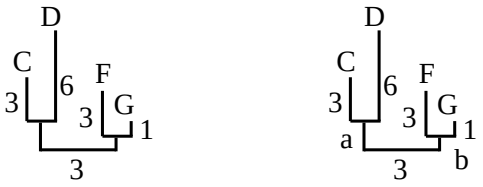


Figure 5.10. This unrooted tree to the left might be encoded in the Newick's notation variously, say, ((C: 3,D: 6):3,(F: 3,G:1)). The corresponding Link version to the right would be: ((Ca: 3,Da: 6)b:3,(Fb: 3,Gb:1))

The unrooted tree of this graphic might be encoded in various ways:

((C: 3,D: 6):3,(F: 3,G:1))
((C: 3,D: 6),(F: 3,G:1):3)
((C: 3,D: 6):2,(F: 3,G:1):1)
((C: 3,D: 6):1.5,(F: 3,G:1):1.5)

The corresponding Link versions would look like this:

((Ca: 3,Da:6)b:3,(Fb: 3,Gb:1))
((Ca: 3,Da:6),(Fb: 3,Gb:1)a:3)
((Ca: 3,Da:6)b:2,(Fb: 3,Gb:1):1)
((Ca: 3,Da:6)b:1.5,(Fb: 3,Gb:1)a:1.5)

152 ***The grammar to add lengths.***

To add lengths, we receive an input that denotes a tree in Link notation as the following:

(((Ad,Fd)a,Ba),((Cb,Db)c,Ec)a)

and we must output something like:

(((A:10d:2,F:5d:2)a:7,B:1a:7),((C:4b:3,D:9b:8)c:2,E:6c:2)a:7)

What shall we do? It seems that we are dealing with a grammar of the following type:

$A -> A{:}5$

where A represents a leaf or an inner node and 5 is an output of the random generator.

This seems pretty simple apart from a minor hindrance. Our trees must be translated to binary notation which includes 0's and 1's to denote binary addresses and we also find these numbers here to indicate lengths. This adds interpretation troubles to our notation. So, we will change the binary notation of addresses as follows: a 0 is replaced by +, the plus sign that means go up, and a 1 by -, the minus sign that means go down.

Let us update our Java code for trees with this new option that allows links to have a length. In that way, we will get closer to our problem of finding the tree that best fits a distance matrix.

153 ***The following program generates all possible trees up to certain number of leaves and allows the assignment of a length at random to each link of a selected tree.***

```
//Program J153 TreeOfTrees4
//Beginning with tree ((A,B),C) = ((AI,BI),CI)
//we generate all possible topologies of trees.
//The resultant data structure is that of a tree.
//It is shown here.


//Topologies are generated
//and one can select one of them
//for visualization.
```

```
//Inner nodes also depicted.

//The corresponding tree is endowed with
//links of different length according to assignment
//at random.

//Topology = the graph of the tree without lengths

//Formats = Link and Newick's notations.
//You can learn these notations by looking at drawings.


import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;
```

```
public class TreeOfTrees4 extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;



 //Everything is built by recursion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
 "ABCDEFGHIJKLMNOPRSTUVWXYZ";
 private static String internalNodes =
 "abcdefghijklmnopqrstuvwxyz";


 private static boolean print = false;
 private static boolean print2 = true;


 //Number of leaves that must go in all trees
 private static int numberOfLeaves = 6;


 //Maximal allowed length
 private static int maxLength = 10;


 private static String[] Record =  new String[10000];
 //Last filled position  in Record[]
 private static int lastUsedPosition;
 private static String selectedTree = "";

  //DynamicTree is an inner service class that comes
  //at the end of this file.
  private static DynamicTree7 treePanel;
```

```
  private static String VISUALIZE_COMMAND = "Visualize";

  private static Random r = new Random();

  //Constructor of main class: GUI basics
  public TreeOfTrees4()
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructionsLeft = new JLabel(
      "Select  a tree and click the Visualize button.");

   instructionPanel.setFocusable(true);
   instructionPanel.add(instructionsLeft);
   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree7();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(1200, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
     //Selected tree must be unveiled
     selectedTree = treePanel.unveilTree();
     System.out.println("\nSelected tree in Link notation"
      + " after unveilTree() = \n"   + selectedTree);
     selectedTree = addLenghts(selectedTree);
```

```
      System.out.println("\nSelected tree with random "
      + "lengths of Links = \n"   + selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      translate(selectedTree);
      DrawBiTree3 y = new DrawBiTree3();
      y.createAndShowGUI();
 }
  }

//======================


//Random lengths are added to a tree in Link notation
public static String addLenghts(String treeLinkN)
 {
  int l = leaves.length();
  int lengthsLeaves[]  = new int[l];
  //Last two entries are redundant
  int lengthsInnerNodes[]  = new int[l];
  //Lengths are generated
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextInt(maxLength);
 lengthsInnerNodes[i] =  r.nextInt(maxLength);
   }

  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
  //Tree is scanned char by char
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      Integer j = lengthsLeaves[k];
```

```
     newTree = newTree + ":" + j.toString();
     }
    if (internalNodes.contains(thisChar))
    {
     int k = internalNodes.indexOf(thisChar);
     Integer j = lengthsInnerNodes[k];
    newTree = newTree + ":" + j.toString();
    }
   }

   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Visual generator of tree topologies.");
```

```
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees4 newContentPane = new TreeOfTrees4();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //add char c to output
   tree = tree + c;
 }
return tree;
}
```

```
//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
```

```
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //Trees are generated
  private static void generateTrees()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }
```

```
//=============== Translation in within formats ============




//Translation from Newick's to quasi-binary notation
  //Inner nodes are lacked
private static String toBinary(String tree)
{
 String regulators = "(,)";
 //Output
 String treeToBi = "";
 //The binary address of leaf
 String address = "";
 //The name or content of the tree
 String content = "";
 String thisChar = "";
 String nextChar = "";
 String previousChar = "";
 String prepreviousChar = "";
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if ( i < l-1) nextChar = "" + tree.charAt(i+1);
   else nextChar = "end";
   if ( i > 0) previousChar = "" + tree.charAt(i-1);
   else previousChar = "start";
   //Is thisChar a regulator?
   if (regulators.contains(thisChar))
 {
   treeToBi =  treeToBi + thisChar;

   //This char = (
```

```
   if (thisChar.contentEquals("("))
   {
address = address + '+';
if (print)
System.out.println("Address = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
 {

  content = ""+ nextChar;
  treeToBi =  treeToBi + address + content;

}
   }


 //This char = ,
   if (thisChar.contentEquals(","))
   {
if (print)
System.out.println("Address at ,   = " + address);
int m = address.length();
//replace last 0 by a 1
address = address.substring(0, m-1) + '-';
if (print)
  System.out.println("Address at , 1 = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
  {

 content = ""+ nextChar;

 treeToBi =  treeToBi + address + content;

  }
   }


 //This char = )
  if (thisChar.contentEquals(")"))
```

```
   {
  prepreviousChar = "" + tree.charAt(i-2);
if ( (internalNodes.contains(previousChar)
& (leaves.contains(prepreviousChar))))
{
 content = "" + tree.charAt(i-2);


}
int m = address.length();
//Go one level down the tree
    address = address.substring(0, m-1);
   }


 } //End of   if (regulators.contains(thisChar))
   }//End of for(int i...
 return treeToBi;
}


//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
```

```
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}
```

```
//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
//Output
String treeToBi = "";
String numeric = ":0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
```

```
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
```

```
   if (address.contentEquals("")) address = "+";
   treeToBi =  treeToBi + address + thisChar;
   //if (m==1) treeToBi =   "0" + thisChar;
  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (numeric.contains(previousChar))
  address = address + '-';//Add a 1 to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree in LL (links with lengths) notation
```

```
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ":0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
```

```
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method



//Tree is translated
//from link notation to binary.
//Output in Record[]: even entires= addresses, odd = names
public static void translate(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
 if (print)
 System.out.println("Tree  at translate = "    + tree);
 //String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
 System.out.println("Selected tree in " +
  "pseudo-Newick's notation = " + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

//A vector with addresses and names is composed
makeInputInner(bTree);
```

```
 if (print)
 System.out.println("lastUsedPosition = "
                     + lastUsedPosition);
}



public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        createAndShowGUI();
        generateTrees();
      }
    });
  }




//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree7 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
```

```
//Next leaf to be grafted
char nextLeaf = leaves.charAt(3);
//Next internal node to be inserted
char nextIntNode = internalNodes.charAt(1);
//counter of generations
int counterGen = 0;


//Constructor
public DynamicTree7()
{
   super(new GridLayout(1, 0));

   rootNode = new DefaultMutableTreeNode(seed);
   treeModel = new DefaultTreeModel(rootNode);

   tree = new JTree(treeModel);
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);

   JScrollPane scrollPane = new JScrollPane(tree);
   add(scrollPane);
 }


 /** Selected tree is unveiled */
 public String unveilTree()
 {
   DefaultMutableTreeNode selectedNode = null;
   TreePath selectedPath = tree.getSelectionPath();

   if (selectedPath == null) {
    selectedNode = rootNode;
   } else {
    selectedNode = (DefaultMutableTreeNode)
            (selectedPath.getLastPathComponent());
   }
```

```
    String tree =  selectedNode.toString();
    if (print)
    System.out.println("\nSelected tree at unveilTree() = "
                         + tree);
    return tree;
  }


  //Add a node
  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
               new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;
```

```
   Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
System.out.println("ParentTree = " +  parentTree );

//===============

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
```

```
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                      + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());
```

```
      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }

    public void treeNodesRemoved(TreeModelEvent e)
    {
    }

    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree7


//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
```

```
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.

//Resultant trees are drawn.




class DrawBiTree3 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane24;
JScrollPane scroller;




//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength;
```

```
  String  NwTree = "";


 boolean print = false;


//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];



//Set up the instructions.
JLabel instructionsLeft;

public DrawBiTree3()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (pseudo-Newick's notation) = "
  + simplify(selectedTree));

instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);

//Set up the drawing area.
```

```
drawingPane24 = new drawingPane24();
drawingPane24.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane24);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
 class drawingPane24 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;
```

```
//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane24.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane24.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
```

```
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;
//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;
}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                                + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
```

```
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
```

```
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
```

```
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane24



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame =
   new JFrame("Tree (Link notation) = " + selectedTree);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree3();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree3
```

```
}//End of main class J153 TreeOfTrees4
```

154 ***Exercise.*** *Run the previous program and play with the code. Play enough to get confidence in accepting or rejecting the code.*

155 ***Exercise.*** *The previous program allows the assignment of lengths to links but with digit values, in 0,1,..,9. Extend the allowed repertoire to include real numbers say 123.456.* Answer

156 ***Challenge.*** *The lengths of links are indicated as strings in the trees drawn by previous programs but the lengths are not drawn accordingly. Make the necessary amendment. Also, remake the drawing machinery to depict trees in your preferred form, say triangular. Notice also that our drawings have too many void gaps. Squeeze them.*

We need one more detail to get closer to the matrix-tree evolutionary problem: to calculate the path through the tree from anyone leaf to another. To solve this problem, we calculate the path to a given internal node, say I. Then, to calculate the path from leaf A to B, we join the path from A to I to the path from I to B.

157 ***Exercise.*** *Add to the previous program the capability to output to the console the path from anyone leaf to the internal node a.* Answer

If we know how to go from leaf A to node Root, then we can concatenate that path to the reverse path from leaf B to node Root to get from A to B.

158 ***Distance matrix of a tree.*** *Let us calculate the distance matrix associated to a tree but restricted to leaves. See example 126, pag. 238. The code follows:*

```
//Program J158 TreeOfTrees7
//Beginning with tree ((A,B),C) = ((AI,BI),CI)
//we generate all possible topologies of trees.
//The resultant data structure is that of a tree.
//It is shown here.

//Topologies are generated
//and one can select one of them
//for visualization.
//Inner nodes are also depicted.
```

```
//The corresponding tree is endowed with
//links of different length according to assignment
//at random.
//Lengths can  take on real values such as 123.456

//The distance matrix associated to the selected tree
//but restricted to leaves
//is calculated and printed to the console.

//Topology = the graph of the tree without lengths

//Formats = Link and Newick's notations. And variants.
//You can learn these notations by looking at drawings.


import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;
```

```
import javax.swing.JComponent;


public class TreeOfTrees7 extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;



 //Everything is built by recursion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
 "ABCDEFGHIJKLMNOPRSTUVWXYZ";
 private static String internalNodes =
 "abcdefghijklmnopqrstuvwxyz";


 private static boolean print = false;
 private static boolean print2 = true;


 //Number of leaves that must go in all trees
 private static int numberOfLeaves = 6;




 private static String[] Record =  new String[10000];
 //Last filled position  in Record[]
 private static int lastUsedPosition;
 private static String selectedTree = "";


  //DynamicTree is an inner service class that comes
```

```
//at the end of this file.
private static DynamicTree10 treePanel;
private static String VISUALIZE_COMMAND = "Visualize";

 //Length of the link that glues a leaf to the tree
private static int lengthsLeaves[]  =
              new int[numberOfLeaves];
//Length of the link that glues
//an inner node to another one
private static int lengthsInnerNodes[]  =
             new int[numberOfLeaves];


private static Random r = new Random();
private static int maxLength = 10;

//Paths from each leaf to node a
private static String[] Paths =
             new String[numberOfLeaves];

private static double distanceM[][] =
   new double[numberOfLeaves][numberOfLeaves];
//Constructor of main class: GUI basics
public TreeOfTrees7()
{
 super(new BorderLayout());
 JPanel instructionPanel = new JPanel(new GridLayout(0,1));
 JLabel instructionsLeft = new JLabel(
    "Select  a tree and click the Visualize button.");

 instructionPanel.setFocusable(true);
 instructionPanel.add(instructionsLeft);
 add(instructionPanel, BorderLayout.PAGE_END);

 //A new node is added
 JButton visualizeButton = new JButton("Visualize");
 visualizeButton.setActionCommand(VISUALIZE_COMMAND);
 visualizeButton.addActionListener(this);

 // Create the components.
```

```
   treePanel = new DynamicTree10();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(700, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
      //Selected tree must be unveiled
      selectedTree = treePanel.unveilTree();
      System.out.println("\nSelected tree in Link notation"
       + " after unveilTree() = \n"   + selectedTree);
      //Lengths are added
      selectedTree = addLenghts(selectedTree);
      System.out.println("\nSelected tree in " +
       "Link notation with random "
      + "lengths of Links = \n"   + selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      translate(selectedTree);
      DrawBiTree6 y = new DrawBiTree6();
      y.createAndShowGUI();
 }
  }

//======================


//Random lengths are added to a tree in Link notation
public static String addLenghts(String treeLinkN)
 {
  //int l = leaves.length();

  //Lengths with digit values are generated
```

```
 for(int i = 0; i < numberOfLeaves; i++)
  {
lengthsLeaves[i] = r.nextInt(maxLength);
lengthsInnerNodes[i] =  r.nextInt(maxLength);
  }


 /*
 //Lengths with real values are generated
 double lengthsLeaves[]  = new double[l];
 //Last two entries are redundant
 double lengthsInnerNodes[]  = new double[l];
 for(int i = 0; i < numberOfLeaves; i++)
  {
lengthsLeaves[i] = r.nextDouble();
lengthsInnerNodes[i] =  r.nextDouble();
  }
 */

 String thisChar = "";
 String newTree = "";
 //length of String   tree
 int m = treeLinkN.length();
 //Tree is scanned char by char
 for(int i = 0; i < m; i++ )
  {
   thisChar = "" + treeLinkN.charAt(i);
   newTree = newTree + thisChar;
   if (print)
   System.out.println(newTree.length() + " " + newTree);
   //Lengths are appended
   if (leaves.contains(thisChar))
    {
     int k = leaves.indexOf(thisChar);
     double j = lengthsLeaves[k];
    newTree = newTree + ":" + j;
    }
   if (internalNodes.contains(thisChar))
   {
```

```
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerNodes[k];
     newTree = newTree + ":" + j;
    }
   }

   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Visual generator of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees7 newContentPane = new TreeOfTrees7();
  // content panes must be opaque
```

```
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
```

```
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
```

```
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //Trees are generated
  private static void generateTrees()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }
```

```
//=============== Translation in within formats ============



//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
```

```
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Outputs a word in reversed order.
private static String reverser(String word)
{
 return new StringBuilder(word).reverse().toString();
}


private static String cleanTree(String tree)
{
 System.out.println("Input tree at deleteNumerics = \n "
        + tree);
 String treeClean = "";
 String numeric = ".:0123456789";
 String plusMinus = "+-";
 String thisChar = "";
  boolean leaf = false;
 //Numerics are washed
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if (numeric.contains(thisChar));
    else treeClean = treeClean + thisChar;
  }
 System.out.println("Tree without numerics = \n"
+ treeClean);
 //+/- are deleted from inner nodes that accompany leaves
 l = treeClean.length();
 String treeOut = "";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
```

```
 {
  thisChar = "" + treeClean.charAt(i);
  if (leaves.contains(thisChar))
  leaf  = true;
  if ((leaf) &  (plusMinus.contains(thisChar)));
else treeOut = treeOut + thisChar;
  if (internalNodes.contains(thisChar))
  leaf  = false;
}


 System.out.println("Tree at output of cleanTree = \n"
+ treeOut);
return treeOut;

}



//The path from each leaf to node a is found.
//The input contains branch's indicator, say:
//((++Aa,+-Ba),(-+Cb,((--++Dd,--+-Fd)--c,---Ec)-b)+a)
//Output for leaf A: Aa+.
//Output for leaf C: Cba-
//Output for leaf D: Ddcba-
private static void findPaths(String tree)
{
 System.out.println("Input tree at findPaths = \n" + tree);
 //length of String   tree
 int l = tree.length();
 String plusMinus = "+-";
 String thisChar;
 String nextChar;
 String path = "";
 //Branch indicative with respect to root, + else -
 String branch;
 String address ;
 //The number of links of a path to root
 int level = 0;
```

```
 String treeReversed = reverser(tree);
 System.out.println("Reversed tree = " + treeReversed);

//Each leaf has a path to a
for(int k= 0; k < numberOfLeaves; k++)
{
 String leaf = "" + leaves.charAt(k);
 //Candidate nodes to be part of the path to node a
 //must have  sign +/- as prefix.
 //Or as suffix in the reversed tree.
 int position = treeReversed.indexOf(leaf);
 if (position > -1) // if leaf is in tree
 {
  System.out.println("\nLeaf = " + leaf );
  address = "";
  boolean go = true;
  //branchIndicator and level are found
  for(int i = position+1 ; ((i < l) & (go)); i++ )
  {
   thisChar = "" + treeReversed.charAt(i);
   if   (plusMinus.contains(thisChar)  )
address = thisChar + address;
   else go = false;
  }
  level = address.length();
  branch = "" + address.charAt(0);
  if (print)
  System.out.println("Address  = " + leaf + address + level);



  //Path from leaf to node a is found in (not reversed) tree
  position = tree.indexOf(leaf);
  thisChar = "" + tree.charAt(position);
  nextChar = "" + tree.charAt(position + 1);
  //Output
  path = "" + leaf + nextChar;
  go = true;
  if  (nextChar.contentEquals("a") ) //leaf is neighbor of a
   {
```

```
  path = path + branch;
  if (print)
  System.out.println("Path short = " + path);
   }
  else  //level > 1
  {



  //Tree is scanned char by char to get path
  String iChar = "";
  for(int i = position + 2; ((i < l) & (go)); i++ )
  {
   iChar = "" + tree.charAt(i);


   int newLevel = 0;
   if (internalNodes.contains(iChar))//InternalNodes
{
     if   (iChar.contentEquals("a")  )
      {
   path  = path +  iChar;
   go = false; //end
      }
     else //Internal node different than a
     {
      if (print)
      System.out.println("Internal node not a) = " + iChar );
      int rP = l -i-1; //index of iChar in treeReversed
      String  nextNewChar = "" + treeReversed.charAt(rP+1);

      if ( plusMinus.contains(nextNewChar )) //candidate node
       {
       if (print)
     System.out.println("next candidate = " + iChar);
        String addressNew = "";

        //level of candidate node iChar  is inquired
        String thatChar = "";
        boolean go2 = true;
```

```
        for(int j = rP+1 ; ((j < l) & (go2)); j++ )
         {
       thatChar = "" + treeReversed.charAt(j);
       if   (plusMinus.contains(thatChar)  )
          addressNew = thatChar + addressNew;
       else go2 = false;
         }
        if (print)
        System.out.println("AddressNew " + iChar + addressNew);
       newLevel = addressNew.length();
       //A path links a node with another in a lower level
       if  ( (newLevel < level) &
        (!(nextChar.contentEquals(iChar))))
        {
       path = path + iChar;
       level = newLevel;
       if (print)
       System.out.println("Path newLevel= " + path);
        }
      } //End of if candidate node
     }//End of else Internal node different than a
}//End of if InternalNodes
   }//End of Tree is scanned char by char

  path = path + branch;

 }//End of  else level > 1
 }//End of if leaf is in tree
 System.out.println("Path completed  = " + path);
 Paths[k] = path;
}//End of each leaf
}



//Paths from leaves to node a are found
//Reverse paths are also calculated.
//Input tree in quasibinary notation with addresses and lengths
private static void pathFinder(String tree)
{
```

```
 //Useless information is deleted
 tree = cleanTree(tree);
 //Find the path to node a from each leaf.
 //Add an indicator of the branch
 //with respect to the root: up(+) else down(-)
  findPaths(tree);
}

//Input: a long raw string with path from leaf to node a
//plus path from node a to leaf.
//Editing is done to get true paths.
private static String grammar (String path)
{
 int n = path.length();
 String thisChar = "";
 String nextChar = "";
 String oldPath = path;
 String plusMinus = "a+-a";
 String minusPlus = "a-+a";
 String two = "";
 int N = path.length();
//Rule 1: two equal symbols produce nothing
 for(int j=1; j < N; j++)
  {
   for(int i=1; i < n-1;)
   {
    thisChar = "" + path.charAt(i);
    nextChar = "" + path.charAt(i+1);
    two = "" + thisChar + nextChar;
    if (thisChar.contentEquals(nextChar))
  {
   i = i + 2;
   path = path.replace(two, "");
  }
    else
  {
   i = i+1;
  }
  n = path.length();
 }
```

```
}

 //Rule 2: a+-a -> a

 if  (path.contains(plusMinus))
      path = path.replace(plusMinus, "a");
 if   (path.contains(minusPlus))
     path = path.replace(minusPlus, "a");

 System.out.println("input = " + oldPath +
                     " output = " + path);
 return path;
}

//The distance matrix among leaves is calculated
private static void matrixBuilder()
{
 //Path to node a from leaf
 String pathLeafToa = "";
 //Path to leaf from node a
 String pathaToLeaf = "";
 //Path from leaf to lead
 String pathLeafToLeaf = "";
 String innerNode = "";
 double distance;
 for(int i = 0; i < numberOfLeaves; i++)
  for(int j = 0; j < numberOfLeaves; j++)
  {
   pathLeafToa = Paths[i];
   pathaToLeaf = reverser( Paths[j] );
   //Raw paths
   pathLeafToLeaf = pathLeafToa + pathaToLeaf;
   //An editing grammar to get true paths
   pathLeafToLeaf = grammar( pathLeafToLeaf );
   if (i == j) distanceM[i][j] = 0;
   else
    {
 String leaf = "" + pathLeafToLeaf.charAt(0);
 int p = leaves.indexOf(leaf);
 distance = lengthsLeaves[p];
```

```
 int n = pathLeafToLeaf.length();
 for(int k = 1; k < n-1; k++)
 {
  innerNode =  "" + pathLeafToLeaf.charAt(k);
  p = internalNodes.indexOf(innerNode);
  distance = distance + lengthsInnerNodes[p];
 }
 leaf =  "" + pathLeafToLeaf.charAt(n-1);
 p = leaves.indexOf(leaf);
 distance = distance + lengthsLeaves[p];
 distanceM[i][j] = distance;
    }
  }
}



//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
System.out.println("Input tree at toQuasiBinaryInner " +
                        "= \n" + tree);
//Output
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
```

```
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "+";
   treeToBi =  treeToBi + address + thisChar;

  }
```

```
 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (numeric.contains(previousChar))
  address = address + '-';//Add a - to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree in LL (links with lengths) notation
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
```

```
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
```

```
}//End of method


//Prints  a matrix and its title
private static void printMatrix(double Matrix[][],
                        String title)
{
System.out.println( "\n" + title );
int m = Matrix.length;
System.out.print(  "\t");
for ( int j = 0; j < m; j++ )
  System.out.print( leaves.charAt(j)+ "\t");
System.out.println();
for ( int i = 0; i < m; i++ )
 {
  System.out.print(leaves.charAt(i ) + "\t");
  for ( int j = 0; j < m; j++ )
 System.out.print( Matrix[ i ][j]+ "\t");
  System.out.println();
 }
}


//Tree is translated
//from link notation to binary.
//Output in Record[]: even entries= addresses, odd = names
public static void translate(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
 if (print)
 System.out.println("Tree  at translate = "    + tree);
 //String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);
```

```
 if (print2)
 {
 System.out.println("Selected tree in " +
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

 //A vector with addresses and names is composed
 makeInputInner(bTree);
 //Paths from leaves to node a are found.
 //Reverse paths are also listed.
 pathFinder(bTree);
 //The distance matrix among leaves is calculated
 matrixBuilder();
 printMatrix(distanceM, "Distance Matrix");

}



public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        createAndShowGUI();
        generateTrees();
      }
    });
  }
```

```
//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree10 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree10()
 {
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode(seed);
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);

    JScrollPane scrollPane = new JScrollPane(tree);
    add(scrollPane);
  }
```

```
/** Selected tree is unveiled */
public String unveilTree()
{
  DefaultMutableTreeNode selectedNode = null;
  TreePath selectedPath = tree.getSelectionPath();

  if (selectedPath == null) {
   selectedNode = rootNode;
  } else {
   selectedNode = (DefaultMutableTreeNode)
           (selectedPath.getLastPathComponent());
  }

  String tree =  selectedNode.toString();
  if (print)
  System.out.println("\nSelected tree at unveilTree() = "
                       + tree);
  return tree;
}


//Add a node
public DefaultMutableTreeNode
             addObject(DefaultMutableTreeNode parent,
    Object child, boolean shouldBeVisible) {
  DefaultMutableTreeNode childNode =
              new DefaultMutableTreeNode(child);

  if (parent == null) {
    parent = rootNode;
  }

  // It is key to invoke this on the TreeModel,
  //and NOT DefaultMutableTreeNode
  treeModel.insertNodeInto
            (childNode, parent, parent.getChildCount());

  // Make sure the user can see the lovely new node.
  if (shouldBeVisible) {
```

```
      tree.scrollPathToVisible
                     (new TreePath(childNode.getPath()));
    }
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
System.out.println("ParentTree = " +  parentTree );

//=================

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
```

```
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                      + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())
```

```
 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }

    public void treeNodesRemoved(TreeModelEvent e)
    {
    }
```

```
    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree10



//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.

//Resultant trees are drawn.




class DrawBiTree6 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane27;
JScrollPane scroller;




//=======================================
```

```
//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;



  String  NwTree = "";



 boolean print = false;



//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];



//Set up the instructions.
JLabel instructionsLeft;

public DrawBiTree6()
```

```
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (pseudo-Newick's notation) = "
  + simplify(selectedTree));

instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);

//Set up the drawing area.
drawingPane27 = new drawingPane27();
drawingPane27.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane27);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//========================================
//============= Painting class ===========
//========================================


/** The component inside the scroll pane. */
 class drawingPane27 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
```

```
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
```

```
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane27.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane27.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                                + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
```

```
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}
```

```
//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}
```

```
//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane27



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
```

```
JFrame frame =
   new JFrame("Tree (Link notation) = "
                  + simplify(selectedTree));
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree6();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree6

}//End of main class J158 TreeOfTrees7
```

159 ***Exercise.*** *Run the previous program and play with the code. This is a critical program: do your best to test it to comfortably agree else disagree with the Author: the code is correct.*

5.3 Our solution

We are at last ready to face the problem of extracting the maximum degree of treehood contained in a distance matrix.

160 ***A challenge for you***

In official terms our problem reads:

Given a target matrix D_L of n-gram distances for a set L, find the tree T with set of leaves L and with associated distance matrix D_T such that that $d(D_T, D_L)$, the absolute distance between matrix D_T and D_L, is a minimum.

So, we have an optimization problem. But we have not an anonymous problem. Quite to the contrary, we have a problem about trees that incarnate the very essence

of the Evolutionary Theory. Therefore and by our honor this problem must be solved by evolution. But evolution is real. This means that a price must be paid in terms of bugs made along the development of the program and of employed resources at the time of running it, say, memory and time, plus the work of testing the code for correctness. Plus trying to optimize parameters for fast performance. Besides, we already know that the number of different tree topologies is prey of a combinatorial explosion in the number of leaves.

In concrete terms we are saying that we must close our eyes to reality and work as if we had no constraint at all. Nevertheless, the most astringent resource is that of our own intelligence that is very good to solve very simple problems but very bad to deal with huge problems. That is why we go step by step. So, we will use evolution just to finely tune lengths but not to find the best fitting topology: we will scan them all to filter that with best performance.

161 *Our attack*

We have everything needed to propose an exact solution for the case in which the number of leaves is around 6. We will run an immediate test to assess the quality of our solution: we must end with the same tree if we go forth from a tree to a distance matrix and back to the best fitting tree.

162 ***The code for 6 leaves.*** *The following code fabricates a tree at random T, calculates the associated distance matrix D_T and finds the tree T_{D_T} that best fit D_T. In that way, we can test the requirement that T must be the best fitting to D_T or that $T_{D_T} = T$. We work ordinarily with less than 8 species. To find the tree that best fit D, we use the* ***tree of trees*** *to gauge the performance of all possible tree topologies, each one of which is submitted to evolution to finely tune lengths. For sake of ease, lengths take on digital values.*

```
//Program J162 EvoTree
/*
 * We generate a random tree topology
 * and attach random integer lengths to links.
 * In that way we get the target tree, T.
 * Next, we find the associated distance matrix D  to T.
 * Our problem is:
 *
 * Find tree t such that the absolute distance between
```

```
 * matrix D and the associated matrix to t is a minimum.
 * Absolute distance between M and N = sum |Mij-Nij|.
 *
 * We run the next test:
 * If we begin with a tree T, compute D, and then t,
 * we expect to return with the initial tree: t = T.
 *
 * To associate a distance matrix to a tree,
 * we calculate the distance in within
 * each pair of nodes (leaves + inner nodes).
 * To that aim, we begin with the first node and
 * walk over the tree until the second node
 * and add the lengths of all traversed links.
 * The resulting number is the  distance.
 *
 * To find the tree that best fit D,
 * we  scan all possible topologies,
 * each one of which is submitted to evolution
 * to finely tune lengths.
 *
 * Formats = Link and Newick's notations. And variants.
 *
 */



import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
```

```
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class EvoTree extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;




 //Everything is built by recursion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 private static String internalNodes =
 "abcdefghijklmnopqrstuvwxyz";

 //Next leaf to be grafted
 private static  int nextLeafIndex = 2;
 //Next internal node to be inserted
 private static  int nextIntNodeIndex = 0;


 static int maxNumberOfLeaves = leaves.length();
```

```
 //Number of leaves that must go in all trees > 3
 private static int numberOfLeaves = 4;

 private static String[] Record =  new String[10000];
 //Last filled position  in Record[]
 private static int lastUsedPosition;
 private static String selectedTree = "";


  //DynamicTree is an inner service class that comes
  //at the end of this file.
  private static DynamicTree11 treePanel;
  private static String VISUALIZE_COMMAND = "Visualize";

   //Length of the link that glues a leaf to the tree
  private static int lengthsLeaves[]  =
                new int[numberOfLeaves];
  //Length of the link that glues
  //an inner node to another one
  private static int lengthsInnerLinks[]  =
               new int[numberOfLeaves];


  private static Random r = new Random();
  private static int maxLength = 10;

  //Paths from each leaf to node a
  private static String[] Paths =
               new String[numberOfLeaves];

  private static String targetTree;
  private static double targetMatrix[][] =
     new double[numberOfLeaves][numberOfLeaves];


  //=======Evolution parameters============


//Population to evolve =
//lengths of links that attach leaves to tree
```

```
// + lengths of inner links.
//N individuals at all.

static private int N = 10;
static private double oldLengths[][][] =
 new double[N][2][maxNumberOfLeaves];

static private double newLengths[][][] =
new double[N][2][maxNumberOfLeaves];

static private double distance[] = new double[N];
static private double fitness[] = new double[N];

static private int numberOfGenerations = 2;

//Mutation rate per site (= entry) per generation
static private double    m = 0.5;

static private int best;
//The best tree for a given topology
static private String bestTree = "";

//=========Debugging help ===============

private static boolean print  = false;
private static boolean print2 = false;
private static boolean print3 = false;
private static boolean print4 = false;
//Turn on print5 firstly
private static boolean print5 = false;

  //=======Constructor of main class: GUI basics=====
  public EvoTree()
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructionsLeft = new JLabel(
      "Select  a tree and click the Visualize button.");

   instructionPanel.setFocusable(true);
```

```
   instructionPanel.add(instructionsLeft);
   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree11();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(700, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
      //Selected tree must be unveiled
      selectedTree = treePanel.unveilTree();
      System.out.println("\nSelected tree in Link notation"
       + " after unveilTree() = \n"   + selectedTree);
      makeLengths();
      //Lengths are added
      selectedTree = insertLengths(selectedTree);
      System.out.println("\nSelected tree in " +
       "Link notation with random "
      + "lengths of Links = \n"   + selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      associatedMatrix(selectedTree);
      DrawBiTree7 y = new DrawBiTree7();
      y.createAndShowGUI();
 }
  }
```

```
//======================



public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
     String tree = randomTopology();
     makeLengths();
     targetTree = insertLengths(tree);
     System.out.println("Target tree: each link " +
     "has its own length = \n" + targetTree);
     targetMatrix = associatedMatrix(targetTree);
        createAndShowGUI();
        System.out.println("\nDistance to target = " +
          " Absolute distance between" +
         " target matrix  and \n" +
         "the distance matrix associated to " +
         "the tree with lengths of links that \n" +
         "is the best fitting  of the topology.\n" +
         "Perfect fitting: distance = 0 \n");
        System.out.println("\nNumber of Generations = "
         + numberOfGenerations);
        //To find the best fitting, all tree topologies are
        //scanned and each one of them is evolved for
        //the best particular fitting.
        //All this is done along the process of
        //generating all topologies
        generateTopologies();
      }
    });
  }
```

```
//From parentTree, a tree in Newick's notation,
//a Child  tree is generated by a generative grammar.
//A random election is made among all possibilities.
private static String generateChild(String parentTree)
{

 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);


 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = parentTree.length();
 //counter of links = 0;
 int counter = 0;
 //Links are counted
 for(int i = 0; i < l; i++ )
  {
   //parentTree is scanned char by char
   char I = parentTree.charAt(i);
 //Is I an internal node that denotes a link?
 if (charIsInternalNode(I)) counter++;
  }
 //A link is chosen
 int k = r.nextInt(counter);
 if (print2)
 System.out.println("Random number = " + k);
 //new branch is grafted at k
 counter = 0;
 for(int i = 0; i < l; i++ )
```

```
   {
     //parentTree is scanned char by char
  char I = parentTree.charAt(i);
  //Is I an internal node that denotes a replacement?
  if (charIsInternalNode(I))
   {
    if (counter == k) //Is I the chosen link?
     {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree, i,
                 nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
    //Link to be replaced
    link = "" + a + I;
    //Replacement rule 1:
    //aI -> (aN,FN)I;   N= nextIntNode;
    //F = nextLeaf
   replacement = "(" + a + nextIntNode + ","
            + nextLeaf + nextIntNode + ")" + I;
   //Rule 1 is executed
   newTree = parentTree.replace(link, replacement);
   }
  if (print2)
  System.out.println(newTree + " -> "
                        + simplify(newTree));

   i = l;
     }
    counter++;
  }
   }
  return newTree;
}


//A tree topology is generated at random.
public static String randomTopology()
```

```
 {
  if (print2)
  System.out.println("Parent tree = \n" + seed
   + " -> "    + simplify(seed));
  String tree = seed;
  int limit = numberOfLeaves-3; //less than 11.
  for(int i = 0; i < limit; i++ )
  tree = generateChild(tree);
  System.out.println("Target topology = \n" + tree);
  return tree;
 }


//======================


//Random lengths are added to a tree in Link notation
public static void makeLengths()
 {
  //Lengths with digit values are generated
  for(int i = 0; i < numberOfLeaves; i++)
   {
 //Zero lengths are not allowed
 lengthsLeaves[i] = 1 + r.nextInt(maxLength - 1);
 lengthsInnerLinks[i] = 1 +  r.nextInt(maxLength - 1);
   }
 }

  /*
  //Lengths with real values are generated
  double lengthsLeaves[]  = new double[l];
  //Last two entries are redundant
  double lengthsInnerLinks[]  = new double[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextDouble();
 lengthsInnerLinks[i] =  r.nextDouble();
   }
  */
```

```
//Random lengths are added to a tree in Link notation
public static String insertLengths(String treeLinkN)
 {
  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
  //Tree is scanned char by char
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    if (print)
    System.out.println(newTree.length() + " " + newTree);
    //Lengths are appended
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      double j = lengthsLeaves[k];
     newTree = newTree + ":" + j;
     }
    if (internalNodes.contains(thisChar))
    {
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerLinks[k];
     newTree = newTree + ":" + j;
    }
   }
   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
```

```
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Visual generator of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  EvoTree newContentPane = new EvoTree();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
```

```
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
String numeric = ".:012356789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
```

```
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
```

```
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //Trees are generated
  private static void generateTopologies()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }



//========= Translation in within formats ==========



//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput ////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
```

```
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Outputs a word in reversed order.
private static String reverser(String word)
{
 return new StringBuilder(word).reverse().toString();
}


private static String cleanTree(String tree)
{
 if (print2)
 System.out.println("Input tree at deleteNumerics = \n "
```

```
        + tree);
 String treeClean = "";
 String numeric = ".:0123456789";
 String plusMinus = "+-";
 String thisChar = "";
  boolean leaf = false;
 //Numerics are washed
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if (numeric.contains(thisChar));
    else treeClean = treeClean + thisChar;
  }
 if (print2)
 System.out.println("Tree without numerics = \n"
+ treeClean);
 //+/- are deleted from inner nodes that accompany leaves
 l = treeClean.length();
 String treeOut = "";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
 {
  thisChar = "" + treeClean.charAt(i);
  if (leaves.contains(thisChar))
  leaf  = true;
  if ((leaf) &  (plusMinus.contains(thisChar)));
else treeOut = treeOut + thisChar;
  if (internalNodes.contains(thisChar))
  leaf  = false;
}

 if (print2)
 System.out.println("Tree at output of cleanTree = \n"
+ treeOut);
return treeOut;

}
```

```
//The path from each leaf to node a is found.
//The input contains branch's indicator, say:
//((++Aa,+-Ba),(-+Cb,((--++Dd,--+-Fd)--c,---Ec)-b)+a)
//Output for leaf A: Aa+.
//Output for leaf C: Cba-
//Output for leaf D: Ddcba-
private static void findPaths(String tree)
{
 if (print2)
 System.out.println("Input tree at findPaths = \n" + tree);
 //length of String   tree
 int l = tree.length();
 String plusMinus = "+-";
 String thisChar;
 String nextChar;
 String path = "";
 //Branch indicative with respect to root, + else -
 String branch;
 String address ;
 //The number of links of a path to root
 int level = 0;
 String treeReversed = reverser(tree);
 if (print2)
 System.out.println("Reversed tree = " + treeReversed);

//Each leaf has a path to a
for(int k= 0; k < numberOfLeaves; k++)
{
 String leaf = "" + leaves.charAt(k);
 //Candidate nodes to be part of the path to node a
 //must have  sign +/- as prefix.
 //Or as suffix in the reversed tree.
 int position = treeReversed.indexOf(leaf);
 if (position > -1) // if leaf is in tree
 {
  if (print2)
```

```
  System.out.println("\nLeaf = " + leaf );
  address = "";
  boolean go = true;
  //branchIndicator and level are found
  for(int i = position+1 ; ((i < l) & (go)); i++ )
  {
   thisChar = "" + treeReversed.charAt(i);
   if   (plusMinus.contains(thisChar)  )
address = thisChar + address;
   else go = false;
  }
  level = address.length();
  if (print2)
  System.out.println("LeafAddressLevel  = " + leaf
                     + address + level);
  branch = "" + address.charAt(0);


  //Path from leaf to node a is found in (not reversed) tree
  position = tree.indexOf(leaf);
  thisChar = "" + tree.charAt(position);
  nextChar = "" + tree.charAt(position + 1);
  //Output
  path = "" + leaf + nextChar;
  go = true;
  if  (nextChar.contentEquals("a") ) //leaf is neighbor of a
   {
  path = path + branch;
  if (print)
  System.out.println("Path short = " + path);
   }
  else  //level > 1
  {

  //Tree is scanned char by char to get path
  String iChar = "";
  for(int i = position + 2; ((i < l) & (go)); i++ )
  {
   iChar = "" + tree.charAt(i);
```

```
   int newLevel = 0;
   if (internalNodes.contains(iChar))//InternalNodes
{
     if   (iChar.contentEquals("a")  )
      {
   path  = path +  iChar;
   go = false; //end
      }
     else //Internal node different than a
     {
      if (print)
      System.out.println("Internal node not a) = " + iChar );
      int rP = l -i-1; //index of iChar in treeReversed
      String  nextNewChar = "" + treeReversed.charAt(rP+1);

      if ( plusMinus.contains(nextNewChar )) //candidate node
       {
       if (print)
     System.out.println("next candidate = " + iChar);
        String addressNew = "";

        //level of candidate node iChar  is inquired
        String thatChar = "";
        boolean go2 = true;
        for(int j = rP+1 ; ((j < l) & (go2)); j++ )
         {
       thatChar = "" + treeReversed.charAt(j);
       if   (plusMinus.contains(thatChar)  )
          addressNew = thatChar + addressNew;
       else go2 = false;
         }
        if (print)
        System.out.println("AddressNew " +
                                    iChar + addressNew);
       newLevel = addressNew.length();
       //A path links a node with another in a lower level
       if  ( (newLevel < level) &
        (!(nextChar.contentEquals(iChar))))
        {
```

```
        path = path + iChar;
        level = newLevel;
        if (print)
        System.out.println("Path newLevel= " + path);
         }
       } //End of if candidate node
      }//End of else Internal node different than a
}//End of if InternalNodes
   }//End of Tree is scanned char by char

  path = path + branch;

 }//End of  else level > 1
 }//End of if leaf is in tree
 if (print2)
 System.out.println("Path completed  = " + path);
 Paths[k] = path;
}//End of each leaf
}



//Paths from leaves to node a are found
//Reverse paths are also calculated.
//Input tree in quasibinary notation
//with addresses and lengths
private static void pathFinder(String tree)
{
 //Useless information is deleted
 tree = cleanTree(tree);
 //Find the path to node a from each leaf.
 //Add an indicator of the branch
 //with respect to the root: up(+) else down(-)
  findPaths(tree);
}

//Input: a long raw string with path from leaf to node a
//plus path from node a to leaf.
//Editing is done to get true paths.
private static String grammar (String path)
```

```
{
 int n = path.length();
 String thisChar = "";
 String nextChar = "";
 String oldPath = path;
 String plusMinus = "a+-a";
 String minusPlus = "a-+a";
 String two = "";
 int N = path.length();
//Rule 1: two equal symbols produce nothing
 for(int j=1; j < N; j++)
  {
   for(int i=1; i < n-1;)
   {
    thisChar = "" + path.charAt(i);
    nextChar = "" + path.charAt(i+1);
    two = "" + thisChar + nextChar;
    if (thisChar.contentEquals(nextChar))
  {
   i = i + 2;
   path = path.replace(two, "");
  }
    else
  {
   i = i+1;
  }
  n = path.length();
 }
}

 //Rule 2: a+-a -> a

 if  (path.contains(plusMinus))
      path = path.replace(plusMinus, "a");
 if   (path.contains(minusPlus))
     path = path.replace(minusPlus, "a");
 if (print2)
 System.out.println("input = " + oldPath +
                    " output = " + path);
 return path;
```

```
}

//The distance matrix among leaves is calculated
private static double[][] matrixBuilder()
{
double M[][] =
new double[numberOfLeaves][numberOfLeaves];
 //Path to node a from leaf
 String pathLeafToa = "";
 //Path to leaf from node a
 String pathaToLeaf = "";
 //Path from leaf to lead
 String pathLeafiToLeafj = "";
 String innerNode = "";
 double distance;
 for(int i = 0; i < numberOfLeaves; i++)
  for(int j = 0; j < numberOfLeaves; j++)
  {
   pathLeafToa = Paths[i];
   pathaToLeaf = reverser( Paths[j] );
   //Raw paths
   pathLeafiToLeafj = pathLeafToa + pathaToLeaf;
   //An editing grammar to get true paths
   pathLeafiToLeafj = grammar( pathLeafiToLeafj );
   if (i == j) M[i][j] = 0;
   else
    {
 String leaf = "" + pathLeafiToLeafj.charAt(0);
 int p = leaves.indexOf(leaf);
 distance = lengthsLeaves[p];
 int n = pathLeafiToLeafj.length();
 for(int k = 1; k < n-1; k++)
 {
  innerNode =  "" + pathLeafiToLeafj.charAt(k);
  p = internalNodes.indexOf(innerNode);
  distance = distance + lengthsInnerLinks[p];
 }
 leaf =  "" + pathLeafiToLeafj.charAt(n-1);
 p = leaves.indexOf(leaf);
 distance = distance + lengthsLeaves[p];
```

```
  M[i][j] = distance;
    }
  }
 return  M;
}



//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
if (print2)
System.out.println("Input tree at toQuasiBinaryInner " +
                         "= \n" + tree);
//Output
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
```

```
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "+";
   treeToBi =  treeToBi + address + thisChar;

  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);
```

```
   if    (numeric.contains(previousChar))
  address = address + '-';//Add a - to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree  (in links with lengths notation)
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
```

```
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method


//Prints  a matrix and its title
private static void printMatrix(double Matrix[][],
                        String title)
{
System.out.println( "\n" + title );
```

```
int m = Matrix.length;
System.out.print(  "\t");
for ( int j = 0; j < m; j++ )
  System.out.print( leaves.charAt(j)+ "\t");
System.out.println();
for ( int i = 0; i < m; i++ )
 {
  System.out.print(leaves.charAt(i ) + "\t");
  for ( int j = 0; j < m; j++ )
 System.out.print( Matrix[ i ][j]+ "\t");
  System.out.println();
 }
}


//Tree is translated
//from link notation to binary.
//Output in Record[]: even entries= addresses, odd = names.
//Associated matrix is also calculated
public static double [][] associatedMatrix(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
 if (print)
 System.out.println("Tree  at translate = "    + tree);
 //String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
  System.out.println("Selected tree in " +
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }
```

```
 //A vector with addresses and names is composed
 makeInputInner(bTree);
 //Paths from leaves to node a are found.
 //Reverse paths are also listed.
 pathFinder(bTree);
 //The distance matrix among leaves is calculated
 double M[][] = matrixBuilder();
 printMatrix(M,
 "Target Distance Matrix = \n"
 + "Distance Matrix associated to the leaves"
 + " of target tree");
 return M;
}



//=========================================================
//================== Evolution of fitting =================
//Lengths are added to given tree topology and evolved
//to find the best fitting of  topology
//to target distance matrix.
//Returned value at evolveLengths(String topology) =
//the distance between
//target matrix and the matrix associated to
//the best fitting of this topology.
//=========================================================




//Population of lengths is initialized at random.
//Lengths take values on digital numbers 0,...,9
private void initialize()
{
 for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
```

```
 oldLengths[i][j][k] = r.nextInt(maxLength);

}

//Recently generated new population becomes the old one
private static void newToOld()
{
for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = newLengths[i][j][k];

}

//Old lengths are copied to a service vector
private static void oldToService(int i)
{

for(int k = 0; k < numberOfLeaves; k++)
 {lengthsLeaves[k] =  (int) oldLengths[i][0][k];}

for(int k = 0; k < numberOfLeaves; k++)
{lengthsInnerLinks[k] =  (int) oldLengths[i][1][k];}
}


//The absolute distance among two matrices is calculated.
//Only triangular part of matrices are compared.
private static double absDistance(int n,
               double[][] M1, double[][] M2)
{
double abs = 0;
for(int i = 0; i < n; i++)
for(int j = i+1; j < n; j++)
 abs = abs + Math.abs(M1[i][j]-M2[i][j]);
return abs;
}


//Input : tree in link notation.
```

```
//Output:
//distance matrix associated to the leaves of the tree.
private static double[][] distanceMatrix(String tree)
{
String s = toQuasiBinaryInner(tree) ;
if (print4)
 System.out.println("\n Tree = " + s);
//Paths from leaves to node a are found.
//Reverse paths are also listed.
pathFinder(s);
//The distance matrix among leaves is calculated
double M[][] = matrixBuilder();
return M;
}



//A tree topology is received as input.
//Lengths from oldLengths are inserted into the topology
//to get a tree with lengths.
//The distance between target matrix and
//the associated distance matrix to tree is calculated.
//The minimal distance is found:
//this renders the fittest tree.
private double findDistances(String topology)
{
 for(int i = 0; i < N; i++)
 {
 oldToService(i);

  String tree = insertLengths(topology);
  if (print4)
 System.out.println("\n Tree = " + tree);
  double M[][] = distanceMatrix(tree);
  if (print4)
{
 printMatrix(M,
 "Distance Matrix associated to   leaves"
 + " of  tree");
```

```
}
  double d= absDistance(numberOfLeaves,targetMatrix, M);
  if (print3)
  System.out.println("\n Individual " + i + ":\n" + tree);
  //Distance from the associated distance matrix of the tree
  //to target matrix is evaluated
  distance[i] = d;
  if (print4)
  System.out.println(i + " Dist to tarrrrrrget = "
                                  + distance[i]);


 }//End of for(int i...

 //The fittest individual minimizes distance function
 double min = distance[0];
 best = 0;
 for(int i = 1; i < N; i++)
 {
  if (distance[i] < min )
  {
   min = distance[i];
       best = i;
  }
 }
 //lengths are copied to  service vectors
 oldToService(best);
 //Best tree for the given topology
 bestTree = insertLengths(topology);



 if (print4)
 {
  System.out.println("\n Distance + BestTreeeeeee  = " +
  + min + " \n"  + bestTree
             + "  "  + "\n");
 }
 return min;
}
```

```
private void makeFitnesses()
{
 double maxD = distance[0];
 double minD = distance[0];
 double b = 1 ;
 //Nin and max distances are found
 for(int i = 1; i < N; i++)
   {
 if ( distance[i] > maxD ) maxD =  distance[i];
 if ( distance[i] < minD ) minD =  distance[i];
   }

 if (minD == maxD) //All   equally fit
 {
  for(int i = 0; i < N; i++)
fitness[i] = 1;
 }
 else
 {
 //Distance to target determine fitness:
//fitness[maxD] = 0;    fitness[minD] = 1;
 for(int i = 0; i < N; i++)
 {
  b = 1;
  b = b/(minD - maxD);
  double a = -b*maxD;
  fitness[i] = a + b * distance[i];
  if (print3)
   {
  System.out.println("Individual = "
             + i + " has distance " +
          distance[i] + " fitness " + fitness[i]);
   }
 }
 }
}
//Distances are translated to
//probabilities of not reproducing.
//N individuals are sampled at random from the population
```

```
//according to those probabilities
private void reproduction()
{
 makeFitnesses();
 //The best is cloned
 for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[0][0][j] = oldLengths[best][0][j];
 newLengths[0][1][j] = oldLengths[best][1][j];
}

 //N-1 children are sampled
 for(int i = 1; i < N; i++)
  {
 //flag
 boolean go = true;
   while (go)
    {
 //An individual is chosen at random
 int k = r.nextInt(N);
 //k is reproduced according to its fitness
 double q = r.nextDouble();
 if (print4)
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] );
 if ( q < fitness[k]) //make a copy of k
  {
   for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[i][0][j] = oldLengths[k][0][j];
 newLengths[i][1][j] = oldLengths[k][1][j];
}
 go = false;//Pass to making a new individual
   if (print4)
 System.out.println("Individual = "
                       + k + " with fitness " +
 fitness[k] + "->" + i);
  }
```

```
    }

  }//End of for //N-1 children are sampled
}



//Individuals are mutated
private void mutation()
{
 //The old best, i = 0, is conserved intact
 for(int i = 1; i < N; i++)
   {
//Mutation in links that connect leaves
for(int k = 0; k < numberOfLeaves; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][0][k] = r.nextInt(maxLength);

 }
//Mutation in links that connect inner nodes
for(int k = 0; k < numberOfLeaves-2; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][1][k] = r.nextInt(maxLength);

 }
   }
}


//Individual l is recombined to another
//that is chosen at random
private static void recInd(int l)
{
 //leaf else inner node?
 int a = r.nextInt(2);
 //Partner at random
```

```
 int k = r.nextInt(N);

 if (a == 0)
  {
   //Random site for recombination
   int site = r.nextInt(numberOfLeaves);
   //Take a part from l
   for(int j = 0; j < site; j++)
  newLengths[l][0][j] = oldLengths[l][0][j];
   //Another part from k
   for(int j = site; j < numberOfLeaves; j++)
  newLengths[l][0][j] = oldLengths[k][0][j];
  }
 else
 {
  int site = r.nextInt(numberOfLeaves-2);
  for(int j = 0; j < site; j++)
  newLengths[l][1][j] = oldLengths[l][1][j];
  for(int j = site; j < numberOfLeaves; j++)
 newLengths[l][1][j] = oldLengths[k][1][j];
}
}



//Recombination master
private static void recombination()
{
if (print) System.out.println("\nRECOMBINATION\n");
newToOld();
//Individual l>0 is a recombinant version
for(int l=1; l< N; l++)
{
  recInd( l );
}
}
```

```
//Lengths are evolved to find the best fitting
//of the given topology to target matrix
private double evolveLengths(String topology)
{
 if(print4)
 System.out.println("\n========Evolution of lengths " +
  "for topology = "   + topology );
 initialize();
 double min = -1;
 for(int gen = 0; gen < numberOfGenerations; gen++)
 {
  if (print4)
System.out.println("\n=== Gen = " + gen);
  min =  findDistances(topology);
  reproduction();
  mutation();
  recombination();
  newToOld();
 }

 return min;
}




//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree11 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
```

```
protected JTree tree;
//Next leaf to be grafted
char nextLeaf = leaves.charAt(3);
//Next internal node to be inserted
char nextIntNode = internalNodes.charAt(1);
//counter of generations
int counterGen = 0;


//Constructor
public DynamicTree11()
{
   super(new GridLayout(1, 0));

   rootNode = new DefaultMutableTreeNode(seed);
   treeModel = new DefaultTreeModel(rootNode);

   tree = new JTree(treeModel);
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);

   JScrollPane scrollPane = new JScrollPane(tree);
   add(scrollPane);
 }


 /** Selected tree is unveiled */
 public String unveilTree()
 {
   DefaultMutableTreeNode selectedNode = null;
   TreePath selectedPath = tree.getSelectionPath();

   if (selectedPath == null) {
    selectedNode = rootNode;
   } else {
    selectedNode = (DefaultMutableTreeNode)
            (selectedPath.getLastPathComponent());
   }
```

```
    String tree =  selectedNode.toString();
    if (print)
    System.out.println("\nSelected tree at unveilTree() = "
                        + tree);
    return tree;
  }


  //Add a node
  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
               new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
             (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                   (new TreePath(childNode.getPath()));
    }
    return childNode;
  }



//Leaves are found and children are generated.
private DefaultMutableTreeNode reproductionOfLeaves()
 {
  if (print)
System.out.println("counterGen = " + counterGen );
```

```
   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();
   int indexOfTopologies = 0;

while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
if (print)
System.out.println("ParentTree = " +  parentTree );

//=================

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
```

```
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
       + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 if ((4+counterGen)== numberOfLeaves)
 {

  //Lengths are added and evolved to find the
  //best fitting of this topology
  //to target distance matrix.
  //Returned value: the distance between
  //target matrix and the matrix associated to
  //the best fitting of this topology.

  double distance = evolveLengths(newTree);
  System.out.println("\n====================" +
   "\nIndex + Topology + " +
   "distance to target + best tree: \n"
  + indexOfTopologies
   +  ":  "  +
  newTree + " -> "   + simplify(newTree)
 + " " +  distance  + "\n" + bestTree + "\n");
 if (print5)
 {
//Distance from M(tree)
```

```
 //to target matrix is evaluated
 double M[][] = distanceMatrix(bestTree);
  if (print5)
{
 printMatrix(M,
 "Distance Matrix associated to leaves"
 + " of  tree");
}
  double d= absDistance(numberOfLeaves,
                  targetMatrix,
M);

System.out.println("Dist to target = " + d);
 }
  indexOfTopologies++;
 }
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());
```

```
      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }

    public void treeNodesRemoved(TreeModelEvent e)
    {
    }

    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree11



//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
```

```
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.

//Resultant trees are drawn.




class DrawBiTree7 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane28;
JScrollPane scroller;




//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;
```

```
  String  NwTree = "";


 boolean print = false;


//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];



//Set up the instructions.
JLabel instructionsLeft;

public DrawBiTree7()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (pseudo-Newick's notation) = "
  + simplify(selectedTree));

instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);

//Set up the drawing area.
```

```
drawingPane28 = new drawingPane28();
drawingPane28.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane28);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//========================================
//============= Painting class ============
//========================================


/** The component inside the scroll pane. */
 class drawingPane28 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;
```

```
//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane28.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
```

```
//turn on scrolling bars.
drawingPane28.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                                + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
```

```
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
```

```
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
```

```
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane28



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame =
   new JFrame("Tree (Link notation) = "
                  + simplify(selectedTree));
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree7();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
```

```
}

}//End of  inner class DrawBiTree7

}//End of main class J162 EvoTree
```

163 ***Exercise.*** *Run the previous program and play with the code. Enable the printing options, around line 155, to somehow check the correctness of the program.*

164 ***Exercise.*** *Does the code stand the test of ending with the same tree with which it began? Answer*

165 ***Challenge.*** *The Author never tested the correct implementation of recombination in the previous program. Devise, implement and run an appropriate test.*

166 ***Challenge.*** *Verify that the performance of the code depends on the combination of size population and mutation rate. A recombination rate is not implemented. Include one and make an overall optimization, i.e., find the dynamic relation among parameters to get the fastest convergence.*

167 ***Challenge.*** *Study the mode of implementing selection, around line 1249, and devise a better rule with faster convergence.*

168 ***Exercise.*** *Improve the performance of the GUI: show the tree that best fits the target matrix (associated to target tree) and in response to a prompt of the User over a topology show the best fitting tree restricted to that topology. Answer*

169 ***Challenge.*** *The Author did not solved the problem of showing at the same time the target tree and the best fitting tree. This would gracefully serve the purpose of testing that our software fulfills the elementary requirement of ending with the same tree with which it began. Solve this problem.*

170 ***Exercise.*** *Include in the previous program a halting instruction to automatically end when a perfect matching is attained. A solution by the Author will be included in the program that solves the next exercise.*

171 ***Exercise.*** *We have worked with the family of n-gram distances. Moreover, we have begun with trees to devise distance matrices. By construction, our distances matrices are indeed distance matrices. There are, nevertheless, many more procedures to calculate distances in within pairs of texts. So, let us add to our program the possibility to begin with a distance matrix directly. A test for being a distance matrix must be run at entrance for not every matrix is a distance matrix. Answer*

172 ***Exercise: the program is exasperating!*** *Run the previous program changing the number of leaves in the tree that gives rise to the target matrix. Play enough to confidently agree else disagree with the Author: the program runs very slowly even for so low number of leaves as 6. It is so lazy that patience gets burnt before getting perfect matching with the expected distance = 0.*

Can we invent a form to accelerate convergence?

173 ***Let us try to help ourselves!*** *We have observed that the correct topology has the fastest convergence. This allows us to try the next approach: all topologies are scanned and run during some few generations to choose a candidate to be the correct topology: it is that with the less achieved distance to target. We have chosen 100 generations for 6 leaves and a size population of 1000. When a promising topology has been chosen, we run it in isolation to finely tune lengths of links. Besides, we launch the program various times, a task that in Eclipse can be done in parallel. The code follows:*

```
//Program J173 EvoTree4
/*
 * We generate a random tree topology
 * and attach random integer lengths to links.
 * In that way we get the target tree, T.
 * Next, we find the associated distance matrix D  to T.
 * Our problem is:
 *
 * Find tree t such that the absolute distance between
 * matrix D and the associated matrix to t is a minimum.
 * Absolute distance between M and N = sum |Mij-Nij|.
 *
 * We run the next test:
 * If we begin with a tree T, compute D, and then t,
 * we expect to return with the initial tree: t = T.
 *
 * To associate a distance matrix to a tree,
 * we calculate the distance in within
 * each pair of nodes (leaves + inner nodes).
 * To that aim, we begin with the first node and
 * walk over the tree until the second node
 * and add the lengths of all traversed links.
 * The resulting number is the  distance.
```

```
 *
 * To find the tree that best fit D,
 * we  scan all possible topologies,
 * each one of which is submitted to evolution
 * to finely tune lengths.
 *
 * Formats = Link and Newick's notations. And variants.
 *
 * The performance of the GUI has been improved.
 * An automatic halting option has been added to be applied
 * if perfect matching is found.
 * The program accepts for input
 * a distance matrix with integer values.
 * A test for being a distance matrix is run.
 *
 * Efficiency is slightly improved.
 */



import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
```

```
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class EvoTree4 extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;

 //The program accepts a distance matrix as input
 //else
 //it generates a random tree, calculates the distance
 //matrix associated to leaves and posits it
 //as target matrix.
 private static boolean inputMatrix = true;
 private static boolean randomTree = (!(inputMatrix));


//Input distance matrix:
//Example 5.4, pag 99, Isaev (2001)
//Introduction to Mathematical Methods in Bioinformatics
//Springer
private static double targetMatrix[][]
 = {{0,8,3,14,10,12},
   {8,0,9,10,6,8},
   {3,9,0,15,11,13},
   {14,10,15,0,10,8},
   {10,6,11,10,0,8},
   {12,8,13,8,8,0}};

private static double targetMatrix5[][]
 = {{0,8,3,14,10},
   {8,0,9,10,6},
   {3,9,0,15,11},
```

```
   {14,10,15,0,10},
   {10,6,11,10,0}};

private static double targetMatrix4[][]
 = {{0,8,3,14},
   {8,0,9,10},
   {3,9,0,15},
   {14,10,15,0}};

//Number of leaves that must go in all trees > 3
private static int numberOfLeaves = targetMatrix.length;

 //Everything is built by recursion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 private static String internalNodes =
 "abcdefghijklmnopqrstuvwxyz";

 //Next leaf to be grafted
 private static  int nextLeafIndex = 2;
 //Next internal node to be inserted
 private static  int nextIntNodeIndex = 0;


 static int maxNumberOfLeaves = leaves.length();


 private static String[] Record =  new String[10000];

 private static  int indexOfTopologies;
 //The distance to target of the best fitting tree
 //of the given topology
 private static double[] bestFitting =  new double[10000];
 private static String[] bestTreeOfTopology
                             = new String[10000];

 //Last filled position  in Record[]
```

```
 private static int lastUsedPosition;
 private static String selectedTree = "";


  //DynamicTree is an inner service class that comes
  //at the end of this file.
  private static DynamicTree14 treePanel;
  private static String VISUALIZE_COMMAND = "Visualize";

   //Length of the link that glues a leaf to the tree
  private static int lengthsLeaves[]  =
               new int[numberOfLeaves];
  //Length of the link that glues
  //an inner node to another one
  private static int lengthsInnerLinks[]  =
              new int[numberOfLeaves];


  private static Random r = new Random();
  //The control parameter of the random generator for int
  private static int maxLength = 10;

  //Paths from each leaf to node a
  private static String[] Paths =
              new String[numberOfLeaves];

  private static String targetTree;



  //=======Evolution parameters============



//N individuals at all.
static private int N = 100;
static private double oldLengths[][][] =
 new double[N][2][maxNumberOfLeaves];

static private double newLengths[][][] =
```

```
new double[N][2][maxNumberOfLeaves];

static private double distance[] = new double[N];
static private double fitness[] = new double[N];

//Number of generations for phase 1 =
//choose the best topology
static private int numberOfGenerations = 30;

//Mutation rate per site (= entry) per generation
static private double    m = 0.5;

//Index of the best
static private int best;
//The best tree for a given topology
static private String bestTree = "";
static private boolean reporting = false;
static private long genOfSolution = 0;

//=========Debugging help ===============

private static boolean print  = false;
private static boolean print2 = false;
private static boolean print3 = false;
private static boolean print4 = false;
//Turn on print5 firstly
private static boolean print5 = false;

//==============

private static String titleFrame = "";
private static String title1 = "";
private static String title2 = "";
private static EvoTree4 newContentPane = new EvoTree4();



  //=======Constructor of main class: GUI basics=====

  public EvoTree4()
```

```
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructions1 = new JLabel(
      "Select  a tree and click the Visualize button.");


   instructionPanel.setFocusable(true);
   instructionPanel.add(instructions1);

   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree14();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(700, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
      //Selected topology must be read
      selectedTree = treePanel.unveilTree();
      System.out.println("\nSelected topology " +
       "in Link notation  after unveilTree() = \n"
       + selectedTree);
      /*Random lengths are added to topology
      generateLengths();
      //Lengths are added
      selectedTree = insertLengths(selectedTree);
```

```
      */
      int i =  findTheBest(selectedTree);
      selectedTree =  bestTreeOfTopology[i];
      double d = bestFitting[i];
      System.out.println( " Best fitting = \n " +
                                  selectedTree);
      System.out.println( " Best distance = \n "
       + d);


      decoderLengths(selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      double[][] M =  associatedMatrix(selectedTree);
      printMatrix(M, "distance matrix associated to the " +
      "best tree for this topology");
      //Selected tree is drawn
      title1 = "Best tree for the topology = "
       + simplify(selectedTree);
      title2 = "Distance to target = " + d;
      DrawBiTree11 a = newContentPane.new DrawBiTree11();
      a.createAndShowGUI();

 }
  }

//======================

public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
     public void run()
     {
      //If input is a distance matrix:
      if (inputMatrix)
      {
       printMatrix(targetMatrix, "Input Matrix");
```

```
   boolean isD = isMatrixDistanceMatrix(targetMatrix);
     if (isD) System.out.println("\nMatrix " +
    "is a distance matrix");
     else
      {
    System.out.println("Matrix " +
         "is NOT a distance matrix");
       }
  }
else//if a random tree is generated -> Matrix
 {
 String tree = randomTopology();
 generateLengths();
 targetTree = insertLengths(tree);
 System.out.println("Target tree: each link " +
 "has its own length = \n" + targetTree);
 targetMatrix  =
     new double[numberOfLeaves][numberOfLeaves];
 targetMatrix = associatedMatrix(targetTree);
  //Draw Target tree: this malfunctions.
    /*
     {
      titleFrame = "Target tree = " + tree;
       selectedTree = targetTree;
       DrawBiTree11 b = newContentPane.new DrawBiTree11();
       b.createAndShowGUI();
       }
      */
 }
createAndShowGUI();

System.out.println("\nDistance to target = " +
      " Absolute distance between" +
     " target matrix  and \n" +
     "the distance matrix associated to " +
     "the tree with lengths of links that \n" +
     "is the best fitting  of the topology.\n" +
     "Perfect fitting: distance = 0 \n");
 System.out.println("\nNumber of Generations = "
       + numberOfGenerations);
```

```
     System.out.println("\nMutation rate = "   + m);
     //To find the best fitting, all tree topologies are
     //scanned and each one of them is evolved for
     //the best particular fitting.
     //All this is done along the process of
     //generating all topologies
     String bestTopology = generateTopologies();
     System.out.println("\nBEST FITTING TOPOLOGY = \n" +
      deleteNumbers(bestTopology));
    //Lengths are finely tuned
      optimizeLengths(deleteNumbers(bestTopology));
     }
    });
  }


//Test for being a distance matrix.
//MaxLenght = max value among entries is calculated.
private static boolean
      isMatrixDistanceMatrix(double[][] M)
{
int l = M.length;
boolean isD = true;
//Is M(u,u) = 0?
for(int u = 0; u < l; u++)

if (M[u][u] ==  0) ;
 else
  {
isD = false;
return isD;
  }

//Is M(u,v) = M(v,u)?
for(int u = 0; u < l; u++)
 for(int v = u+1; v < l; v++)
if (M[u][v] == M[v][u] ) ;
 else
{
 isD = false;
```

```
 return isD;
}

//Is M(u,v)>0 for u not v?
for(int u = 0; u < l; u++)
 for(int v = u+1; v < l; v++)
if (M[u][v] >  0) ;
 else
{
 isD = false;
 return isD;
}



//Is M(u,v) < M(u,w) + M(w,v)?
for(int u = 0; u < l; u++)
for(int v = 0; v < l; v++)
 for(int w = 0; w < l; w++)

if (M[u][v] <= M[u][w]+M[w][v]  ) ;
 else
{
 isD = false;
 return isD;
}
double m = 0;
for(int u = 0; u < l; u++)
 for(int v = u+1; v < l; v++)
 if (M[u][v] > m ) m = M[u][v];
maxLength = (int) m + 1;
return true;
}



 //From parentTree, a tree in Newick's notation,
 //a Child  tree is generated by a generative grammar.
 //A random election is made among all possibilities.
 private static String generateChild(String parentTree)
```

```
{

 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);


 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = parentTree.length();
 //counter of links = 0;
 int counter = 0;
 //Links are counted
 for(int i = 0; i < l; i++ )
  {
   //parentTree is scanned char by char
   char I = parentTree.charAt(i);
 //Is I an internal node that denotes a link?
 if (charIsInternalNode(I)) counter++;
  }
 //A link is chosen
 int k = r.nextInt(counter);
 if (print2)
 System.out.println("Random number = " + k);
 //new branch is grafted at k
 counter = 0;
 for(int i = 0; i < l; i++ )
  {
    //parentTree is scanned char by char
 char I = parentTree.charAt(i);
 //Is I an internal node that denotes a replacement?
 if (charIsInternalNode(I))
  {
```

```
    if (counter == k) //Is I the chosen link?
     {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree, i,
                 nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
    //Link to be replaced
    link = "" + a + I;
    //Replacement rule 1:
    //aI -> (aN,FN)I;   N= nextIntNode;
    //F = nextLeaf
   replacement = "(" + a + nextIntNode + ","
            + nextLeaf + nextIntNode + ")" + I;
   //Rule 1 is executed
   newTree = parentTree.replace(link, replacement);
   }
  if (print2)
  System.out.println(newTree + " -> "
                        + simplify(newTree));

   i = l;
     }
    counter++;
  }
   }
  return newTree;
}


//A tree topology is generated at random.
public static String randomTopology()
{
 if (print2)
 System.out.println("Parent tree = \n" + seed
  + " -> "    + simplify(seed));
 String tree = seed;
 int limit = numberOfLeaves-3; //less than 11.
```

```
  for(int i = 0; i < limit; i++ )
  tree = generateChild(tree);
  System.out.println("Target topology = \n" + tree);
  return tree;
 }


//======================


//Random lengths are added to a tree in Link notation
public static void generateLengths()
 {
  //Lengths with digit values are generated
  for(int i = 0; i < numberOfLeaves; i++)
   {
 //Zero lengths are not allowed
 lengthsLeaves[i] = 1 + r.nextInt(maxLength - 1);
 lengthsInnerLinks[i] = 1 +  r.nextInt(maxLength - 1);
   }
 }


//The best fitting tree for chosen topology is found
public static int findTheBest(String tree)
{
 String selectedTopology = simplify(tree);
 String thisTopology = "";
 String thisTree = "";
 int bestPosition = -1;

for(int i = 0; i < indexOfTopologies; i++)
 {
  thisTree = bestTreeOfTopology[i];
  thisTopology = simplify(thisTree);
  if (selectedTopology.equals(thisTopology))
  {
   bestPosition = i;
   return bestPosition;
  }
```

```
 }
return  bestPosition;
}


  /*
  //Lengths with real values are generated
  double lengthsLeaves[]  = new double[l];
  //Last two entries are redundant
  double lengthsInnerLinks[]  = new double[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextDouble();
 lengthsInnerLinks[i] =  r.nextDouble();
   }
  */

//Random lengths are added to a tree in Link notation
public static String insertLengths(String treeLinkN)
 {
  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
  //TreeLinkN is scanned char by char
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    if (print)
    System.out.println(newTree.length() + " " + newTree);
    //Lengths are appended
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      double j = lengthsLeaves[k];
     newTree = newTree + ":" + j;
     }
    if (internalNodes.contains(thisChar))
    {
```

```
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerLinks[k];
     newTree = newTree + ":" + j;
    }
   }
   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Tree of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.

  // content panes must be opaque
  newContentPane.setOpaque(true);
```

```
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Numbers are delete from full content
private static String deleteNumbers(String newTree)
{
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
  char c = newTree.charAt(i);
  String s = "" + c;
  //Is c numeric? delete it!
  if (numeric.contains(s));
  else //add char c to output
    tree = tree + c;
}
return tree;
}


//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
```

```
{
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
```

```
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
```

```
return newTree;
}

//Input: string tree with complete information.
//Work: lengthsLeaves[] and lengthsInnerLinks[]
//are read from tree.
private static void decoderLengths(String tree)
{
 if(print2)
 System.out.println(
"Input tree at recoverLengths = \n "   + tree  );
 //Delete :
 tree = tree.replaceAll(":", "");
 if(print2)
 System.out.println(
"Input tree without : = \n "   + tree  );
 int n = tree.length();
 String thisChar = "";
 String nextChar = "";
 String numbers = ".0123456789";
 boolean leaf = false;
 boolean node = false;
 String Leaf = "";
 String numberLeaf = "";
 String numberNode = "";
 String Node = "";
 int j ;
 double f;

//All chars in tree are scanned
 for(int i=0; i < n; i++)
  {
   thisChar = "" + tree.charAt(i);
   if (i < n-1) nextChar  = "" + tree.charAt(i+1);
   //Do we have a leaf?
   if (leaves.contains(thisChar))
    {
 leaf = true;
 Leaf = "" + thisChar;
    }
```

```
   //Do we have an internal node?
   if (internalNodes.contains(thisChar))
   {
 node = true;
 Node = "" + thisChar;
   }

   //Do we have a number that belongs in a leaf?
   if ( (leaf) & (numbers.contains(thisChar)))
{
 numberLeaf = numberLeaf + thisChar;
}

  //Do we have a number that belongs in a leaf?
   if ( (node) & (numbers.contains(thisChar)))
{
 numberNode = numberNode + thisChar;
}

   //Must we end accumulation of numbers of a leaf?
   if ( (leaf) & (internalNodes.contains(nextChar)))
{
  leaf = false;
  j = leaves.indexOf(Leaf);
  if (print4)
  System.out.println("Leaf = " + Leaf);
  f = new Double(numberLeaf);
  lengthsLeaves[j] = (int) Math.floor(f);
  numberLeaf = "";
}

   //Must we end accumulation of numbers of a node?
   if ( (node) & (!(numbers.contains(nextChar))))
{
  node = false;
  j = internalNodes.indexOf(Node);
  if (print4)
  System.out.println("Node = " + Node);
  f = new Double(numberNode);
```

```
  lengthsInnerLinks[j] = (int) Math.floor(f);
  numberNode = "";
}

  }
}


//Topologies are generated and
//the best fitting is filtered
private static String generateTopologies()
 {
  //New leaves are incorporated one by one
  for(int i = 0; i < numberOfLeaves - 3; i++)
   {
    if (print) System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
   }
  //The best topology is found
  double minD = bestFitting[0];
  int place = 0;
  for(int i = 1; i <   indexOfTopologies; i++)
 {
   if ( bestFitting[i] < minD )
{
 minD =  bestFitting[i];
     place = i;
}
 }

  System.out.println("\n\n\n============================" +
   "\nBest Topology is chosen " +
   "\nNumber of generations = " +
numberOfGenerations +
               "\nPopulation size = " + N
   + "\n============================\n");
  System.out.println();
  //Was initializing through a random tree?
  if (randomTree)
  {
```

```
  System.out.println("TARGET TOPOLOGY = \n" +
  simplify(targetTree));
  System.out.println("TARGET TREE = \n" + targetTree);
  }
  printMatrix(targetMatrix, "TARGET DISTANCE MATRIX  ");
  String theBestTree = bestTreeOfTopology[place];
  System.out.println("\nBEST FITTING TOPOLOGY = \n" +
  simplify(theBestTree));
  System.out.println(
    "\nBEST DISTANCE AND FITTING TREE TO TARGET MATRIX  "
    + minD + "\n"    + theBestTree);


  //Distance from M(bestTree)
  //to target matrix is evaluated


  decoderLengths(theBestTree);
  double M[][] = associatedMatrix(theBestTree);
  double d= absDistance(numberOfLeaves,
             targetMatrix, M);
  printMatrix(M, "Distance Matrix associated to " +
                    "the leaves of the best tree");
  System.out.println("Distance to target matrix = "   + d);
  //Draw theBestTree
  titleFrame = "Best fitting tree";
  title1 =  simplify(theBestTree);
  selectedTree = theBestTree;
  DrawBiTree11 c = newContentPane.new DrawBiTree11();
  c.createAndShowGUI();
  c.setLocation(500,  100);
  return  theBestTree;
 }



//========= Translation in within formats ==========
```

```
//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput ////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
```

```
}



//Outputs a word in reversed order.
private static String reverser(String word)
{
 return new StringBuilder(word).reverse().toString();
}


private static String cleanTree(String tree)
{
 if (print2)
 System.out.println("Input tree at cleanTree = \n "
        + tree);
 String treeClean = "";
 String numeric = ".:0123456789";
 String plusMinus = "+-";
 String thisChar = "";
  boolean leaf = false;
 //Numerics are washed
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if (numeric.contains(thisChar));
    else treeClean = treeClean + thisChar;
  }
 if (print2)
 System.out.println("Tree without numerics = \n"
+ treeClean);
 //+/- are deleted from inner nodes that accompany leaves
 l = treeClean.length();
 String treeOut = "";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
 {
```

```
  thisChar = "" + treeClean.charAt(i);
  if (leaves.contains(thisChar))
  leaf  = true;
  if ((leaf) &  (plusMinus.contains(thisChar)));
else treeOut = treeOut + thisChar;
  if (internalNodes.contains(thisChar))
  leaf  = false;
}

 if (print2)
 System.out.println("Tree at output of cleanTree = \n"
+ treeOut);
return treeOut;

}



//The path from each leaf to node a is found.
//The input contains branch's indicator, say:
//((++Aa,+-Ba),(-+Cb,((--++Dd,--+-Fd)--c,---Ec)-b)+a)
//Output for leaf A: Aa+.
//Output for leaf C: Cba-
//Output for leaf D: Ddcba-
private static void findPaths(String tree)
{
 if (print2)
 System.out.println("Input tree at findPaths = \n" + tree);
 //length of String   tree
 int l = tree.length();
 String plusMinus = "+-";
 String thisChar;
 String nextChar;
 String path = "";
 //Branch indicative with respect to root, + else -
 String branch;
 String address ;
 //The number of links of a path to root
 int level = 0;
```

```
 String treeReversed = reverser(tree);
 if (print2)
 System.out.println("Reversed tree = " + treeReversed);

//Each leaf has a path to a
for(int k= 0; k < numberOfLeaves; k++)
{
 String leaf = "" + leaves.charAt(k);
 //Candidate nodes to be part of the path to node a
 //must have  sign +/- as prefix.
 //Or as suffix in the reversed tree.
 int position = treeReversed.indexOf(leaf);
 if (position > -1) // if leaf is in tree
 {
  if (print2)
  System.out.println("\nLeaf = " + leaf );
  address = "";
  boolean go = true;
  //branchIndicator and level are found
  for(int i = position+1 ; ((i < l) & (go)); i++ )
  {
   thisChar = "" + treeReversed.charAt(i);
   if   (plusMinus.contains(thisChar)  )
address = thisChar + address;
   else go = false;
  }
  level = address.length();
  if (print2)
  System.out.println("LeafAddressLevel  = " + leaf
                     + address + level);
  branch = "" + address.charAt(0);


  //Path from leaf to node a is found in (not reversed) tree
  position = tree.indexOf(leaf);
  thisChar = "" + tree.charAt(position);
  nextChar = "" + tree.charAt(position + 1);
  //Output
  path = "" + leaf + nextChar;
  go = true;
```

```
  if  (nextChar.contentEquals("a") ) //leaf is neighbor of a
   {
  path = path + branch;
  if (print)
  System.out.println("Path short = " + path);
   }
  else  //level > 1
  {

  //Tree is scanned char by char to get path
  String iChar = "";
  for(int i = position + 2; ((i < l) & (go)); i++ )
  {
   iChar = "" + tree.charAt(i);


   int newLevel = 0;
   if (internalNodes.contains(iChar))//InternalNodes
{
     if   (iChar.contentEquals("a")  )
      {
   path  = path +  iChar;
   go = false; //end
      }
     else //Internal node different than a
     {
      if (print)
      System.out.println("Internal node not a) = " + iChar );
      int rP = l -i-1; //index of iChar in treeReversed
      String  nextNewChar = "" + treeReversed.charAt(rP+1);

      if ( plusMinus.contains(nextNewChar )) //candidate node
       {
       if (print)
     System.out.println("next candidate = " + iChar);
        String addressNew = "";

        //level of candidate node iChar  is inquired
        String thatChar = "";
        boolean go2 = true;
```

```
        for(int j = rP+1 ; ((j < l) & (go2)); j++ )
         {
       thatChar = "" + treeReversed.charAt(j);
       if   (plusMinus.contains(thatChar)  )
          addressNew = thatChar + addressNew;
       else go2 = false;
         }
        if (print)
        System.out.println("AddressNew " +
                                    iChar + addressNew);
       newLevel = addressNew.length();
       //A path links a node with another in a lower level
       if  ( (newLevel < level) &
        (!(nextChar.contentEquals(iChar))))
        {
       path = path + iChar;
       level = newLevel;
       if (print)
       System.out.println("Path newLevel= " + path);
        }
      } //End of if candidate node
     }//End of else Internal node different than a
}//End of if InternalNodes
   }//End of Tree is scanned char by char

  path = path + branch;

 }//End of  else level > 1
 }//End of if leaf is in tree
 if (print2)
 System.out.println("Path completed  = " + path);
 Paths[k] = path;
}//End of each leaf
}



//Paths from leaves to node a are found
//Reverse paths are also calculated.
//Input tree in quasibinary notation
```

```
//with addresses and lengths
private static void pathFinder(String tree)
{
 //Useless information is deleted
 String tree2 = cleanTree(tree);
 //Find the path to node a from each leaf.
 //Add an indicator of the branch
 //with respect to the root: up(+) else down(-)
  findPaths(tree2);
}

//Input: a long raw string with path from leaf to node a
//plus path from node a to leaf.
//Editing is done to get true paths.
private static String grammar (String path)
{
 int n = path.length();
 String thisChar = "";
 String nextChar = "";
 String oldPath = path;
 String plusMinus = "a+-a";
 String minusPlus = "a-+a";
 String two = "";
 int N = path.length();
//Rule 1: two equal symbols produce nothing
 for(int j=1; j < N; j++)
  {
   for(int i=1; i < n-1;)
   {
    thisChar = "" + path.charAt(i);
    nextChar = "" + path.charAt(i+1);
    two = "" + thisChar + nextChar;
    if (thisChar.contentEquals(nextChar))
  {
   i = i + 2;
   path = path.replace(two, "");
  }
    else
  {
   i = i+1;
```

```
  }
  n = path.length();
   }
  }

 //Rule 2: a+-a -> a

 if  (path.contains(plusMinus))
      path = path.replace(plusMinus, "a");
 if   (path.contains(minusPlus))
     path = path.replace(minusPlus, "a");
 if (print2)
 System.out.println("input = " + oldPath +
                    " output = " + path);
 return path;
}

//The distance matrix among leaves is calculated
private static double[][] matrixBuilder()
{
double M[][] =
new double[numberOfLeaves][numberOfLeaves];
 //Path to node a from leaf
 String pathLeafToa = "";
 //Path to leaf from node a
 String pathaToLeaf = "";
 //Path from leaf to lead
 String pathLeafiToLeafj = "";
 String innerNode = "";
 double distance;
 for(int i = 0; i < numberOfLeaves; i++)
  for(int j = 0; j < numberOfLeaves; j++)
  {
   pathLeafToa = Paths[i];
   pathaToLeaf = reverser( Paths[j] );
   //Raw paths
   pathLeafiToLeafj = pathLeafToa + pathaToLeaf;
   //An editing grammar to get true paths
   pathLeafiToLeafj = grammar( pathLeafiToLeafj );
   if (i == j) M[i][j] = 0;
```

```
   else
    {
 String leaf = "" + pathLeafiToLeafj.charAt(0);
 int p = leaves.indexOf(leaf);
 distance = lengthsLeaves[p];
 int n = pathLeafiToLeafj.length();
 for(int k = 1; k < n-1; k++)
 {
  innerNode =  "" + pathLeafiToLeafj.charAt(k);
  p = internalNodes.indexOf(innerNode);
  distance = distance + lengthsInnerLinks[p];
 }
 leaf =  "" + pathLeafiToLeafj.charAt(n-1);
 p = leaves.indexOf(leaf);
 distance = distance + lengthsLeaves[p];
  M[i][j] = distance;
    }
  }
 return  M;
}



//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
if (print2)
System.out.println("Input tree at toQuasiBinaryInner " +
                          "= \n" + tree);
//Output
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
```

```
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
```

```
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "+";
   treeToBi =  treeToBi + address + thisChar;

  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (numeric.contains(previousChar))
  address = address + '-';//Add a - to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}
```

```
//bTree  (in links with lengths notation)
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
```

```
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method


//Prints  a matrix and its title
private static void printMatrix(double Matrix[][],
                        String title)
{
System.out.println( "\n" + title );
int m = Matrix.length;
System.out.print(  "\t");
for ( int j = 0; j < m; j++ )
  System.out.print( leaves.charAt(j)+ "\t");
System.out.println();
for ( int i = 0; i < m; i++ )
 {
  System.out.print(leaves.charAt(i ) + "\t");
  for ( int j = 0; j < m; j++ )
 System.out.print( Matrix[ i ][j]+ "\t");
  System.out.println();
 }
}


//Tree is translated
//from link notation to binary.
//Output in Record[]: even entries= addresses, odd = names.
//Associated matrix is also calculated
public static double [][] associatedMatrix(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
```

```
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
//String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
 System.out.println("Input tree  at translate = "    + tree);
  System.out.println("Selected tree in " +
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

 //A vector with addresses and other with names are composed
 makeInputInner(bTree);
 //Paths from leaves to node a are found.
 //Reverse paths are also listed.
 pathFinder(bTree);
 //The distance matrix among leaves is calculated
 double M[][] = matrixBuilder();
 return M;
}


//============================================================
//================== Evolution of fitting ==================
//Lengths are added to given tree topology and evolved
//to find the best fitting of  topology
//to target distance matrix.
//Returned value at evolveLengths(String topology) =
//the distance between
//target matrix and the matrix associated to
//the best fitting of this topology.
//============================================================



//Population of lengths is initialized at random.
```

```
//Lengths take values on digital numbers 0,...,maxLength
private static void initialize()
{
 for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = r.nextInt(maxLength);

}

//Recently generated new population becomes the old one
private static void newToOld()
{
for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = newLengths[i][j][k];

}

//Old lengths are copied to a service vector
private static void oldToService(int i)
{

for(int k = 0; k < numberOfLeaves; k++)
 {lengthsLeaves[k] =  (int) oldLengths[i][0][k];}

for(int k = 0; k < numberOfLeaves-2; k++)
{lengthsInnerLinks[k] =  (int) oldLengths[i][1][k];}
}


//The absolute distance among two matrices is calculated.
//Only triangular part of matrices are compared.
private static double absDistance(int n,
               double[][] M1, double[][] M2)
{
double abs = 0;
for(int i = 0; i < n; i++)
for(int j = i+1; j < n; j++)
```

```
 abs = abs + Math.abs(M1[i][j]-M2[i][j]);
return abs;
}


//Input : tree in link notation.
//Output:
//distance matrix associated to the leaves of the tree.
private static double[][] distanceMatrix(String tree)
{
String s = toQuasiBinaryInner(tree) ;
if (print4)
 System.out.println("\n Tree = " + s);
//Paths from leaves to node a are found.
//Reverse paths are also listed.
pathFinder(s);
//The distance matrix among leaves is calculated
double M[][] = matrixBuilder();
return M;
}


//A tree topology is received as input.
//Lengths from oldLengths are inserted into the topology
//to get a tree with lengths.
//The distance between target matrix and
//the associated distance matrix to tree is calculated.
//The minimal distance is found:
//this renders the fittest tree.
private static double findDistances(String topology)
{
 if (print4)
System.out.println("Topology = " + topology);
for(int i = 0; i < N; i++)
{
//Data preparation
oldToService(i);
//Lengths of links are added to tree topology
String tree = insertLengths(topology);
if (print4)
```

```
{
 System.out.println("Topology = \n" + topology );
 System.out.println( "Tree = \n" + tree);
}
double M[][] = distanceMatrix(tree);
if (print4)
 {
 //The tree generates a distance matrix
 //among pair of leaves
 printMatrix(M,
 "Distance Matrix associated to   leaves"
 + " of  tree");
}

//The absolute distance from M to target matrix:
double d = absDistance(numberOfLeaves,targetMatrix, M);
if (print3)
System.out.println("\n Individual " + i + ":\n" + tree);
//Distance from the associated distance matrix of the tree
//to target matrix is evaluated
distance[i] = d;
if (print4)
System.out.println(i + " Dist to tarrrrrrget = "
                                + distance[i]);

if (reporting)
System.out.println(" \n BEST SOLUTION THUS FAR = \n");

//If solution has been found, halt.
if (d == 0)
{
System.out.println( "\n \nSOLUTION HAS BEEN FOUND: \n" +
"at generation " + genOfSolution);

reporting = true;
}

//System.out.println("\n Tree = " + tree);

if (reporting)
```

```
{
 System.out.println( " Distance to target = " + d
   + " \nSolution tree = " + tree);
 printMatrix(M, "Distance matrix associated to tree");
 printMatrix(targetMatrix, "Target matrix");
 if (d == 0) System.exit(0);
 if (reporting) reporting = false;
}


}//End of for(int i...

//The fittest individual minimizes distance function
double min = distance[0];
best = 0;
for(int i = 1; i < N; i++)
{
if (distance[i] < min )
  {
   min = distance[i];
     best = i;
  }
}
//lengths are copied to  service vectors
oldToService(best);
//Best tree for the given topology
bestTree = insertLengths(topology);



if (print4)
{
System.out.println("\n Distance + BestTreeeeeee  = " +
+ min + " \n"  + bestTree
           + "  "  + "\n");
}
return min;
}
```

```
//Selection is implemented
private static void makeFitnesses()
{
 double maxD = distance[0];
 double minD = distance[0];
 double b = 1 ;
 //Min and max distances are found
 for(int i = 1; i < N; i++)
  {
if ( distance[i] > maxD ) maxD =  distance[i];
if ( distance[i] < minD ) minD =  distance[i];
  }

 if (minD == maxD) //All   equally fit
   {
  for(int i = 0; i < N; i++)
fitness[i] = 1;
   }
 else
  {
 //Distance to target determine fitness:
//fitness[maxD] = 0;    fitness[minD] = 1;
 for(int i = 0; i < N; i++)
 {
  b = 1;
  b = b/(minD - maxD);
  double a = -b*maxD;
  fitness[i] = a + b * distance[i];
  if (print3)
   {
  System.out.println("Individual = "
            + i + " has distance " +
          distance[i] + " fitness " + fitness[i]);
   }
}
  }
}
```

```
//Distances are translated to
//probabilities of not reproducing.
//N individuals are sampled at random from the population
//according to those probabilities
private static void reproduction()
{
 makeFitnesses();
 //The best is cloned
 for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[0][0][j] = oldLengths[best][0][j];
 newLengths[0][1][j] = oldLengths[best][1][j];
}

 //N-1 children are sampled
 for(int i = 1; i < N; i++)
  {
 //flag
 boolean go = true;
   while (go)
    {
 //An individual is chosen at random
 int k = r.nextInt(N);
 //k is reproduced according to its fitness
 double q = r.nextDouble();
 if (print4)
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] );
 if ( q < fitness[k]) //make a copy of k
  {
   for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[i][0][j] = oldLengths[k][0][j];
 newLengths[i][1][j] = oldLengths[k][1][j];
}
 go = false;//Pass to making a new individual
   if (print4)
```

```
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] + "->" + i);
  }

    }

  }//End of for //N-1 children are sampled
}



//Individuals are mutated
private static void mutation()
{
 //The old best, i = 0, is conserved intact
 for(int i = 1; i < N; i++)
   {
//Mutation in links that connect leaves
for(int k = 0; k < numberOfLeaves; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][0][k] = r.nextInt(maxLength);

 }
//Mutation in links that connect inner nodes
for(int k = 0; k < numberOfLeaves-2; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][1][k] = r.nextInt(maxLength);

 }
   }
}


//Individual l is recombined to another
//that is chosen at random
```

```
private static void recInd(int l)
{
 //leaf else inner node?
 int a = r.nextInt(2);
 //Partner at random
 int k = r.nextInt(N);

 if (a == 0)
  {
   //Random site for recombination
   int site = r.nextInt(numberOfLeaves);
   //Take a part from l
   for(int j = 0; j < site; j++)
  newLengths[l][0][j] = oldLengths[l][0][j];
   //Another part from k
   for(int j = site; j < numberOfLeaves; j++)
  newLengths[l][0][j] = oldLengths[k][0][j];
  }
 else
 {
  int site = r.nextInt(numberOfLeaves-2);
  for(int j = 0; j < site; j++)
  newLengths[l][1][j] = oldLengths[l][1][j];
  for(int j = site; j < numberOfLeaves; j++)
 newLengths[l][1][j] = oldLengths[k][1][j];
}
}



//Recombination master
private static void recombination()
{
if (print) System.out.println("\nRECOMBINATION\n");
newToOld();

//Inividual 0 is cloned
for(int j = 0; j <  numberOfLeaves; j++)
  newLengths[0][0][j] = oldLengths[0][0][j];
for(int j = 0; j < numberOfLeaves-2; j++)
```

```
  newLengths[0][1][j] = oldLengths[0][1][j];

//Individual l>0 is a recombinant version
for(int l=1; l< N; l++)
 recInd( l );

}



//Lengths are evolved to find the best fitting
//of the given topology to target matrix
private double evolveLengths(String topology)
{
 if(print4)
 System.out.println("\n========Evolution of lengths " +
  "for topology = "   + topology );
 initialize();
 double min = -1;
 for(int gen = 0; gen < numberOfGenerations; gen++)
 {
  if (print4)
System.out.println("\n=== Gen = " + gen);
  min =  findDistances(topology);
  reproduction();
  mutation();
  recombination();
  newToOld();
 }
 return min;
}


//Lengths are evolved to find the best fitting
//of the (best) topology to target matrix
private static void optimizeLengths(String topology)
{

 System.out.println("\n========Unconstrained Evolution " +
```

```
"of lengths " +  "for topology = \n"
 + deleteNumbers(topology ));
 initialize();
 int pack = 1000;
 int f = 0;
 //An infinite number of generations is allowed.
 //A report is made after each 1000 generations.
 for(int k = 1; k < 1000000; k++)
 {
  for(int gen = 0; gen < pack; gen++)
   {
    if (print4)
 System.out.println("\n=== Gen = " + gen);
    findDistances(topology);
    reproduction();
    mutation();
    recombination();
    newToOld();
    genOfSolution++;
   }
  f = pack * k;
  System.out.println("\n=========GENERATION = " + f);
 reporting = true;
 }
}



//==========================================================
//==================Inner service class=====================
//==========================================================



class DynamicTree14 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
```

```
protected DefaultTreeModel treeModel;
protected JTree tree;
//Next leaf to be grafted
char nextLeaf = leaves.charAt(3);
//Next internal node to be inserted
char nextIntNode = internalNodes.charAt(1);
//counter of generations
int counterGen = 0;


//Constructor
public DynamicTree14()
{
   super(new GridLayout(1, 0));

   rootNode = new DefaultMutableTreeNode(seed);
   treeModel = new DefaultTreeModel(rootNode);

   tree = new JTree(treeModel);
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);

   JScrollPane scrollPane = new JScrollPane(tree);
   add(scrollPane);
 }


 /** Selected tree is unveiled */
 public String unveilTree()
 {
   DefaultMutableTreeNode selectedNode = null;
   TreePath selectedPath = tree.getSelectionPath();

   if (selectedPath == null) {
    selectedNode = rootNode;
   } else {
    selectedNode = (DefaultMutableTreeNode)
            (selectedPath.getLastPathComponent());
```

```
    }

    String tree =  selectedNode.toString();
    if (print)
    System.out.println("\nSelected tree at unveilTree() = "
                        + tree);
    return tree;
  }


  //Add a node
  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }



 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   if (print)
```

```
 System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();
   indexOfTopologies = 0;

while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
if (print)
System.out.println("ParentTree = " +  parentTree );

//=================

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
```

```
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
       + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 if ((4+counterGen)== numberOfLeaves)
 {

  //Lengths are added and evolved to find the
  //best fitting of this topology
  //to target distance matrix.
  //Returned value: the distance between
  //target matrix and the matrix associated to
  //the best fitting of this topology.

  double distance = evolveLengths(newTree);
  bestFitting[indexOfTopologies] = distance;

  System.out.println("\n====================" +
   "\nIndex + Topology + " +
   "distance to target + best tree: \n"
  + indexOfTopologies
   +  ":  "  +
  newTree + " -> "   + simplify(newTree)
 + " " +  distance  + "\n" + bestTree + "\n");
```

```
  bestTreeOfTopology[indexOfTopologies]
                      = bestTree;

 if (print5)
 {
//Distance from M(bestTree)
 //to target matrix is evaluated
 double M[][] = distanceMatrix(bestTree);
  if (print5)
{
 printMatrix(M,
 "Distance Matrix associated to leaves"
 + " of  tree");
}
  double d= absDistance(numberOfLeaves,
                  targetMatrix,
M);

System.out.println("Dist to target = " + d);
 }
  indexOfTopologies++;
 }

String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
```

```
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }

    public void treeNodesRemoved(TreeModelEvent e)
    {
    }

    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree14
```

```
//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.

//Resultant trees are drawn.




class DrawBiTree11 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane31;
JScrollPane scroller;




//=====================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;
```

```
//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;



  String  NwTree = "";



 boolean print = false;



//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];



//Set up the instructions.
JLabel instruct1;
JLabel instruct2;

public DrawBiTree11()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));
```

```
//Set up the instructions.
instruct1 = new JLabel(title1);
instruct2 = new JLabel(title2);

instructionPanel.setFocusable(true);
instructionPanel.add(instruct1);
instructionPanel.add(instruct2);

//Set up the drawing area.
drawingPane31 = new drawingPane31();
drawingPane31.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane31);
scroller.setPreferredSize(new Dimension(500,900));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//========================================
//============= Painting class ============
//========================================


/** The component inside the scroll pane. */
 class drawingPane31 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
```

```
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
```

```
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,12*verticalSize);
//area is set as the preferred size of the graphic
drawingPane31.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane31.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 600;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}



//Tree is drawn
private void DrawTree3(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree3=  "
                              + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
```

```
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
```

```
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
```

```
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree3(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane31



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame =
   new JFrame(titleFrame);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
```

```
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree11();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree11

}//End of main class J173 EvoTree4
```

174 ***Exercise.*** *Run the program and play with the code. Play enough to agree else disagree with the Author: the performance of the code continues to be very low because running time diverges towards infinite pass 5 leaves.*

175 ***Let us stop here.*** *Our previous program is intended to find the exact solution of the problem of finding the best fitting of a tree to a distance matrix. To make sure that our program is correct a test was run: to begin with a tree that gives rise to a distance matrix, the target matrix, that must be fitted in the best way by a tree. The resultant tree gives rise to another matrix that must coincide with the target one. We consider that the problem is solved for a number of leaves 4 and 5. But due to the exaggerated number of generations needed to deal with 6 leaves, beyond 50000 with a population size of 500, this case and all ensuing ones are considered by us as insolvable by our program when it is run over a desktop. If you have enough patience you might reach a solution for seven leaves. And if your patience is as that of a holy person, you eventually might arrive to a mark with 9 leaves. So this is our conclusion: evolution is a wonderful idea that can be used to solve the most varied type of problems specially if it receives some helping hints but it might become useless due to the exaggerated amount of resources, time and memory, that it consumes.*

5.4 Graduation

To graduate from this volume, hit the next challenge:

176 ***Graduation.*** *Develop a program that reads from a folder the proteomes of a given set of species, with some 5 members, calculates a matrix of n-gram distances*

and finds the tree that best fit that matrix. Help: The last task is solved by the previous program and the first two by program I116 EuclideanDistMatrix2 of Vol IX.

5.5 Conclusion

We lack at present an universally accepted definition of species. So, a tentative definition of species for sexual beings is the following: it is a maximal group of living beings that can mate and reproduce. Or, constructively, begin with a family and join every specimen that eventually would produce progeny if it mated with a member in that family. The resultant group is a species. This definition is evolutionary: if mating is impossible in within members of two groups or if it does not lead to reproduction, to the appearing of viable offspring, then the two groups will diverge in time. Divergence will eventually lead to morphological, biochemical and ethological separation. One can theoretically load this definition with Evolutionism as follows: one can assume by dogma that all species arouse by this mechanism and interpret the world accordingly. Else, one can realize that evolution is a reality and so to produce functional complexity it must pass over too many trials that must be evidenced by malfunctions and malformations as in the fossil record as in extant populations. Since we lack such a witness, evolution is not the explanation of the existence of most species. So, the Evolutionary Theory is obviously false. This is valid for Darwinian evolution, in which mutation is at random, as well as for Lamarckian evolution, when it has been furnished with powerful hints that twist randomness.

Nevertheless, there should be no surprise if evolution turns to be the correct explanation of the origin of many groups of species, say, of flies. The reason is that crossed infertility could be the result of a mutation in just a single gen. Therefore, the following problem makes sense in our community: can we forward a particular evolutionary theory in the form of an evolutionary tree to propose an evolutionary interrelatedness of a given group of species that live just now? We learned how to restate this problem in terms of distance matrices:

The distance matrix to tree problem: given a matrix of pairwise n-gram distances among objects that may represent living species, find the unrooted tree that best fits it.

Because evolutionary trees are iconic to evolution, this problem must be solved by evolution. Thus, we devised a program that uses evolution to solve this problem. To make sure that our program is correct a test was run: to begin with a tree that

gives rise to a distance matrix, the target matrix, that must be fitted in the best way by a tree. The resultant tree gives rise to another matrix that must coincide with the target one. We consider that the problem is perfectly solved for a number of leaves 4 and 5. But due to the exaggerated number of generations needed to deal with 6 leaves, beyond 50000 with a population size of 500, this case and all ensuing ones are considered by us as insolvable by our program when it is run over a desktop. Our experience allows us to conclude: evolution is a wonderful idea that can be used to solve the most varied types of problems specially if it receives some helping hints but it might become useless due to the exaggerated amount of resources, time and memory, that it consumes.

As never before, I have experienced while I was working in this chapter that Java gives power to make dreams come true and that no-Java allows time to dream.

Chapter 6

Evolution as a very expensive craftsmanship

Our final aim was to use evolution to find the best fitting tree to a matrix whose entries represent distances between pairs of species. After too much preparative work we at last were able to attack our problem. Summarizing our experience we can say that evolution is a wonderful idea but it may turn useless by some problems, as this one, because demanded running time and memory might rapidly diverge beyond realistic constrains. Else one might get content with rather uncooked products. Else one must invent extremely powerful hints to help evolution. Or one must carefully choose where to apply evolution so that it could look as a diamond in a golden ring.

6.1 The mutation model

Mutation at random at each DNA(RNA) basis is an inherent property of life on Earth. So, we contend that the null hypothesis of science in regard with the origin of species is captured by the **mutation-model**, in which random mutation is all to life, including every fold of morphological or ethological variability and divergence. In effect, our mutation-model explains everything by a strike of luck, be something that happened or something that has not been observed. This model is simple and laconic.

6.2 The function paradigm

We live in a social world in which many things happen not by strikes of luck but by necessity, at least during some period of time and under some circumstances. Say, we love the employment system in which one can earn a life without fluctuating troubles. Therefore, it is normal to look after other models in which usual facts are explained by necessity and exceptions by randomness.

This implies that we must leave away the mutation model and accept another stand. In that regard, nature has forced us to consider another world view to which we are extremely used to:

Eyes are for seeing and ears for hearing.

Thus, every one is inborn with the **function paradigm**: structures may have functions that serve a given purpose. We see that the (intuitive) association of functions to structures is biologically given and so it is inherent to every human culture. In spite of how obvious and natural this paradigm might sound, it is indeed a metaphysical construction. It is the same that happens with gravity: to explain why bodies fall towards the Earth, one must invoke randomness or one might invent gravity or at least believe that it exists. Nevertheless, gravity is not part of direct observations. The same happens with functions: they cannot be weighted or caught. It is through the predictions of formulated models that one can test beliefs. In regard with gravity, one can predict that a stone and a feather fall in vacuum with the same acceleration. In regard with function, one must design devices to test whether or not postulated functions correctly fit performance.

177 ***On the importance of the function paradigm***

Our new paradigm is interesting because it explains at once why like begets like: the ecstasy of species is an expected fact in the function-paradigm because things to be really functional need to be very complex and this usually means that changes, even tiny, are lethal to the function.

This paradigm also faces a challenge that tolerates no delay and this is what makes it very interesting:

From the perspective of the mutation model the origin of life and of species would offer no problem. But these become very terrible puzzles for fans of the function paradigm: given that ecstasy is the rule, where do so huge levels of diversity, complexity, perfection and adaptation come from?

178 ***Evolution as a necessity***

If science is loaded with the functional paradigm over the background of mutation, then evolution in nature is a necessity that unfolds as the result of the interaction of mutation and complexity. So, evolution is the change in genetic content due to the surviving of allowed mutations. Everyone on Earth accepts this as a fact.

179 ***The evolutionary theory in a nutshell***

The incredible similarity among the fish, the pig and the human being is a sufficient reason to propose the Evolutionary Theory over the background of the function paradigm: the expected output of the interaction of mutation and complexity is that ecstasy is what we must observe over some few generations while shifts of structure and function is what we must see over geological times. This theory explains immediately two important questions: the origin of complexity and the origin of species.

180 ***The origin of complexity***

The answer proposed by the theory to explain complexity is extremely simple: mutants have diverse functional performance and so, some specimens leave more offspring that others. With the cumulation of tiny changes generations over generations functions are improved and perfection appears.

181 ***The origin of species***

We lack at present an universally accepted definition of species. Anyway, a popular definition of species for sexual beings is as the following: it is a maximal group of living beings that can mate and reproduce. Or, constructively, begin with a family and join every specimen that eventually would produce progeny if it mated with a member in that family. The resultant group is a species. This definition is evolutionary: if mating is impossible in within members of two groups or if it does not lead to reproduction, to the appearing of viable offspring, then the two groups will diverge in time. Divergence will eventually lead to morphological, biochemical and ethological separation. One can theoretically load this discussion with Evolutionism as follows: one can assume by dogma that evolution is a fairy tale and that all species arouse by this mechanism and interpret the world accordingly. This way of being is not welcome in our community:

Theories in science are proposals to falsify: evolution is a reality, a crude reality and so, mandatory predictions about it must be formulated to test whether or not they are observed. What do we have here? A small detour would help us:

182 ***Evolutionary view over complexity***

We have used the word complexity as a term of the general language: something is complex when it is difficult to construct, difficult to understand, difficult to describe. Functional perfection is usually accompanied with high complexity. Let us make now an attempt to be more formal. To that aim, let us consider an example:

We all know how to make a flying craft taking as raw material a sheet of paper. But only until some days ago experts were able of making artificial flying crafts that resemble insects. Now, what is the connection of our papered flying objects and those wonders? They are connected by a human driven evolutionary process that is full in false starts and blind alleys. That is the signature of complexity. So,

Complexity is the property of every evolutionary process of getting trapped by achieved solutions that are far from the optimum or even from an acceptable solution.

Our discussion has been extremely theoretic, in the air. We need to add a year of reality to our ideas.

183 ***Experimenting with evolution***

The correct understanding of evolution leads immediately to the conception of the best possible lab to study evolution. Our insight is very natural and captures at once the idea that complexity is the air to be breathed in biology:

The genome is software and evolution is a software developer.

In fact, the primary function of the genome is to encode for verbal instructions for the continuation of life. As we all know, the development of software is amidst the most complex tasks in the world. Nevertheless, we also know that the capability to design software is not a privative skill of man. Instead, we have made it clear in Vol II that evolution, natural is also a software developer.

We can formulate now a mandatory prediction, the zero-law of software development:

There is no software without huge amounts of bugs whose correction generate more bugs.

Is this mandatory prediction observed by natural evolution? The problem is that every piece of software developed by human beings has a purpose but evolution has no purpose at all. So, the word purpose makes no sense. Therefore, a question arises: How are we going to apply to evolution the results of our experience with purposefully development of software? The answer is:

Evolution has an emergent and sophisticated aim: to perfect rudimentary functions, specially those that serve the purpose of reproduction and surviving. The mechanism of emergent perfection is the very essence of evolution: if by mere randomness a structure appears that better fits a function, then the possibility exists that it could leave more offspring than its brethren.

That is why the experimental generalization about bugs can be immediately applied to natural evolution: were evolution the cause of the existence of species, the fossil record and modern populations should be plagued with imperfections and malfunctions. Since this is not found, the belief in the origin of most species by Darwinian evolution is just a flaw created by the evolutionary dogma, i.e., by the fear to think when thinking brings light.

Nevertheless, it is not excluded that expected imperfections and malfunctions are not found in some concrete cases because the probability of appearing + registering them might be too low. This may happen when speciation is due to simple changes, say, when it is reduced to the creation of sexual barriers, which can appear after a change in a single gene:

Reproductive isolation may hang on a simple locus, as it happens with bindin, a protein mediating recognition and binding between the sperm and the egg once the sperm has penetrated the egg jelly. Recognizing might be species specific. A high rate of evolution of bindin, i.e., of the creation of sexual barriers, has been found in many species of sea urchin that occupy the same region (Palumbi et al, [6] 2005).

184 ***The style of the genome***

Software development is extremely heavy but there are forms as it could be made still harder: when that happens one says that one is producing not evolvable software. Actually, one dreams of evolvable software as one that can be easily modified with easy to correct bugs. A concrete example as one can produce non evolvable software is very common:

Let us suppose that there are two tasks. The first one is to draw a bush, the second to add names to predefined nodes. If one tries to solve the two problems at the same time with intertwining code the result is an intractable, unmodifiable

mesh. The solution is to resort to the function paradigm of software: problems must be divided in simple tasks that must be solved by exclusive methods. Or: one simple task is solved by one exclusive method with specified input and only one output and without global variables (see Vol II and V).

So, the invitation is to face the posed problem with an approach in which drawing the bush and drawing the names of nodes are treated as two separated problems that must be solved by two separated methods.

Let us turn now to the genome. We know that it is software. The astonishing and permanent advances in genetic engineering clearly show that it is evolvable. We also know that evolution is a software developer. Is the evolvability of the genome the very reason of considering evolution as a possibly correct explanation of the existence of many species? Certainly not: evolvability is not needed for surviving or reproduction. Unless you filter genomes for evolvability, evolution will produce messy software whose evolvability would be extremely limited, in general. That is another sufficient reason because of which the evolutionary theory of the origin of species is obviously false. Nevertheless, an objection may appear:

No one has excluded the possibility that evolvable software appeared by mere randomness and of course it immediately became the norm because of its highly productive rates. This impressive event would amount to the origin of life.

Our criticism to this objection is as follows: to say that randomness is all to life is simpler and under this vision the expression evolvable software is unnecessary.

But if the evolutionary theory is false, how are we going to explain our existence? One might resort to one of the many versions of creationism. These insights are paradigms in their own and in general are outside the scope of science. Nevertheless, a doctrine to be accepted by people must have some parts that agree with common sense and that one can see in our daily life. These parts are subjected to the scrutiny of science. For instance: many forms of creationism assure that the Creator is smart. Now, intelligence is a fundamental quality that we permanently use in our ordinary life. So, a question immediately arises: how are we going to detect intelligence in the genome, if any? The problem with this question is that it is ill: evolution is a reality and it classifies as software developer. So, evolution is intelligent. Therefore, the question to be correct must take a form like this: How are we going to detect a sign of intelligence in the genome that cannot be explained by natural evolution?

Our proposal is to extrapolate to biology what we see here in our work:

We faced the problem of making a drawing with the correct up-down polarity. But we had a problem: the polarity inherited by the code from natural numbers is opposite to that of the display when it is looked by a human being. The solution is very simple: we need a reflection, to upright the polarity, and then a translation to shift the figure appropriately. This amounts to coherent changes in the code that a posteriori can suggest mathematical guidance and an intelligent creator. Such a suggestion to get some interest must be reinforced by many similar cases. In particular, we recall that we have used in volume V some sophisticated techniques of numerical analysis to enrich our programs and simulations. Would anyone dare to claim that evolution is capable of doing things like that and in the same style?

On the other hand, let us argue that the genome is obviously filled in signs of extremely high intelligence. Proof: What is the genome? It is a library of genes, of modularized groups of instructions to synthesize poly-peptide chains to be used in the repairing and construction of amazing ultra-complex and supra-intelligent buildings, such as any living cell is. Now, the genome is an awesome example of a great project that is divided in simple, realizable tasks. According to our experience, the art of divide to conquer demands high intelligence joined to huge amounts of work -i.e., of bugs- a lot of experience, preparation and tenacity.

How are we going to explain that sign of extreme intelligence?

The mutation model is a direct explanation to such a wonder. But it is incredible. So, if you want to explain it with a more credible theory, say, through evolution, if would suffice to exhibit a self organized path from a messy world in which there is no function to an ordered world in which great functions are subdivided into sets of simpler ones and are encoded by (mostly) exclusive genes. If you achieve this, pay attention, over all things, to the almost infinite number of hindrances that you should overcome at every day.

6.3 Java trees

Java comes with an inbuilt service for the management of trees. It is powerful and delightful. It allows to see the clean correspondence between evolution and trees based on the parent & offspring pairing. Besides, one also has the opportunity to devise own formalisms to deal with trees and the Author made a very transparent implementation of binary trees.

Binary trees are those in which each node can have two children at maximum. They are simple and we have managed to keep our own Java implementations quite

simple. In fact, we encoded binary trees in a vector, even entries for nodes and odd for names or contents. Each node is described by its address in the tree which is a binary string, say 1001, in which a zero means going to the left in the tree and a one is an instruction to go to the right. It has been straightforward to simulate evolution along a mutation-model that includes mutation, recombination and sexual dimorphism. Because of simplicity and transparency, we were encouraged to use our binary trees in the sequel.

We also had the opportunity to see, in hindsight, that situations may arise in which various tasks in software development must be solved and it would be desirable to unify them into a single class. Unification works are of great esteem and may be taken as distinctive of both a great intelligence and a exquisite style. From outside, what one sees is that a program makes various tasks that are executed by a code that is rich in regulating instructions (if this and this happens do this else if ...).

Now, it is very astonishing that even the most primitive bacteria are full in regulating instructions. Can natural evolution create things like that?

Yes, possibly it can. But a price must be paid: huge amounts of bugs. So, if somebody wants the Evolutionary Theory to become a part of science, he or she must accompany his or her claims with reports of relevant malformations and malfunctions in living populations, which must abound due to regulating troubles. And forget not to prove that these malformations are not the result of degeneration but that they show that evolution is an ongoing process as it rises along the complexity ladder.

With a Java machinery sufficiently strong, we were ready to take over a very important for us problem:

6.4 Tracing the ancestor

We have relied on the nearest neighbor's principle to propose an algorithm to unveil in line the ancestors of a given node in an evolutionary tree. That principle reads: my father is that being of the just past generation to whom I am the most like. To measure differences between texts we used the n-gram technology that we worked thoroughly in Vol IX. Our study was restricted to binary trees. We found that our principle works nicely under the mutation-model (all mutations are viable and therefore randomness is more than enough to explain evolution) when the mutation rate per site per generation is small to moderate (0.1 or less).

So, we are done with the mutation model. But, what about the function paradigm?

185 ***Language as a model.***

An extremely dare approach to the function paradigm would be to research AL, artificial life, which for us means the simulation in Java of complete themes of life to study precisely how it unfolds under given definition and circumstances. Another also extreme approach is genetic programming which is the use of evolution to design software. We have not touched on AL but we made in Vol II some experiments on genetic programming and concluded that it is not a problem to be solved with a desktop.

In general what we have done is more related to models, to parables, to abstractions. Our present problem is another occasion to show that models are good because they allow us to make a preliminary analysis of ideas under study.

To test the power of the n-gram distance + the nearest neighbor's principle under the function paradigm, we propose to use languages as a natural testbed. The reason is that a human language is a programming language that conveys software for humans. Every text written by a human being transmits some information, a message, a meaning, instructions to do something. So, it is a very natural model of the genome. We can simulate an evolutionary process under the function paradigm if one poses as seed a phrase with meaning. Next, for the nodes of the next generation one adds variations by meaning. And so on. The result is evolutionary because it was actually an evolutionary process that was guided by semantic association in the head of the Author.

We found that evolution under the function paradigm breaks in general the nearest neighbor's principle. The reason is that evolution under the function paradigm might include changes in one generation that represent sufficiently large n-gram distances that breaks the idea that my father is that man that is closest to me in n-gram space. So, what is common under the mutation model is in general not viable under the function paradigm and vice verse, what is common under the function paradigm can be attained under the mutation model but only as and extremely improbable coincidence.

Nevertheless, we found enough evidence to consider that our technology is really excellent to trace back ancestors in a genetic cladistic world, one in which evolution goes from simplicity to complexity by adding more and more functional characters encoded by genes.

6.5 Phylogenetic trees

Although the Evolutionary Theory is in general false, there should be no surprise if evolution turns to be the correct explanation of the origin of many groups of

species, say, of flies. The reason is that a sexual separation may result after a mutation in just a single gen. Therefore, the following problem makes sense in our community: can we forward a particular evolutionary theory in the form of an evolutionary tree to propose an evolutionary interrelatedness of a given group of species that live just now?

To attack this problem, we make an approximation: we consider speciation as an evolutionary event that is characterized by the appearing of a sexual barrier. We imagine that such events are rare and so they are fairly represented by binary trees. Our trees are unrooted, i.e., they hypothesize splitting events that lack a direction of time: we can propose that two nodes are related by descent but we will be unable to claim which node is the ancestor and which the descendant. Next, we learned how to restate this problem in terms of distance matrices:

The distance matrix to tree problem: given a matrix of pairwise n-gram distances among objects that may represent living species, find the unrooted tree that best fits it. The criterion for fitting is as follows: the fitting tree gives rise to a distance matrix, the fitting matrix. Now, the distance between these two matrices, the fitting and the target, must be a minimum.

Evolutionary trees are iconic to evolution because a phylogenetic trees are actually particular evolutionary theories. Therefore, this problem must be solved by evolution. Thus, we devised a program that uses evolution to solve it. To make sure that our program is correct a test was run: if we begin with a target matrix associated to a tree and find the best fitting tree to that matrix and calculate the associated distance matrix, the two matrices must coincide.

We consider that our software solves perfectly the problem for a number of leaves 4 and 5. But due to the exaggerated number of generations needed to deal with 6 leaves, beyond 50000 with a population size of 500, this case and all ensuing ones are considered by us as insolvable by our program when it is run over a desktop.

Our experience allows us to conclude: evolution is a wonderful idea that can be used to solve the most varied types of problems specially if it receives some helping hints but it might become useless in some cases, such as this, due to the exaggerated amount of resources, time and memory, that it consumes. And in regard with Java, it gives power to make dreams come true while no-Java allows time to dream.

Answers to exercises

Problems of Chapter 1

6, page 7. The Author agrees in general terms with all aforementioned claims. Nevertheless, those persons that like rigor would have some complaints that are much better treated with mathematics.

12, page 11. Cornered model: he finds coins in his way to school. This rarely happens but he is being hit by a strike of luck. This model is cornered because it relies on randomness and very low probabilities. This model cannot be falsified: if today you accompany him in his way to school and no coin is found, no protest can be expressed because this is perfectly allowed by chance. Honorable model: his girlfriend presses him to receive money from her because she has a lot and she loves him. This model is honorable because his welfare is made into a necessity while random fluctuations explain those few moments when he has no money. This model incites further investigation: the girlfriend will be interrogated this very Saturday.

27, page 20. The code that adds a daughter to John follows:

```
//Program J27 JohnSons2
//We learn to draw a tree that represents
//father John and his four sons,
// a daughter included.

import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
```

```
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

public class JohnSons2
{
  public static void main(String args[])
  {
    JFrame frame = new JFrame("The sons of John");
    //A node is declared
    DefaultMutableTreeNode John = new
     DefaultMutableTreeNode("John is the father");
    //John is made into a root of the tree
    JTree tree = new JTree(John);

    //The sons of John
    DefaultMutableTreeNode Peter = new
     DefaultMutableTreeNode("Peter is son 1");
    //The node Peter is grafted to node John.
    John.add(Peter);
    DefaultMutableTreeNode Paul = new
     DefaultMutableTreeNode("Paul is son 2");
    John.add(Paul);
    DefaultMutableTreeNode James = new
     DefaultMutableTreeNode("James is son 3");
    John.add(James);
    //Node Mary
    DefaultMutableTreeNode Mary = new
     DefaultMutableTreeNode("Mary is daughter 1");
    //Node Mary is grafted to John
    John.add(Mary);

    //Set children visible
    tree.expandPath(new TreePath(John.getPath()));

    JScrollPane scrollPane = new JScrollPane(tree);
    frame.getContentPane().
                  add(scrollPane, BorderLayout.CENTER);
    frame.setSize(300, 150);
```

```
    frame.setVisible(true);
  }
}//End of main class J27 JohnSons2
```

51, page 73. We recombine appropriately the code of the previous program, `J46 leavesOfTree`, with that of `J4 OnlyOneSpecies`.

```
//Program J51 RecordingTree
//A very simple evolutionary process is
//recorded into a tree.
//Mutation model = just reproduction + mutation


import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;



public class RecordingTree extends JPanel
                           implements ActionListener
{
```

```
private static final long serialVersionUID = 1L;


//Commands for the buttons
private static String RESET_COMMAND = "Reset";



private static   String   seed =
   "evolution is recorded into a tree";

// Definition of the alphabet
private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
//How many symbols in the alphabet
private static  int lengthAlf   = Alphabet.length();

private static Random r = new Random();
//Mutation rate.
private static double m = 0.1;
//Max number of children per leaf.
//The actual number of children is
//a random number in within 1 and maxNumbChildren
private static int maxNumbChildren = 4;

 //DynamicTree is an inner service class that comes
 //at the end of this file.
 private static DynamicTree3 treePanel;



 //Constructor of main class: GUI basics
 public RecordingTree()
 {
   super(new BorderLayout());

   // Create the components.
   treePanel = new DynamicTree3();
```

```
  //The tree is cleared out.
  JButton clearButton = new JButton("Reset");
  clearButton.setActionCommand(RESET_COMMAND);
  clearButton.addActionListener(this);

  // Lay everything out.
  treePanel.setPreferredSize(new Dimension(1200, 800));
  add(treePanel, BorderLayout.CENTER);

  //Lay all four buttons in line
  JPanel panel = new JPanel(new GridLayout(1, 6));

  //Add buttons to the panel

  panel.add(clearButton);
  add(panel, BorderLayout.SOUTH);
}//End of constructor



public void actionPerformed(ActionEvent e) {
  String command = e.getActionCommand();

  if  (RESET_COMMAND.equals(command)) {
    // Clear button clicked.
    treePanel.clear();
    evolution();
  } //else if (Random_Change_COMMAND.equals(command)) {
   //A random change is made: add, delete, clear.
   // treePanel.clear();
 // }
  report();
}


//Data about the tree is printed to the console
```

```
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame = new JFrame("RecordingTree");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  RecordingTree newContentPane = new RecordingTree();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }
```

```
//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
if (p<m)  //mutation
   { //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
   char c = Alphabet.charAt(l);
   mutant = mutant + c;
  }
 else
  { //no mutation
 mutant = mutant + individual.charAt(i);
  }
}
return mutant;
}

  //An evolutionary process under the
  //mutation-model is triggered.
  //Every leaf (node of the last generation) is subjected
  //to reproduction with a number of children
  //equal or less than 4.
  //Reproduction may undergo mutation in each site.
  private static void evolution()
  {

//Evolution is run for some generations
int nGen = 3;
for(int gen = 0; gen < nGen; gen++)
 {
```

```
  System.out.println("gen = " + gen);
treePanel.reproductionOfLeaves();
//individual = mutation(individual);
 //System.out.println(individual);

   }
  }


  public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        createAndShowGUI();
        evolution();
      }
    });
  }


//==========================================================
//==================Inner service class=====================
//==========================================================


//This program is a modification to the program with
//equal name downloaded from

/*
 * http://www.java2s.com/
 * Tutorial/Java/0240__Swing/DynamicTree.htm
 *
 * Cited 28/VI/2013.
 *
```

```
* Modifications made by JR of the Evoljava Community.
* Modifications have unrestricted license.
*
* License of the original source code:
*
* Copyright (c) 1995 - 2008 Sun Microsystems, Inc.
* All rights reserved.
*
* Redistribution and use in source and binary forms,
* with or without
* modification, are permitted provided that
* the following conditions
* are met:
*
*   - Redistributions of source code must retain
*     the above copyright
*     notice, this list of conditions and the
*     following disclaimer.
*
*   - Redistributions in binary form must reproduce
*     the above copyright
*     notice, this list of conditions and the
*     following disclaimer in the
*     documentation and/or other materials
*     provided with the distribution.
*
*   - Neither the name of Sun Microsystems nor
*     the names of its
*     contributors may be used to endorse or
*     promote products derived
*     from this software without specific
*     prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
* HOLDERS AND CONTRIBUTORS "AS
* IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
* INCLUDING, BUT NOT LIMITED TO,
* THE IMPLIED WARRANTIES OF MERCHANTABILITY
```

```
 * AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

// package components;
/*
 * This code is based on an example provided
 * by Richard Stanford, a tutorial
 * reader.
 */
class DynamicTree3 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 Random r = new Random();

 //Constructor
 public DynamicTree3()
 {
```

```
  super(new GridLayout(1, 0));

  rootNode = new DefaultMutableTreeNode(seed);
  treeModel = new DefaultTreeModel(rootNode);

  tree = new JTree(treeModel);
  tree.setEditable(true);
  tree.getSelectionModel().setSelectionMode(
      TreeSelectionModel.SINGLE_TREE_SELECTION);
  tree.setShowsRootHandles(true);

  JScrollPane scrollPane = new JScrollPane(tree);
  add(scrollPane);
}


/** Remove all nodes except the root node. */
public void clear() {
  rootNode.removeAllChildren();
  treeModel.reload();
}


/** Add child to  node selected at random. */
public DefaultMutableTreeNode
             addObjectAtRandom(Object child) {
  DefaultMutableTreeNode parentNode = null;
  Dimension d = tree.getPreferredScrollableViewportSize();
  int i = r.nextInt(d.width);
  TreePath parentPath =
   tree.getClosestPathForLocation(50, i);

  if (parentPath == null) {
    parentNode = rootNode;
  } else {
    parentNode = (DefaultMutableTreeNode)
             (parentPath.getLastPathComponent());
  }
```

```
    return addObject(parentNode, child, true);
  }

  /** Add child to the currently selected node. */
  public DefaultMutableTreeNode addObject(Object child) {
    DefaultMutableTreeNode parentNode = null;
    TreePath parentPath = tree.getSelectionPath();

    if (parentPath == null) {
      parentNode = rootNode;
    } else {
      parentNode = (DefaultMutableTreeNode)
             (parentPath.getLastPathComponent());
    }

    return addObject(parentNode, child, true);
  }



  public DefaultMutableTreeNode
                addObject(DefaultMutableTreeNode parent,
      Object child) {
    return addObject(parent, child, true);
  }



  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                      new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }
```

```
    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
                (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
DefaultMutableTreeNode node = null;
Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();
  System.out.println(node );
  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
//The number of children is within
//1 and maxNumbChildren;
    int actualNumberOfChildren =
     r.nextInt(maxNumbChildren) +1;
    //System.out.println(actualNumberOfChildren) ;
    //Leaf undergoes reproduction
    for(int i = 1; i <= actualNumberOfChildren; i++)
     {
```

```
     String nodeString = node.toString();
     String child = mutation(nodeString);
     addObject(node, child, true);

     }

//System.out.println(node.toString()) ;
   }

 }
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener {
    public void treeNodesChanged(TreeModelEvent e) {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }
```

```
    public void treeNodesInserted(TreeModelEvent e) {
    }

    public void treeNodesRemoved(TreeModelEvent e) {
    }

    public void treeStructureChanged(TreeModelEvent e) {
    }
  }
}

}//End of main J51 class RecordingTree
```

56, page 76. Java trees can represent binary trees:

```
//Program J56 VectorToBinTree
/*
 * Vector is a class in Java whose instances
represent  ordered n-tuples of whatever nature.
Vectors can represent the pairing
father-offspring if the first entry of the vector
is the name of the father and
the second entry is a linear array with the sons.
The virtue of this representation is that one can
add sons very easily.

Let us see how to use vectors to
input the structure of a BINARY tree.
A binary three is one in which
each father can have at most 2 children.
 */

import java.awt.BorderLayout;
import java.util.Vector;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
```

```
import javax.swing.JTree;


//This inner class represents the
//father-offspring pairing
class PairingVector2<E> extends Vector<E>
 {

  private static final long serialVersionUID = 1L;
  //The name of the father
  String name;

  //Constructor1: just the father
  PairingVector2(String name)
   {
    this.name = name;
   }

  //Constructor2: a father  plus sons
  PairingVector2(String name, E elements[])
  {
    this.name = name;
    for (int i = 0, n = elements.length; i < n; i++)
    {
      add(elements[i]);
    }
  }

  //Draws automatically  the name of the father
  public String toString()
  {
    return "[" + name + "]";
  }
}

//Main class
public class VectorToBinTree {
  public static void main(final String args[])
```

```
{
  JFrame frame = new JFrame("VectorToTree");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  //Last generation of sons and corresponding fathers
  //Vector<String> allows no grafting
  Vector<String> v4 = new
   PairingVector2<String>("4", new String[]
   { "8","9" });
  Vector<String> v5 = new
   PairingVector2<String>("5", new String[]
   { "10","11" });
  Vector<String> v6 = new
   PairingVector2<String>("6", new String[]
   { "12","13" });
  Vector<String> v7 = new
   PairingVector2<String>("7", new String[]
   { "14","15" });

  //First generation of sons
  //A father<Object> with a name
  Vector<Object> v2 = new PairingVector2<Object>("2");
  Vector<Object> v3 = new PairingVector2<Object>("3");

  //v4 is grafted to v2<Object> = 2
  v2.add(v4);
  v2.add(v5);
  v3.add(v6);
  v3.add(v7);

  //1 has two sons: 2 and 3
  Object rootNodes[] = {v2, v3 };
  //The 1 = grandpa pairing is officially made
  Vector<Object> v1 = new
   PairingVector2<Object>("1", rootNodes);
  //A tree is defined based on the Grandpa pairing
  JTree tree = new JTree(v1);
  //Pane to draw in
```

```
    frame.add(new JScrollPane(tree), BorderLayout.CENTER);
    frame.setSize(300, 500);
    frame.setVisible(true);
  }
}//End of main class J56 VectorToBinTree
```

62, page 82. A program to deal with general binary trees:

```
//Program J62 BinaryInput2
/*
 * The representation of binary trees must be
 * as simple as the trees themselves.
 * This program deals with the following notation:
 * each node has an address given by
 * the binary full expansion of its position in the tree,
 * beginning from 0 and 1, the sons of the root,
 * and ending with 1....1,
 * the second son of the last father.
 *
 * The program receives a tree in this binary notation
 * and outputs a drawing of the tree using
 * the Java Tree class (of package Swing).
 *
 */

import java.awt.BorderLayout;
import java.util.Enumeration;


import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

import javax.swing.tree.TreePath;


//Main class
```

```
public class BinaryInput2
{

 //This is data for a tree in our binary notation:
 //Even entries are the addresses in the tree of the nodes,
 //odd ones are the names.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
"10", "ba",
"11", "bb",
"000", "aaa",
"001", "aab",
"010", "aba",
"011", "abb",
"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

private static int numberOfNodes = Input.length;

private static DefaultMutableTreeNode rootNode =
              new DefaultMutableTreeNode("Root");

    private static DefaultTreeModel treeModel =
                   new DefaultTreeModel(rootNode);

    private static JTree tree = new JTree(treeModel);



 /** Add child to the currently selected node. */
 public DefaultMutableTreeNode addObject(Object child)
```

```
 {
  DefaultMutableTreeNode parentNode = null;
  TreePath parentPath = tree.getSelectionPath();

  if (parentPath == null)
   {
    parentNode = rootNode;
   } else
    {
     parentNode = (DefaultMutableTreeNode)
           (parentPath.getLastPathComponent());
    }

  return addObject(parentNode, child, true);
}



public static DefaultMutableTreeNode
             addObject(DefaultMutableTreeNode parent,
    Object child)
 {
  return addObject(parent, child, true);
 }


public static DefaultMutableTreeNode
       addObject(DefaultMutableTreeNode parent,
          Object child, boolean shouldBeVisible)
 {
  DefaultMutableTreeNode childNode =
             new DefaultMutableTreeNode(child);

 if (parent == null)   parent = rootNode;


 // It is key to invoke this on the TreeModel,
 //and NOT DefaultMutableTreeNode
```

```
  treeModel.insertNodeInto
    (childNode, parent, parent.getChildCount());

  // Make sure the user can see the lovely new node.
  if (shouldBeVisible)
     tree.scrollPathToVisible
       (new TreePath(childNode.getPath()));

  return childNode;
}

 //Generic procedure to search a node with name nodeStr
 public static DefaultMutableTreeNode
              searchNode(String nodeStr)
  {
DefaultMutableTreeNode node = null;
Enumeration<?> e = rootNode.breadthFirstEnumeration();
while (e.hasMoreElements())
{
 node = (DefaultMutableTreeNode) e.nextElement();
 if (nodeStr.equals(node.getUserObject().toString()))
 return node;
}
   return null;
  }

 //ParentAddress is a string that indicates the address
 //in the tree of the parent node. This must be deciphered
 //and the name of the parent must be captured.
 public static String parentName(String parentAddress)
 {
  //The position in Input of parentAddress is looked for
  int parentAddressPosition = 0;
  boolean go = true;
  for(int i = 0;  ((i < numberOfNodes) & (go)); i = i + 2)
{
 if (Input[i].equals(parentAddress))
  {
```

```
 parentAddressPosition = i;
 go = false;
  }
}
 /* System.out.println(parentAddress + " "
    + parentAddressPosition ); */

  return Input[parentAddressPosition + 1];
 }


 //The father of a node of a given address
 //in the tree is returned.
 public static DefaultMutableTreeNode
               parent(String address)
 {
  //The length of address: how many bits
  int l = address.length();
  String parentAddress = "";
  String parentName = "";

  //New node
  DefaultMutableTreeNode dad;
  //If address is not that of a son of the rootNode
  if (l >1)
   {
//The address of the parent is contained in
//the first l-1 bits
parentAddress = address.substring(0, l-1);
//ParentAddress is a string that must be transformed
//into a number that indicates the position of
//parentAddress in the input array.
parentName = parentName(parentAddress);
    dad = searchNode(parentName);
   }
  else
  //If address is that of a son of the rootNode
  dad = rootNode;
```

```
 return dad;
}


  public static void main(final String args[])
  {
    JFrame frame = new JFrame("Binary Tree in binary notation");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Read data and generate a java tree
    for(int i = 0; i < numberOfNodes; i = i + 2)
    {
      //An address and a name are read.
      //These represent a new node that must be added
      //to the tree.
      String address = Input[i];
      String name = Input[i+1];
      //System.out.println(address + " " + name);
      //The parent node must be identified
      DefaultMutableTreeNode pa = parent(address);
      //The new node must be  grafted to its parent
      addObject(pa,name );
    }

   //Pane to draw in
    new JScrollPane(tree);
    frame.add(new JScrollPane(tree), BorderLayout.CENTER);
    frame.setSize(400, 500);
    frame.setVisible(true);
  }
} //End of main J62 class BinaryInput2
```

69, page 92. The up-down polarity of the drawing is as expected:

```
//Program J69 BinaryBushDrawing3
/*
* The representation of binary trees must be
* as simple as the trees themselves.
```

```
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
* and outputs a drawing of the tree using
* our own methods.
*
* The up-down polarity of the drawing is user friendly:
* y -> y - k causes a vertical reflection plus a translation.
*
*/

import javax.swing.*;


import java.awt.*;




public class BinaryBushDrawing3 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane3;


 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Relative length of branches of the first U
```

```
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
"10", "ba",
"11", "bb",
"000", "aaa",
"001", "aab",
"010", "aba",
"011", "abb",
"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb"
};

private static int numberOfNodes = Input.length;
private static int numberOfGenerations =
              Input[numberOfNodes-1].length();

 //Constructor
 public BinaryBushDrawing3()
 {
  super(new BorderLayout());
  //Set up the drawing area.
  drawingPane3 = new DrawingPane3();
  //Put the drawing area in a scroll pane.
  JScrollPane scroller = new JScrollPane(drawingPane3);
  //scroller.setPreferredSize(new Dimension(700,800));
  add(scroller, BorderLayout.CENTER);
```

```
 }



 //The tree is drawn
 public class DrawingPane3 extends JPanel
  {
   private static final long serialVersionUID = 1L;


  /* a horizontal U  is drawn:
   * center is the center of the vertical stroke,
   * H is the length of the Horizontal stroke.
   * V is the half the length of the vertical stroke
   * Up = name of the upper branch
   * Down = name of the lower branch
   *
   */
  private void drawU(Graphics g, Dimension center,
             int H, int V, String Up, String Down)
  {
int x0 =  center.width;
int y0 =  center.height;
int x1 =  x0;
int y1 =  y0 + V;
g.drawLine(x0, verticalSize -y0, x1, verticalSize- y1);
g.drawLine(x1, verticalSize - y1,
   x1 + H,  verticalSize - y1);
//String is drawn
g.drawString(Up, x1 + 1, verticalSize - y1-1);


y1 =  y0 - V;
g.drawLine(x0, verticalSize - y0,
   x1,  verticalSize - y1);
g.drawLine(x1, verticalSize - y1,
   x1 + H,  verticalSize - y1);
//String is drawn
```

```
g.drawString(Down, x1 + 1, verticalSize - y1-1);
  }

  //The binary tree is drawn
  protected void paintComponent(Graphics g)
   {
     super.paintComponent(g);

     g.setColor(Color.red);
     int centerX = 1;
     int centerY = 1;
     //The position of first string in Input[]
     int stringPosition = 1;

     //Draw stem
     int x0 =    scale*centerX;
  int y0 =   scale* centerY + 400;
  int x1 =  x0 + (int) ( scale*horizontalLength);
  int y1 =  y0;
    g.drawLine(x0, verticalSize - y0,
        x1,  verticalSize - y1);

    //Horizontal and basic vertical strokes
    int H = scale * horizontalLength;
  int V = scale * verticalLength;

   //For each generation,
   //the upper U is drawn in first place
     Dimension upperCenter = new Dimension(x1,y1);
     Dimension localCenter = new Dimension(0,0);
     for(int j = 1; j < numberOfGenerations + 1; j++)
     {
      //System.out.println("V = " + V);
      //The number of U's in each generation
      //equal half the number of leaves
     double numberOfLeaves =
        Math.pow(2, j);
     int  numberOfU_s = (int) numberOfLeaves /2;
```

```
      /*
      System.out.println("Number of leaves = " +
       numberOfLeaves);
      System.out.println("Number of U's = " +
       numberOfU_s);
      */
      localCenter.width = upperCenter.width;
      localCenter.height = upperCenter.height;
      //The U's are drawn
      for(int i = 0; i < numberOfU_s; i++)
       {
      //drawU(g,localCenter, H,V );
       String Up = Input[stringPosition];
       //Strings are at the odd position of Input
       stringPosition = stringPosition + 2;
       String Down = Input[stringPosition];
       if (stringPosition < numberOfNodes -4 )
            stringPosition = stringPosition + 2;
       drawU(g,localCenter, H,V,Up, Down  );
      //Center of the U below
         localCenter.height = localCenter.height - 4*V;
       }
      //The center of the upper U of the next generation
      upperCenter.height = upperCenter.height + V;
      upperCenter.width = upperCenter.width + H;
      //Vertical stroke is divided by two
      V = V/2;
      }
   }
}


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing3");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
   //Content pane where to draw
   JComponent newContentPane3 = new BinaryBushDrawing3();
   frame.setContentPane(newContentPane3);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);
 }

public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
  }
}//End of main class J69 BinaryBushDrawing3
```

78, page 116. Improving the style of the code:

```
//Program J78 BinaryBushDrawing7
/*
* The representation of binary trees must be
* as simple as the trees themselves.
* This program deals with the following notation:
* each node has an address given by
* the binary full expansion of its position in the tree,
* beginning from 0 and 1, the sons of the root,
* and ending with 1....1,
* the second son of the last father.
*
* The program receives a complete tree
* in this binary notation
* and outputs a drawing of the tree using
* our own methods.
```

```
* The input must be ordered.
*
* The up-down polarity of the drawing is user friendly:
* y -> y - k causes a vertical reflection plus
* a translation.
*
* The present code
* accepts an input with
* incomplete content. So, the complete tree
* is drawn but some nodes may lack
* corresponding strings.
*
* We rely on a natural interpretation of addresses:
* Address in the tree 0110 means:
* go  up, down, down and up, 4  generations in all.
*
* Code has been made more readable.
*
*/



import javax.swing.*;


import java.awt.*;


public class BinaryBushDrawing7 extends JPanel
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane7;


 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
```

```
 private static int scale = 16;

 //Relative length of branches of the first U
 private static int horizontalLength = 2;
 private static int verticalLength = 8;

//This is data for a tree in our binary notation:
//Even entries are the addresses in the tree of the nodes,
//odd ones are the names or contents.
private static String[] Input =
{
"0", "a",
"1", "b",
"00", "aa",
"01", "ab",
//"10", "ba",
"11", "bb",
//"000", "aaa",
//"001", "aab",
"010", "aba",
//"011", "abb",
//"100", "baa",
"101", "bab",
"110", "bba",
"111", "bbb",
"0000", "aaaa",
"1111", "bbbb",
"11111", "bbbbb"
};

 private static int InputLength = Input.length;
 //The number of generation is
 //the length of the last address
 private static int inputhLength =
              Input[InputLength-1].length();

 private static int numberOfGenerations = 5;
 //The terminal  leaves define the complete tree
```

```
private static int numberOfLeaves =
  (int) Math.pow(2, numberOfGenerations);
//Binary paths to leaves
private static String[] pathsToLeaves =
                    new String[numberOfLeaves];

//Constructor
public BinaryBushDrawing7()
{
 super(new BorderLayout());
 //Set up the drawing area.
 drawingPane7 = new DrawingPane7();
 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane7);
 //scroller.setPreferredSize(new Dimension(700,800));
 add(scroller, BorderLayout.CENTER);
}



//The tree is drawn
public class DrawingPane7 extends JPanel
 {
  private static final long serialVersionUID = 1L;


  //Paths to leaves are constructed as binary strings
  private void createPathsToLeaves(int numberOfGenerations)
  {
String s;
for(Integer i = 0; i < numberOfLeaves; i++ )
{
 s  = Integer.toBinaryString(i);
 int l = s.length();
 if (l < numberOfGenerations)
 for(int k = l; k < numberOfGenerations; k++)
 s = "0" + s;
 pathsToLeaves[i] = s;
```

```
  System.out.println(pathsToLeaves[i]);
 }
   }


  //===== DRAWING MACHINERY ============

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00 =    scale*centerX;
int y00 =   scale* centerY + 400;
//Target coordinates
int x11 =  x00 + (int) ( scale*horizontalLength);
int y11 =  y00;
//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V = scale * verticalLength;
//Addresses are interpreted as paths
String path = "";
char direction;

//Tree is drawn
private void drawTree(Graphics g)
 {
//The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
     for(int i = 0; i < numberOfLeaves; i++)
     {
      //Addresses will be used as paths to  be drawn
      path = pathsToLeaves[i];
      //Reinitialization
      Dimension center = new Dimension(x11,y11);
      V = scale * verticalLength;
       //Path is interpreted
```

```
         for(int j = 0; j < path.length(); j++)
          {
        direction = path.charAt(j);
        int x0 =  center.width;
    int y0 =  center.height;
    int x1 =  x0;
    int y1;
        if (direction == '0') //go up
         {
          y1 =  y0 + V;
          center.height = center.height + V;
         }
        else // if direction == '1'  =  go down
         {
          y1 =  y0 - V;
          center.height = center.height - V;
         } //End of if + else
      //Draw vertical stroke
    g.drawLine(x0, verticalSize -y0, x1,
    verticalSize- y1);
    //Draw horizontal stroke
    g.drawLine(x1, verticalSize - y1,
       x1 + H,  verticalSize - y1);
        //Move center
        center.width = center.width + H ;
        //Vertical stroke is divided in two
        V = V/2;
       }//End of for j
     }  //End of for: numberOfLeaves
 }


//Strings are drawn
private void drawStrings(Graphics g)
{
    //Strings are drawn:
    for(int i = 0; i < InputLength; i = i+2)
     {
```

```
    //Addresses will be used as paths to the places
    //where strings must be drawn
    path = Input[i];
    String s = Input[i+1];
    //Reinitialization
    Dimension center = new Dimension(x00,y00);
    Dimension target = center;
    V = scale * verticalLength;
    //Path is interpreted
    for(int j = 0; j < path.length(); j++)
     {
   direction = path.charAt(j);
   if (direction == '0') //go up
    target.height = target.height + V;
   else // if direction == '1'  =  go down
   target.height = target.height - V;

   target.width = target.width + H ;
   V = V/2;
     }//End of for j
    //String is drawn
   g.drawString(s, target.width + 1,
                  verticalSize - target.height-1);
   }
 }


protected void paintComponent(Graphics g)
 {
   super.paintComponent(g);
   createPathsToLeaves(numberOfGenerations);
   g.setColor(Color.red);
   //Draw stem
  g.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
  drawTree(g);
  drawStrings(g);
 }//End of   paintComponent
```

```
  }//End of inner class DrawingPane7


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryBushDrawing7");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent newContentPane7 = new BinaryBushDrawing7();
   frame.setContentPane(newContentPane7);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);
 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();

     }
   });
 }
} //End of main class J78 BinaryBushDrawing7
```

79, page 116. Our drawing machinery, now based on paths, admits the possibility to simulate reproduction with zero, one or two children. Let us see why: if a node has no children, nothing is needed to say. Just no node will appear as its child. If a node has just one child, just one node will appear as its direct progeny. If a node has two children, two nodes will

appear as offspring.

82, page 126.

```
//Program J82 BinaryEvo2
/*
* We run the mutation-model
* restricted over a binary tree.
* We follow our own encoding rules
* and we use our own drawing machinery.
*
* The code has been made more readable
* and more consistent.
*
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Random;


public class BinaryEvo2 extends JPanel
                            implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane9;

 //Number of reproductive cycles
 private static int numberOfGenerations = 5;

 //Vertical  size of the panel
 private static int verticalSize = 1000;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
```

```
 private static int scale = 16;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 2;
 //for 3 generations, vertical length = 6;
//for 4 generations, vertical length = 10;
//for 5 generations, vertical length = 14;
//for 6 generations, vertical length = 25;
//for 7 generations, vertical length = 48;
 private static int verticalLength = 14;


//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[1000000];
 //Last used position  in Record[]
 private static int lastUsedPosition;


 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
    "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   = Alphabet.length();

 private static Random r = new Random();
 //Mutation rate.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
```

```
//a random number in within 0 and maxNumbChildren
private static int maxNumbChildren = 2;


//Constructor
public BinaryEvo2()
{
 super(new BorderLayout());

 //The tree is cleared out.
 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);
 clearButton.setVisible(true);

 //Set up the drawing area.
 drawingPane9 = new DrawingPane9();
 drawingPane9.add(clearButton);
 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane9);
 //scroller.setPreferredSize(new Dimension(700,800));

 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
  String command = e.getActionCommand();
  //If reset button is clicked over
  if  (RESET_COMMAND.equals(command))
  {
    evolution();
    repaint();
  }
}
```

```
//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
 if (p<m)  //mutation
   { //a random integer number
    int l = r.nextInt(lengthAlf);
    //A random char
    char c = Alphabet.charAt(l);
    mutant = mutant + c;
   }
  else
   { //no mutation
  mutant = mutant + individual.charAt(i);
   }
}
return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;

public static void reproduce(int i)
{
String direction; //0 = up; 1 = down
    int t = r.nextInt(2);
    if ( t == 0 ) direction = "0";
```

```
    else direction = "1";
    childPath = fatherPath + direction;
    lastUsedPosition++;
    Record[lastUsedPosition] = childPath;
    //System.out.println("Path first = " + childPath);
    lastUsedPosition++;
    Record[lastUsedPosition] = mutation(Record[i+1]);
    if (actualNumberOfChildren == 1)
 {
     //If a second child exists,
     //its direction is contrary to that of the first
     if ( t == 0 ) direction = "1";
    else direction = "0";
     childPath = fatherPath + direction;
    //System.out.println("Path second = " + childPath);
    lastUsedPosition++;
    Record[lastUsedPosition] = childPath;
    lastUsedPosition++;
    Record[lastUsedPosition] = mutation(Record[i+1]);
     }
}


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
 {

  //Reproductive leaves are the nodes at the end of Record[]
  //with the max length.
  String lastLeaf = Record[lastUsedPosition-1];
  int maxLength = lastLeaf.length();
for(int i = lastUsedPosition-1; i >= 0; i = i-2)
   {
//Nodes go at even entries
fatherPath = Record[i];
//System.out.println("Nodee = " + i  + "  " + fatherPath);
//If node is a reproductive leaf
```

```
 if( fatherPath.length() == maxLength )
   {
//The number of children is within
//0 and 2;
    actualNumberOfChildren =
     r.nextInt(maxNumbChildren + 1);
    /*
    System.out.println("Number of children = "
      + actualNumberOfChildren) ; */
    //Leaf undergoes reproduction
    if (actualNumberOfChildren > 0)
    {
      reproduce(i);

    }
//System.out.println(node.toString()) ;
   }

    }
  }


   //An evolutionary process under the
   //mutation-model is triggered.
   //Every leaf (node of the last generation) is subjected
   //to reproduction with a number of children
   //equal or less than 2.
   //Reproduction may undergo mutation in each site.
   private static void evolution()
   {
    //Initialization of node 0: path goes up
Record[0] = "0";
//The string of node 0
Record[1] = seed;
//Initialization of node 1: path goes down
    Record[2] = "1";
    //The string of node 1
Record[3] =  mutation(seed);;
```

```
    //Last used position in Record[]
    lastUsedPosition = 3;
  //Evolution is run for some reproductive cycles
//Evolution = the mutation-model = just mutation.
  //Generation 0 was made by hand in the initilization
  for(int gen = 1; gen < numberOfGenerations; gen++)
   {
    //System.out.println("gen = " + gen);
  reproductionOfLeaves();
     }
   }

 //===== Inner class with the drawing machinery =======


 public class DrawingPane9 extends JPanel
  {
   private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00 =    scale*centerX;
int y00 =   scale* centerY + 400;
//Target coordinates
int x11 =  x00 + (int) ( scale*horizontalLength);
int y11 =  y00;
//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V = scale * verticalLength;
//Addresses are interpreted as paths
String path = "";
char direction;

//Tree is drawn
private void drawTree(Graphics g)
 {
```

```
//The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
     for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to  be drawn
      path = Record[i];
      //Reinitialization
      Dimension center = new Dimension(x11,y11);
      V = scale * verticalLength;
       //Path is interpreted
         for(int j = 0; j < path.length(); j++)
          {
        direction = path.charAt(j);
        int x0 =  center.width;
    int y0 =  center.height;
    int x1 =  x0;
    int y1;
        if (direction == '0') //go up
         {
          y1 =  y0 + V;
          center.height = center.height + V;
         }
        else // if direction == '1'  =  go down
         {
          y1 =  y0 - V;
          center.height = center.height - V;
         } //End of if + else
      //Draw vertical stroke
    g.drawLine(x0, verticalSize -y0, x1,
    verticalSize- y1);
    //Draw horizontal stroke
    g.drawLine(x1, verticalSize - y1,
       x1 + H,  verticalSize - y1);
        //Move center
        center.width = center.width + H ;
        //Vertical stroke is divided in two
        V = V/2;
```

```
      }//End of for j
    }  //End of for: numberOfLeaves
 }


//Strings are drawn
private void drawStrings(Graphics g)
{
    //Strings are drawn:
    for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to the places
      //where strings must be drawn
      path = Record[i];
      String s = Record[i+1];
      //Reinitialization
      Dimension center = new Dimension(x00,y00);
      Dimension target = center;
      V = scale * verticalLength;
      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
     g.drawString(s, target.width + 1,
                     verticalSize - target.height-1);
     }
   }
```

```
  protected void paintComponent(Graphics g)
   {
     super.paintComponent(g);
     g.setColor(Color.red);
     //Draw stem
    g.drawLine(x00, verticalSize - y00,
        x11,  verticalSize - y11);
    drawTree(g);
    drawStrings(g);
}//End of   paintComponent
  }


 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryEvo");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent BinaryEvo2 = new BinaryEvo2();
   frame.setContentPane(BinaryEvo2);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);

 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();
      evolution();
```

```
     }
   });
 }
} //End of main class J82 BinaryEvo2
```

87, page 138. Evolution with sexual dimorphism.

```
//Program J87 BinaryEvo4
/*
* We run the full mutation-model  of evolution
* restricted over a binary tree.
* It includes mutation,  recombination,
* a rudimentary sexual dimorphism,
* and sexual reproduction.
*
* We follow our own encoding rules
* and we use our own drawing machinery.
*
* The sign $ is a distinctive of maleness
* and recombination is possible between
* individuals of different sex only.
* Otherwise reproduction = parthenogenesis.
*
* If there is no female, the process stops.
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Random;


public class BinaryEvo4 extends JPanel
                            implements ActionListener
```

```
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane11;

 //Number of reproductive cycles
 private static int numberOfGenerations = 5;

 //Vertical  size of the panel
 private static int verticalSize = 800;
 private static int horizontalSize = 800;
 //Scale must be a power of 2
 private static int scale = 16;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 2;
 //for 3 generations, vertical length = 6;
//for 4 generations, vertical length = 10;
//for 5 generations, vertical length = 14;
//for 6 generations, vertical length = 25;
//for 7 generations, vertical length = 48;
 private static int verticalLength = 14;

 //Maximal number of nodes in the tree
 private static int N = 10000;
//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[2*N];



 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
    "evolution is run and recorded over a binary tree";
```

```
// Definition of the alphabet
private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
//How many symbols in the alphabet
private static  int lengthAlf   = Alphabet.length();

private static Random r = new Random();
//Mutation rate.
private static double m = 0.1;
//Max number of children per leaf.
//The actual number of children is
//a random number in within 0 and maxNumbChildren
private static int maxNumbChildren = 2;


//Constructor
public BinaryEvo4()
{
 super(new BorderLayout());

 //The tree is cleared out.
 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);
 clearButton.setVisible(true);

 //Set up the drawing area.
 drawingPane11 = new DrawingPane11();
 drawingPane11.add(clearButton);
 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane11);
 //scroller.setPreferredSize(new Dimension(700,800));

 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
```

```
public void actionPerformed(ActionEvent e)
{
  String command = e.getActionCommand();
  //If reset button is clicked over
  if  (RESET_COMMAND.equals(command))
  {
    evolution();
    repaint();
  }
}


//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
 if (p<m)  //mutation
   { //a random integer number
    int l = r.nextInt(lengthAlf);
    //A random char
    char c = Alphabet.charAt(l);
    mutant = mutant + c;
   }
  else
   { //no mutation
  mutant = mutant + individual.charAt(i);
   }
}
return mutant;
}
```

```
 //Reproduction

 private static String fatherPath;
 private static String childPath;
 private static   int actualNumberOfChildren;
 //Last used position  in Record[]
 private static int lastUsedPosition;




 //Position of leaves in Record[]
 //Reproductive male leaves (of the last generation)
 private static  int[] maleLeaves = new int[N];
 private static  int[] femaleLeaves = new int[N];
 private static int counterMaleLeaves;
 private static int counterFemaleLeaves;

 //Find leaves and count them.
 //They are classified by gender.
 public static void findLeaves()
 {
  counterMaleLeaves = 0;
  counterFemaleLeaves = 0;
  //Reproductive leaves are the nodes at the end of Record[]
  //with the max length.
  String lastLeaf = Record[lastUsedPosition-1];
  int maxLength = lastLeaf.length();
  for(int i = lastUsedPosition-1; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  fatherPath = Record[i];
  //System.out.println("Leaf = " + Record[i]);
//System.out.println("Node = " + i  + "  " + fatherPath);
  //If node is a reproductive leaf
  if( fatherPath.length() == maxLength )
   {
 //Is it male?
```

```
 String s = Record[i+1];
 if (s.charAt(0) == '$')
  {
maleLeaves[counterMaleLeaves] = i;
counterMaleLeaves++;
//System.out.println("Male =  " + Record[i+1]);
  }
 else
  {
   femaleLeaves[counterFemaleLeaves] = i;
   counterFemaleLeaves++;
   //System.out.println("Female = " + Record[i+1]);
  }
   }

  }
 }


//Flag for printing recombination
private static  boolean print= false;

//Record[i+1], the string associates to node i,
//is recombined
//with another chosen at random leaf.
//Recombination is allowed in the only case
//in which chosen individuals have different sex.
//Otherwise reproduction = Parthenogenesis
public static String recombination(int i)
{
 //Female string to be recombined
String s = Record[i+1];
String  s1= "";

//The recombinant part of s is selected
int recSite = 0;
int m= s.length();
if (m>0)
```

```
{
 recSite = r.nextInt(m) ;
 //Recombinant part of s
 s1= s.substring(0,recSite);
}

String output = "";
//A male partner  is chosen, if it exist
if (counterMaleLeaves > 0)
{
int k ;
String t ;

  k = r.nextInt(counterMaleLeaves);
  t = Record[maleLeaves[k]+1];

//Recombinant part of t
//Bug corrected
String  t2=  t.substring(recSite+1);
//Recombination
output = s1+t2;

if (print)
{
 System.out.println("Recombination");
 System.out.println("recSite = " + recSite);
 System.out.println(s);
 System.out.println(t);
 System.out.println(output);
}
}
else output = s;

return output;
}
```

```
public static void bearChild(int i, String direction)
{

    //data for the first child
    childPath = fatherPath + direction;
    lastUsedPosition++;
    Record[lastUsedPosition] = childPath;
    //System.out.println("Path first = " + childPath);
    //String associated with node i
    String s = Record[i+1];
    //Sex indicator is filtered (if one exists)
    if (s.charAt(0) == '$') s = s.substring(1);
    //The string associated with node i is recombined,
    s = recombination(i);
    //passed under mutation
    s = mutation(s);
    //and recorded
    lastUsedPosition++;

    //Sex is assigned, female by default.
    int sex = r.nextInt(2);
    if (sex == 1 )
     Record[lastUsedPosition] =   "$" + s;
    else
    Record[lastUsedPosition] =    s;
    //System.out.println(childPath + " " +  s);
}

 //Node at position i is reproduced.
 //Reproduction involves mutation and recombination.
 //Children are given a sex at random,
 //$ = male, nothing = female.

 public static void reproduce(int i)
 {
  //First child
  String direction; //0 = up; 1 = down
```

```
 int t = r.nextInt(2);
 if ( t == 0 ) direction = "0";
 else direction = "1";
 bearChild(i, direction);
 //Data for a second child
 if (actualNumberOfChildren == 2)
     {
  //The direction of the second child is
  //contrary to that of the first.
 if ( t == 0 ) direction = "1";
 else direction = "0";
      bearChild(i, direction);
     }
}


//Terminal   leaves, nodes of the last generation,
//are found and children are generated.
//Only females can bear children.
private static void reproductionOfLeaves()
 {
  findLeaves();
for(int i = 0; i < counterFemaleLeaves; i++)
   {
int position = femaleLeaves[i];
//Nodes go at even entries
fatherPath = Record[position];
//System.out.println(Record[position+1]) ;
//The number of children is within
//0 and 2;
actualNumberOfChildren =
    r.nextInt(maxNumbChildren + 1);
   /*
   System.out.println("Number of children = "
     + actualNumberOfChildren) ; */
   //Leaf undergoes reproduction
if (actualNumberOfChildren > 0)
     reproduce(position);
```

```
    }
  }


   //An evolutionary process under the
   //mutation-model is triggered.
   //Every leaf (node of the last generation) is subjected
   //to reproduction with a number of children
   //equal or less than 2.
   //Reproduction may undergo mutation in each site.
   private static void evolution()
   {
    //Initialization of node 0: path goes up
Record[0] = "0";
//The string of node 0 denotes a male: it has $
Record[1] = "$" + mutation(seed);
//Initialization of node 1: path goes down
    Record[2] = "1";
    //The string of node 1 denotes a female:
    //it does not have $
    Record[3] =  mutation(seed);
    //Last used position in Record[]
    lastUsedPosition = 3;
  //Evolution is run for some reproductive cycles
//Evolution = the mutation-model = just mutation.
  //Generation 0 was made by hand in the initilization
  for(int gen = 1; gen < numberOfGenerations; gen++)
   {
    //System.out.println("gen = " + gen);
    reproductionOfLeaves();
   }
   }

 //===== Inner class with the drawing machinery =======


 public class DrawingPane11 extends JPanel
```

```
  {
   private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00 =    scale*centerX;
int y00 =   scale* centerY + 400;
//Target coordinates
int x11 =  x00 + (int) ( scale*horizontalLength);
int y11 =  y00;
//Horizontal and basic vertical strokes
int H = scale * horizontalLength;
int V = scale * verticalLength;
//Addresses are interpreted as paths
String path = "";
char direction;

//Tree is drawn
private void drawTree(Graphics g)
 {
//The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
     for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Addresses will be used as paths to  be drawn
      path = Record[i];
      //Reinitialization
      Dimension center = new Dimension(x11,y11);
      V = scale * verticalLength;
       //Path is interpreted
         for(int j = 0; j < path.length(); j++)
          {
        direction = path.charAt(j);
        int x0 =  center.width;
    int y0 =  center.height;
```

```
    int x1 =  x0;
    int y1;
        if (direction == '0') //go up
         {
          y1 =  y0 + V;
          center.height = center.height + V;
         }
        else // if direction == '1'  =  go down
         {
          y1 =  y0 - V;
          center.height = center.height - V;
         } //End of if + else
      //Draw vertical stroke
    g.drawLine(x0, verticalSize -y0, x1,
    verticalSize- y1);
    //Draw horizontal stroke
    g.drawLine(x1, verticalSize - y1,
       x1 + H,  verticalSize - y1);
        //Move center
        center.width = center.width + H ;
        //Vertical stroke is divided in two
        V = V/2;
       }//End of for j
     }  //End of for: numberOfLeaves
 }


//Strings are drawn
private void drawStrings(Graphics g)
{
//Seed is drawn
g.drawString(seed,x00,verticalSize - y00);
//Strings are drawn:
    for(int i = 0; i < lastUsedPosition; i = i+2)
     {
      //Reinitialization
      Dimension center = new Dimension(x00,y00);
      Dimension target = center;
```

```
      V = scale * verticalLength;
      //Addresses will be used as paths to the places
      //where strings must be drawn
      path = Record[i];

      //Path is interpreted
      for(int j = 0; j < path.length(); j++)
       {
     direction = path.charAt(j);
     if (direction == '0') //go up
      target.height = target.height + V;
     else // if direction == '1'  =  go down
     target.height = target.height - V;

     target.width = target.width + H ;
     V = V/2;
       }//End of for j
      //String is drawn
      String s = Record[i+1];
      g.drawString(s, target.width + 1,
                     verticalSize - target.height-1);
      //System.out.println(path + " " +  s);
     }
   }


  protected void paintComponent(Graphics g)
   {
     super.paintComponent(g);
     g.setColor(Color.red);
     //Draw stem
    g.drawLine(x00, verticalSize - y00,
        x11,  verticalSize - y11);
    drawTree(g);
    drawStrings(g);
}//End of   paintComponent
  }
```

```
 // Create the GUI and show it.
 private static void createAndShowGUI()
  {
   //Main window.
   JFrame frame = new JFrame("BinaryEvo");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Content pane where to draw
   JComponent BinaryEvo4 = new BinaryEvo4();
   frame.setContentPane(BinaryEvo4);
   frame.setSize(horizontalSize,verticalSize);
   //Display the window.
   frame.setVisible(true);

 }


public static void main(String[] args)
 {
  //Thread technology
  javax.swing.SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
     {
      createAndShowGUI();
      evolution();
     }
   });
 }
} //End of main class J87 BinaryEvo4
```

88, page 138. False. Our simulation usually gets extinct because the process ends when there is no female. So, it is enough to augment the probability of female to get the process running.

94, page 156. Scrolling is added to a previous code.

```
//Program 94 BinaryEvo6
```

```
/*
* We run a mutation-model  of evolution
* restricted over a binary tree.
* It includes mutation  and recombination.
*
* We follow our own encoding rules
* and we use our own drawing machinery.
* Scrolling works perfectly.
*
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class BinaryEvo6 extends JPanel
                         implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane14;
 JScrollPane scroller;


 //Number of reproductive cycles
 private static int numberOfGenerations = 8;

 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;
```

```
 private static int verticalLength;


//Maximal number of nodes in the tree
private static int N = 10000;
//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
private static String[] Record =  new String[2*N];



 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
    "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();

 private static Random r = new Random();
 //Mutation rate.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;


public BinaryEvo6()
{
 super(new BorderLayout());
```

```
 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);
 clearButton.setVisible(true);

 JPanel instructionPanel = new JPanel(new GridLayout(0,1));
 instructionPanel.setFocusable(true);
 instructionPanel.add(clearButton);

 //Set up the drawing area.
 drawingPane14 = new drawingPane14();
 drawingPane14.setBackground(Color.white);


 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane14);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
 String command = e.getActionCommand();
 //If reset button is clicked over
 if  (RESET_COMMAND.equals(command))
  {
   evolution();
   repaint();
  }
}
```

```
//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
if (p<m)  //mutation
   { //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
   char c = Alphabet.charAt(l);
   mutant = mutant + c;
  }
 else
  { //no mutation
 mutant = mutant + individual.charAt(i);
  }
}
return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;
//Last used position  in Record[]
private static int lastUsedPosition;



//Number of reproductive leaves,
```

```
//those leaves of the last generation
private static int numberOfLeaves = N;
//Position of leaves in Record[]
private static  int[] leaves = new int[N];

//Output leaves and their number
public static void findLeaves()
{
 int counter = 0;
 //Reproductive leaves are the nodes at the end of Record[]
 //with the max length.
 String lastLeaf = Record[lastUsedPosition-1];
 int maxLength = lastLeaf.length();
 for(int i = lastUsedPosition-1; i >= 0; i = i-2)
  {
 //Nodes go at even entries
 fatherPath = Record[i];
//System.out.println("Node = " + i  + "  " + fatherPath);
 //If node is a reproductive leaf
 if( fatherPath.length() == maxLength )
   {
leaves[counter] = i;
counter++;
//System.out.println("Leaf = " + Record[i]);
   }

  }
 numberOfLeaves = counter;
}

//Flag for printing recombination
private static  boolean print= true;

//Record[i+1], the string associates to node i,
//is recombined
//with another chosen at random leaf
public static String recombination(int i)
{
```

```
//String to be recombined
String s = Record[i+1];
String  s1= "";
int recSite = 0;
int m= s.length();
if (m>0)
{
recSite = r.nextInt(m) ;
//Recombinant part of s
s1= s.substring(0,recSite);
}
//A leaf is chosen at random
int k = r.nextInt(numberOfLeaves);
String t = Record[leaves[k]+1];
//Recombinant part of t
//Bug corrected
String  t2=  t.substring(recSite+1);
//Recombination
String output = s1+t2;

if (print)
{
 System.out.println("Recombination");
 System.out.println("recSite = " + recSite);
 System.out.println(s);
 System.out.println(t);
 System.out.println(output);
}
return output;
}



//Node at position i is reproduced.
//Reproduction involves mutation and recombination
public static void reproduce(int i)
{
 String direction; //0 = up; 1 = down
```

```
    int t = r.nextInt(2);
    if ( t == 0 ) direction = "0";
    else direction = "1";
    childPath = fatherPath + direction;
    lastUsedPosition++;
    Record[lastUsedPosition] = childPath;
    //System.out.println("Path first = " + childPath);
    lastUsedPosition++;
    //The string associated with node i is recombined
    String s = recombination(i);
    //and then passed under mutation
    Record[lastUsedPosition] = mutation(s);
    //Must a second child exist?
    if (actualNumberOfChildren == 2)
  {
     //If a second child exists,
     //its direction is contrary to that of the first
     if ( t == 0 ) direction = "1";
     else direction = "0";
     childPath = fatherPath + direction;
     //System.out.println("Path second = " + childPath);
     lastUsedPosition++;
     Record[lastUsedPosition] = childPath;
     lastUsedPosition++;
     //The string associated with node i is recombined
     s = recombination(i);
     //and then passed under mutation
     Record[lastUsedPosition] = mutation(s);
     }
}


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
 {
  findLeaves();
 for(int i = 0; i < numberOfLeaves; i++)
```

```
   {
 int position = leaves[i];
 //Nodes go at even entries
 fatherPath = Record[position];
 //The number of children is within
 //0 and 2;
 actualNumberOfChildren =
     r.nextInt(maxNumbChildren + 1);
    /*
    System.out.println("Number of children = "
      + actualNumberOfChildren) ; */
    //Leaf undergoes reproduction
 if (actualNumberOfChildren > 0)
      reproduce(position);
//System.out.println(node.toString()) ;
   }
 }


  //An evolutionary process under the
  //mutation-model is triggered.
  //Every leaf (node of the last generation) is subjected
  //to reproduction with a number of children
  //equal or less than 2.
  //Reproduction may undergo mutation in each site.
  private static void evolution()
  {
   //Initialization of node 0: path goes up
Record[0] = "0";
//The string of node 0
Record[1] = mutation(seed);
//Initialization of node 1: path goes down
   Record[2] = "1";
   //The string of node 1
   Record[3] =  mutation(seed);
   //Last used position in Record[]
   lastUsedPosition = 3;
//Evolution is run for some reproductive cycles
```

```
//Evolution = the mutation-model = just mutation.
//Generation 0 was made by hand in the initilization
for(int gen = 1; gen < numberOfGenerations; gen++)
 {
  //System.out.println("gen = " + gen);
  reproductionOfLeaves();
 }
  }

//===== Inner class with the drawing machinery =======

/** The component inside the scroll pane. */
public class drawingPane14 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
```

```
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }
```

```
//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area = new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane14.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane14.revalidate();
 verticalLength = verticalSize/4;
}
```

```
//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
    }
 }



//Graphical  master
protected void paintComponent(Graphics g)
```

```
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;

  Font font = new Font("Arial", Font.PLAIN, 12);

  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);

  g2d.setColor(Color.red);
   //Draw stem
  g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
  sizeOfTree(g2d);
  drawTree(g2d);
  drawStrings(g2d);
 }
}


/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("BinaryEvo6");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new BinaryEvo6();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);
```

```
 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
     createAndShowGUI();
     evolution();
     }
   });
 }
}//End of main class 94 BinaryEvo6
```

95, page 156. Scrolling is added to a previous program.

```
//Program 95 BinaryEvo7
/*
* We run the full mutation-model  of evolution
* restricted over a binary tree.
* It includes mutation,  recombination,
* a rudimentary sexual dimorphism,
* and sexual reproduction.
*
* We follow our own encoding rules
* and we use our own drawing machinery.
* Scrolling works perfectly.
*
* The sign $ is a distinctive of maleness
* and recombination is possible between
* individuals of different sex only.
```

```
* Otherwise reproduction = parthenogenesis.
*
* If there is no female, the process stops.
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class BinaryEvo7 extends JPanel
                         implements ActionListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane15;
 JScrollPane scroller;


 //Number of reproductive cycles
 private static int numberOfGenerations = 4;

 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;


//Maximal number of nodes in the tree
private static int N = 10000;
```

```
//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
private static String[] Record =  new String[2*N];



 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
    "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet = "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();

 private static Random r = new Random();
 //Mutation rate.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;


public BinaryEvo7()
{
 super(new BorderLayout());

 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);
 clearButton.setVisible(true);
```

```
 JPanel instructionPanel = new JPanel(new GridLayout(0,1));
 instructionPanel.setFocusable(true);
 instructionPanel.add(clearButton);

 //Set up the drawing area.
 drawingPane15 = new drawingPane15();
 drawingPane15.setBackground(Color.white);


 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane15);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
 String command = e.getActionCommand();
 //If reset button is clicked over
 if  (RESET_COMMAND.equals(command))
  {
   evolution();
   repaint();
  }
}




//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
```

```
public static String mutation(String individual)
{
String mutant = "";
int lengthIndividual = individual.length();
for(int i = 0; i < lengthIndividual; i++)
{
double  p = r.nextDouble();
if (p<m)  //mutation
   { //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
   char c = Alphabet.charAt(l);
   mutant = mutant + c;
  }
 else
  { //no mutation
 mutant = mutant + individual.charAt(i);
  }
}
return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;
//Last used position  in Record[]
private static int lastUsedPosition;



//Position of leaves in Record[]
//Reproductive male leaves (of the last generation)
private static  int[] maleLeaves = new int[N];
private static  int[] femaleLeaves = new int[N];
private static int counterMaleLeaves;
```

```
private static int counterFemaleLeaves;

//Find leaves and count them.
//They are classified by gender.
public static void findLeaves()
{
 counterMaleLeaves = 0;
 counterFemaleLeaves = 0;
 //Reproductive leaves are the nodes at the end of Record[]
 //with the max length.
 String lastLeaf = Record[lastUsedPosition-1];
 int maxLength = lastLeaf.length();
 for(int i = lastUsedPosition-1; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  fatherPath = Record[i];
  //System.out.println("Leaf = " + Record[i]);
//System.out.println("Node = " + i  + "  " + fatherPath);
  //If node is a reproductive leaf
  if( fatherPath.length() == maxLength )
   {
 //Is it male?
 String s = Record[i+1];
 if (s.charAt(0) == '$')
  {
maleLeaves[counterMaleLeaves] = i;
counterMaleLeaves++;
//System.out.println("Male =  " + Record[i+1]);
  }
 else
  {
   femaleLeaves[counterFemaleLeaves] = i;
   counterFemaleLeaves++;
   //System.out.println("Female = " + Record[i+1]);
  }
   }

  }
```

```
}


//Flag for printing recombination
private static  boolean print= false;

//Record[i+1], the string associates to node i,
//is recombined
//with another chosen at random leaf.
//Recombination is allowed in the only case
//in which chosen individuals have different sex.
//Otherwise reproduction = Parthenogenesis
public static String recombination(int i)
{
//Female string to be recombined
String s = Record[i+1];
String  s1= "";

//The recombinant part of s is selected
int recSite = 0;
int m= s.length();
if (m>0)
{
recSite = r.nextInt(m) ;
//Recombinant part of s
s1= s.substring(0,recSite);
}

String output = "";
//A male partner  is chosen, if it exist
if (counterMaleLeaves > 0)
{
int k ;
String t ;

 k = r.nextInt(counterMaleLeaves);
 t = Record[maleLeaves[k]+1];
```

```
//Recombinant part of t
//Bug corrected
String  t2=  t.substring(recSite+1);
//Recombination
output = s1+t2;

if (print)
{
System.out.println("Recombination");
System.out.println("recSite = " + recSite);
System.out.println(s);
System.out.println(t);
System.out.println(output);
}
}
else output = s;

return output;
}



public static void bearChild(int i, String direction)
{

   //data for the first child
   childPath = fatherPath + direction;
   lastUsedPosition++;
   Record[lastUsedPosition] = childPath;
   //System.out.println("Path first = " + childPath);
   //String associated with node i
   String s = Record[i+1];
   //Sex indicator is filtered (if one exists)
   if (s.charAt(0) == '$') s = s.substring(1);
   //The string associated with node i is recombined,
   s = recombination(i);
   //passed under mutation
```

```
   s = mutation(s);
   //and recorded
   lastUsedPosition++;

   //Sex is assigned, female by default.
   int sex = r.nextInt(2);
   if (sex == 1 )
    Record[lastUsedPosition] =   "$" + s;
   else
   Record[lastUsedPosition] =    s;
   //System.out.println(childPath + " " +  s);
}

//Node at position i is reproduced.
//Reproduction involves mutation and recombination.
//Children are given a sex at random,
//$ = male, nothing = female.

public static void reproduce(int i)
{
 //First child
 String direction; //0 = up; 1 = down
 int t = r.nextInt(2);
 if ( t == 0 ) direction = "0";
 else direction = "1";
 bearChild(i, direction);
 //Data for a second child
 if (actualNumberOfChildren == 2)
     {
   //The direction of the second child is
   //contrary to that of the first.
  if ( t == 0 ) direction = "1";
  else direction = "0";
      bearChild(i, direction);
     }
}
```

```
//Terminal   leaves, nodes of the last generation,
//are found and children are generated.
//Only females can bear children.
private static void reproductionOfLeaves()
 {
  findLeaves();
 for(int i = 0; i < counterFemaleLeaves; i++)
   {
 int position = femaleLeaves[i];
 //Nodes go at even entries
 fatherPath = Record[position];
 //System.out.println(Record[position+1]) ;
 //The number of children is within
 //0 and 2;
 actualNumberOfChildren =
     r.nextInt(maxNumbChildren + 1);
    /*
    System.out.println("Number of children = "
      + actualNumberOfChildren) ; */
    //Leaf undergoes reproduction
 if (actualNumberOfChildren > 0)
      reproduce(position);

   }
 }


  //An evolutionary process under the
  //mutation-model is triggered.
  //Every leaf (node of the last generation) is subjected
  //to reproduction with a number of children
  //equal or less than 2.
  //Reproduction may undergo mutation in each site.
  private static void evolution()
  {
   //Initialization of node 0: path goes up
Record[0] = "0";
//The string of node 0 denotes a male: it has $
```

```
Record[1] = "$" + mutation(seed);
//Initialization of node 1: path goes down
   Record[2] = "1";
   //The string of node 1 denotes a female:
   //it does not have $
   Record[3] =  mutation(seed);
   //Last used position in Record[]
   lastUsedPosition = 3;
//Evolution is run for some reproductive cycles
//Evolution = the mutation-model = just mutation.
//Generation 0 was made by hand in the initilization
for(int gen = 1; gen < numberOfGenerations; gen++)
 {
  //System.out.println("gen = " + gen);
  reproductionOfLeaves();
 }
  }

//===== Inner class with the drawing machinery =======

/** The component inside the scroll pane. */
public class drawingPane15 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
```

```
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
```

```
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }


//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area = new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
```

```
 drawingPane15.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane15.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
```

```
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
    }
 }



//Graphical  master
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;

  Font font = new Font("Arial", Font.PLAIN, 12);

  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);

  g2d.setColor(Color.red);
   //Draw stem
  g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
  sizeOfTree(g2d);
  drawTree(g2d);
  drawStrings(g2d);
 }
}


/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
```

```
 JFrame frame = new JFrame("BinaryEvo7");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new BinaryEvo7();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
     createAndShowGUI();
     evolution();
     }
   });
 }
}//End of main class 95 BinaryEvo7
```

105, page 184. When the mutation rate is as high as 0.8 per site per generation, the previous program fails a significant proportion of times, say, 10%. This is due to the fact that a high mutation rate originates chaotic changes in which the resemblance with the father is lost. As a consequence, our algorithm chooses as direct ancestor of a given leaf the nearest neighbor among a set of candidates that were generated by randomness: of course it fails. So, we conclude that the nearest neighbor's principle endowed with the n-gram distance is good to single out the direct ancestor only when some

bounds are observed, in particular the mutation rate per site per generation must be sufficiently low, say, 0.1.

107, page 184. The n-gram distance singles out all direct ancestors in line:

```
//Program 107 TestNearest2

/*
* We test the power of the
* nearest neighbor's principle to single out
* the   ancestors of a leaf over an evolutionary process
* that is restricted to mutation  over a binary tree.

* Nearest neighbor's principle:
* My father is that man whose distance to me
* is the least among the members of the last generation.
* We choose the n-gram distance to measure differences.
*
* An n-gram is a substring n chars long.
* For a given text we calculate the vector of relative
* n-gram frequencies. The n-gram distance between two
* texts is the Euclidean distance (a generalization
* of the form  we measure the distance in the plane)
* between the two vectors of relative n-gram frequencies
* that are points of R^n.
*/


import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class TestNearest2 extends JPanel
                          implements ActionListener
{
```

```
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane17;
 JScrollPane scroller;


 //Number of reproductive cycles
 private static int numberOfGenerations = 6;

 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;


//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.
 private static String[] Record =  new String[1000000];
 //Last used position  in Record[]
 private static int lastUsedPosition;


 //Commands for the buttons
 private static String RESET_COMMAND = "Reset";

 private static   String   seed =
 "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet =
```

```
 "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();

 private static Random r = new Random();

 //MUTATION RATE.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;



/*The alphabet needed by fasta format.
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
                   */

//letters per n-gram
private static  int n=2;
//Number of n-grams
private static  int N = (int) Math.pow(lengthAlf, n);


//The frequency of appearance of n-grams in the text.
//If n>3,  the assigned memory must be expanded
private static double[]  freq1, freq2
                                = new double[N];

//private static int L = (int) Math.pow(lengthAlf+1, n);



private static Boolean printMore = false;
```

```
public TestNearest2()
{
 super(new BorderLayout());

 JButton clearButton = new JButton("Reset");
 clearButton.setActionCommand(RESET_COMMAND);
 clearButton.addActionListener(this);

 JPanel instructionPanel = new JPanel(new GridLayout(0,1));
 instructionPanel.setFocusable(true);
 instructionPanel.add(clearButton);

 //Set up the drawing area.
 drawingPane17 = new drawingPane17();
 drawingPane17.setBackground(Color.white);


 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane17);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Reaction to clicking of button
public void actionPerformed(ActionEvent e)
{
 String command = e.getActionCommand();
 //If reset button is clicked over
 if  (RESET_COMMAND.equals(command))
  {
   evolution();
   repaint();
```

```
  }
}


//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
 String mutant = "";
 int lengthIndividual = individual.length();
 for(int i = 0; i < lengthIndividual; i++)
   {
    double  p = r.nextDouble();
    if (p<m)  //mutation
       { //a random integer number
        int l = r.nextInt(lengthAlf);
        //A random char
        char c = Alphabet.charAt(l);
        mutant = mutant + c;
     }
    else
     { //no mutation
      mutant = mutant + individual.charAt(i);
      }
    }
 return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;

public static void reproduce(int i)
{
```

```
 String direction; //0 = up; 1 = down
 int t = r.nextInt(2);
 if ( t == 0 ) direction = "0";
 else direction = "1";
 childPath = fatherPath + direction;
 lastUsedPosition++;
 Record[lastUsedPosition] = childPath;
 //System.out.println("Path first = " + childPath);
 lastUsedPosition++;
 Record[lastUsedPosition] = mutation(Record[i+1]);
 if (actualNumberOfChildren == 1)
  {
   //If a second child exists,
   //its direction is contrary to that of the first
   if ( t == 0 ) direction = "1";
   else direction = "0";
   childPath = fatherPath + direction;
   //System.out.println("Path second = " + childPath);
   lastUsedPosition++;
   Record[lastUsedPosition] = childPath;
   lastUsedPosition++;
   Record[lastUsedPosition] = mutation(Record[i+1]);
  }
 }


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
{
 //Reproductive leaves are the nodes at the end of Record[]
 //with the max length.
 String lastLeaf = Record[lastUsedPosition-1];
 int maxLength = lastLeaf.length();
 for(int i = lastUsedPosition-1; i >= 0; i = i-2)
  {
   //Nodes go at even entries
   fatherPath = Record[i];
```

```
   //System.out.println("Nodee = " + i  + "  " + fatherPath);
   //If node is a reproductive leaf
   if( fatherPath.length() == maxLength )
      {
    //The number of children is within
    //0 and 2;
        actualNumberOfChildren =
         r.nextInt(maxNumbChildren + 1);
        /*
        System.out.println("Number of children = "
          + actualNumberOfChildren) ; */
        //Leaf undergoes reproduction
        if (actualNumberOfChildren > 0)
        {
          reproduce(i);
        }
    //System.out.println(node.toString()) ;
       }

  }
}


//An evolutionary process under the
//mutation-model is triggered.
//Every leaf (node of the last generation) is subjected
//to reproduction with a number of children
//equal or less than 2.
//Reproduction may undergo mutation in each site.
private static void evolution()
{
 //Initialization of node 0: path goes up
 Record[0] = "0";
 //The string of node 0
 Record[1] = mutation(seed);
 //Initialization of node 1: path goes down
 Record[2] = "1";
 //The string of node 1
```

```
 Record[3] =  mutation(seed);
 //Last used position in Record[]
 lastUsedPosition = 3;
 //Evolution is run for some reproductive cycles
 //Evolution = the mutation-model = just mutation.
 //Generation 0 was made by hand in the initilization
 for(int gen = 1; gen < numberOfGenerations; gen++)
  {
   //System.out.println("gen = " + gen);
   reproductionOfLeaves();
  }
}

//=========================================
//============= n-Gram methods ============
//=========================================


//Returns the   n-gram corresponding to index ind
//This is the inverse operation of index()
private static String indexInverse(int n, int ind)
{
char c1 = ' ';
int p = 0;
String ngram = "";
int k = 0;
int indl = ind;

//System.out.println("n = " + n);
for(int i = n-1; i > -1; i--)
{
//System.out.println("**** \nIndex = " + indl);
k = ( int ) (Math.pow(lengthAlf, i));
//System.out.println("k = " + k);
if( k <= indl)
{
p = indl / k;
//System.out.println("p = " + p);
```

```
if (p < Alphabet.length())
c1 = Alphabet.charAt(p);

indl = indl - p*k;
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

}
return ngram;
}


 //Returns the index   of ngram in F[] vectors.
 //An n-gram is a string with n chars taken from Alphabet.
 //Take it as a number in base = length of the Alphabet.
 //The index is that number in base 10.
 private static int index(int n, String ngram)
 {
  char c1 = ' ';
  int p = 0;
  int ind = 0;
  boolean correct = true;
  for(int i = 0; (i <n) & correct; i++)
   {
 c1 = ngram.charAt(i);
 p = Alphabet.indexOf(c1);
 if (p>=0)
  ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
 else
 {
  ind = -1;
  //Detecting the presence of an alien symbol
  correct = false;
 }
   }
  if (printMore)
```

```
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

/*
This procedure records in F the
frequency of appearance of n-grams in the given Text.
Each n-gram is given an index, a number that
identifies it in the  vector F.
At the beginning, every entry of F  is set to 0.
The index of each n-gram of the Text is
calculated and the corresponding entry in F  is
incremented by one.
*/
private static double[] nGramsFrequency(int n,
       StringBuffer Text)
{
 //Zeroed initialization
 double[] F = new double[N];
 int nText = Text.length();
 //System.out.println("Length  of working text = " + nText);
 int start = 0;
 int ind = 0;
 //Number of possible n-grams
 int nPossibleNGrams = nText-n+1;
 for(int c=0; c < nPossibleNGrams; c++)
  {
   String ngram =  Text.substring(start, start+n);
   //System.out.println("String ngram = " + ngram);
   ind = index(n,ngram);
   //if a given char of an n-gram is not in
   //the alphabet, its index is negative:
   //you must enlarge the alphabet else ignore it.
   //We ignore it.
   if (ind >= 0)   F[ind] = F[ind] + 1;
   start = start +1;
  }
```

```
  for(int i = 0; i < N; i++)
  F[i] = F[i]/nPossibleNGrams;
  double sum = 0;
  for(int i = 0; i < N; i++)
  sum = sum + F[i];
  //System.out.println("Frequencies add up to "+ sum);
  return F;
 }



 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
              double[] F1, double[] F2)
 {
  double sum = 0;
  double d = 0;
  String nGram = "";
  if (printMore)
System.out.println("\nRelative frequency " +
"vectors to compare");
  for(int i = 0; i < N; i++)
   {
  double a = F1[i]-F2[i];
sum = sum + a*a;
d = Math.sqrt(sum);
if (printMore)
{
 nGram = indexInverse(n, i);
 System.out.println( i + "  " +  nGram + "  " +
                  F1[i] + "  " + F2[i]);
}
   }
  return d;
 }
```

```
//=========================================
//=========== Ancestral tracer ===========
//=========================================



private static int memory = N;
//Number of nodes of the last generation
private static int numberOfFatherNodes = 0;
//Position of nodes of the previous generation in Record[]
private static  int[] PreviousGen = new int[memory];

//Nodes of the previous generation is found
//and the difference of each one of them
//to the  node under test is measured.
public static void
           findPreviousGeneration(int position)
{
int counter = 0;
//Number of bits in the node under test
int lengthThis = Record[position].length();
//The previous generation consists in those  nodes
//whose length = lengthThis-1


for(int i = position; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  String previous = Record[i];
  //If node belongs in the previous generation
  if( previous.length() == lengthThis - 1)
 {
  PreviousGen[counter] = i;
  counter++;
  //System.out.println("Previous = " + Record[i]);
 }
```

```
}
numberOfFatherNodes = counter;
}


//The n-gram distance of the node under test to
//nodes of the previous generation is measured.
//That node with minimum distance is single out.
public static int   findDirectAncestor(int position)
{
 double minDistance = 1000000000;
 int ancestorPosition = 0;
 StringBuffer s1 =
 new StringBuffer(Record[position+1]);
 //System.out.println("String under test = " + s1.toString());
 freq1 = nGramsFrequency(n, s1);

 for(int i = 0; i < numberOfFatherNodes; i++)
 {
  int j = PreviousGen[i];
  StringBuffer s2 =
 new StringBuffer(Record[j+1]);

  //System.out.println("\n" +  n + "-grams of   text 2:");
  freq2 = nGramsFrequency(n, s2);
  Double dist = euclideanDistance(N, freq1,freq2);
  //System.out.println(n + "-distance = " + dist);
  if (dist < minDistance)
   {
  minDistance = dist;
  ancestorPosition = j;
   }
  }
 System.out.println("Direct ancestor       = "
              + Record[ancestorPosition] );
 /*
 System.out.println("Direct ancestor = "
                   + Record[ancestorPosition+1]);*/
```

```
return ancestorPosition;
}

//A node is addressed by position.
//Its ancestors are found by means of the n-gram technology.
//One can verify the answer directly:
//the ancestor of 11101 is 1110 and
//that of 1010100 is 101010. And so on.
private static void findAncerstors(int position)
{
 //Number of bits in the node under test
 //= number of generations
 int lengthThis = Record[position].length();
 int ancestorPosition = position;
 for (int i = 0; i < lengthThis-1; i++)
  {
findPreviousGeneration(ancestorPosition);
ancestorPosition = findDirectAncestor(ancestorPosition);
  }
}



//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
public class drawingPane17 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
```

```
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
```

```
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }


//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
```

```
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area =
 new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane17.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane17.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
```

```
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
    }
 }

//Graphical  master
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;

  Font font = new Font("Arial", Font.PLAIN, 12);

  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);

  g2d.setColor(Color.red);
   //Draw stem
  g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
  sizeOfTree(g2d);
  drawTree(g2d);
  drawStrings(g2d);
  System.out.println("\nThe direct ancestor of  " +
                    Record[lastUsedPosition-1] + " is: ");
  findAncerstors(lastUsedPosition-1);
 }
}
```

```
/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("TestNearest");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new TestNearest2();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
     createAndShowGUI();
     evolution();
     System.out.println( "\nLetters per n-gram = " + n );
     System.out.println( "Lenght of the complete alphabet = " +
```

```
                                                lengthAlf);
     System.out.println( "Number of n-grams = " + N);
     }
   });
 }
}//End of main class 107 TestNearest2
```

108, page 184. We add interactive response:

```
//Program 108 TestNearest3

/*
* We test the power of the
* nearest neighbor's principle to single out
* the   ancestors of a leaf over an evolutionary process
* that is restricted to mutation  over a binary tree.
* Interactive response is implemented.
*
* Nearest neighbor's principle:
* My father is that man whose distance to me
* is the least among the members of the last generation.
* We choose the n-gram distance to measure differences.
*
* An n-gram is a substring n chars long.
* For a given text we calculate the vector of relative
* n-gram frequencies. The n-gram distance between two
* texts is the Euclidean distance (a generalization
* of the form  we measure the distance in the plane)
* between the two vectors of relative n-gram frequencies
* that are points of R^n.
*/



import java.awt.*;
import java.awt.event.*;
import java.util.*;
```

```
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;


public class TestNearest3 extends JPanel
                          implements
                          MouseListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane18;
 JScrollPane scroller;


 //Number of reproductive cycles
 private static int numberOfGenerations = 4;

 //Vertical  size of the panel
 private static int verticalSize;
 private static int horizontalSize;
 //Scale must be a power of 2
 private static int scale = 1;

 //Symbolic horizontal length of each stroke
 private static int horizontalLength = 30;

 private static int verticalLength;



//Evolution is recorded into a tree.
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
```

```
//odd ones are the strings, names or contents.

 private static int numberOfLeaves =
 (int) Math.pow(2, numberOfGenerations);

 private static String[] Record =  new String[4*numberOfLeaves];
 //Last used position  in Record[]
 private static int lastUsedPosition;


 private static   String   seed =
 "evolution is run and recorded over a binary tree";

 // Definition of the alphabet
 private static String Alphabet =
 "abcdefghijklmnopqrstuvwxyz ";
 //How many symbols in the alphabet
 private static  int lengthAlf   =  Alphabet.length();

 //Generator of random numbers
 private static Random r = new Random();

 //MUTATION RATE.
 private static double m = 0.1;
 //Max number of children per leaf.
 //The actual number of children is
 //a random number in within 0 and maxNumbChildren
 private static int maxNumbChildren = 2;



/*The alphabet needed by fasta format.
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
                   */

//letters per n-gram
private static  int n=2;
```

```
//Number of n-grams
private static  int N = (int) Math.pow(lengthAlf, n);


//The frequency of appearance of n-grams in the text.
//If n>3,  the assigned memory must be expanded
private static double[]  freq1, freq2
                                = new double[N];


//Y-coordinates of points where strings are drawn
private static int[] yCoordinate = new int[4*numberOfLeaves];
//Position in Record[] of ancestral nodes
private static int[] Ancestors = new int[numberOfGenerations];
//The color to paint
private static Color color = Color.red;

//Reset = true for evolution anew.
//Reset = false when ancestors are to be found.
private static Boolean reset = false;
private static int numberOfAncestors;
private static Boolean printMore1 = false;
private static Boolean printMore2 = true;
//Position in Record[] of clicked node.
private static int place;


public TestNearest3()
{
 super(new BorderLayout());


 JPanel instructionPanel = new JPanel(new GridLayout(0,1));

 //Set up the instructions.
 JLabel instructionsLeft = new JLabel(
     "Click left mouse button to reset.");
 JLabel instructionsRight = new JLabel(
```

```
  "Click right mouse button to reveal ancestors.");
 JLabel advertisment = new JLabel(
   "Nearest neighbord’s principle + n-gram distance.");

 instructionPanel.setFocusable(true);
 instructionPanel.add(instructionsLeft);
 instructionPanel.add(instructionsRight);
 instructionPanel.add(advertisment);

 //Set up the drawing area.
 drawingPane18 = new drawingPane18();
 drawingPane18.setBackground(Color.white);
 drawingPane18.addMouseListener(this);

 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane18);
 scroller.setPreferredSize(new Dimension(500,500));

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}


//Find the closest leaf to the spot of clicking.
private static int findClosestLeaf(int y)
 {
  int distance = 1000000;
  int position = 0;
  for(int i = 0; i < numberOfLeaves; i++)
   {
//pixels from clicked spot to a y-coordinate of a string
int pixels = Math.abs(y-yCoordinate[i]);
//The closest leaf is found
if ( pixels < distance )
  {
   distance = pixels;
   position  = i;
```

```
  }
   }
  return position;
 }


public void mouseClicked(MouseEvent e) { }


public void mousePressed(MouseEvent e) { }


public void mouseReleased(MouseEvent e)
{
 //Reset?
 if (SwingUtilities.isRightMouseButton(e))
  {
   //Reset: start evolution anew
   evolution();
   reset = true;
   color = Color.red;
   //Order to execute paintComponent(Graphics g)
   //of drawingPane18 with all its settings.
   repaint();
  }
  //Unveil ancestors?
 if (SwingUtilities.isLeftMouseButton(e))
  {
//Coordinates of clicked spot
int x = e.getX();
int y = e.getY();
System.out.println( x + " " + y);
//Closest node is found
int yPlace = findClosestLeaf(y);
System.out.println("Clicked Position = " + yPlace);
```

```
System.out.println("Path = " + Record[2*yPlace]);
reset = false;
//place = position in Record[] of the clicked node
//Place is a global variable used by
//the drawing machinery
    place = 2*yPlace;
    repaint();
  }

}



public void mouseEntered(MouseEvent e) { }



public void mouseExited(MouseEvent e) { }

//======Evolution====

//The string individual is subjected to mutation
//with mutation rate m per site per generation
public static String mutation(String individual)
{
 String mutant = "";
 int lengthIndividual = individual.length();
 for(int i = 0; i < lengthIndividual; i++)
   {
    double  p = r.nextDouble();
    if (p<m)  //mutation
       { //a random integer number
        int l = r.nextInt(lengthAlf);
        //A random char
        char c = Alphabet.charAt(l);
        mutant = mutant + c;
     }
    else
```

```
     { //no mutation
      mutant = mutant + individual.charAt(i);
      }
    }
 return mutant;
}

//Reproduction

private static String fatherPath;
private static String childPath;
private static   int actualNumberOfChildren;

public static void reproduce(int i)
{
 String direction; //0 = up; 1 = down
 int t = r.nextInt(2);
 if ( t == 0 ) direction = "0";
 else direction = "1";
 childPath = fatherPath + direction;
 lastUsedPosition++;
 Record[lastUsedPosition] = childPath;
 //System.out.println("Path first = " + childPath);
 lastUsedPosition++;
 Record[lastUsedPosition] = mutation(Record[i+1]);
 if (actualNumberOfChildren == 1)
  {
   //If a second child exists,
   //its direction is contrary to that of the first
   if ( t == 0 ) direction = "1";
   else direction = "0";
   childPath = fatherPath + direction;
   //System.out.println("Path second = " + childPath);
   lastUsedPosition++;
   Record[lastUsedPosition] = childPath;
   lastUsedPosition++;
   Record[lastUsedPosition] = mutation(Record[i+1]);
  }
```

```
 }


//Terminal leaves, nodes of the last generation, are found
//and children are generated.
private static void reproductionOfLeaves()
{
 //Reproductive leaves are the nodes at the end of Record[]
 //with the max length.
 String lastLeaf = Record[lastUsedPosition-1];
 int maxLength = lastLeaf.length();
 for(int i = lastUsedPosition-1; i >= 0; i = i-2)
  {
   //Nodes go at even entries
   fatherPath = Record[i];
   //System.out.println("Nodee = " + i  + "  " + fatherPath);
   //If node is a reproductive leaf
   if( fatherPath.length() == maxLength )
      {
    //The number of children is within
    //0 and 2;
        actualNumberOfChildren =
         r.nextInt(maxNumbChildren + 1);
        /*
        System.out.println("Number of children = "
          + actualNumberOfChildren) ; */
        //Leaf undergoes reproduction
        if (actualNumberOfChildren > 0)
        {
          reproduce(i);
        }
   //System.out.println(node.toString()) ;
      }

  }
 }
```

```
//An evolutionary process under the
//mutation-model is triggered.
//Every leaf (node of the last generation) is subjected
//to reproduction with a number of children
//equal or less than 2.
//Reproduction may undergo mutation in each site.
private static void evolution()
{
 //Initialization of node 0: path goes up
 Record[0] = "0";
 //The string of node 0
 Record[1] = mutation(seed);
 //Initialization of node 1: path goes down
 Record[2] = "1";
 //The string of node 1
 Record[3] =  mutation(seed);
 //Last used position in Record[]
 lastUsedPosition = 3;
 //Evolution is run for some reproductive cycles
 //Evolution = the mutation-model = just mutation.
 //Generation 0 was made by hand in the initilization
 for(int gen = 1; gen < numberOfGenerations; gen++)
  {
   //System.out.println("gen = " + gen);
   reproductionOfLeaves();
  }
}

//=========================================
//============ n-Gram methods ============
//=========================================


//Returns the   n-gram corresponding to index ind
//This is the inverse operation of index()
private static String indexInverse(int n, int ind)
{
char c1 = ' ';
```

```
int p = 0;
String ngram = "";
int k = 0;
int indl = ind;

//System.out.println("n = " + n);
for(int i = n-1; i > -1; i--)
{
//System.out.println("**** \nIndex = " + indl);
k = ( int ) (Math.pow(lengthAlf, i));
//System.out.println("k = " + k);
if( k <= indl)
{
p = indl / k;
//System.out.println("p = " + p);
if (p < Alphabet.length())
c1 = Alphabet.charAt(p);

indl = indl - p*k;
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

}
return ngram;
}


 //Returns the index   of ngram in F[] vectors.
 //An n-gram is a string with n chars taken from Alphabet.
 //Take it as a number in base = length of the Alphabet.
 //The index is that number in base 10.
 private static int index(int n, String ngram)
 {
  char c1 = ' ';
  int p = 0;
  int ind = 0;
```

```
 boolean correct = true;
 for(int i = 0; (i <n) & correct; i++)
  {
c1 = ngram.charAt(i);
p = Alphabet.indexOf(c1);
if (p>=0)
 ind = (int) (ind + p*Math.pow(lengthAlf, n-i-1));
else
{
 ind = -1;
 //Detecting the presence of an alien symbol
 correct = false;
}
  }
 if (printMore1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

/*
This procedure records in F the
frequency of appearance of n-grams in the given Text.
Each n-gram is given an index, a number that
identifies it in the  vector F.
At the beginning, every entry of F  is set to 0.
The index of each n-gram of the Text is
calculated and the corresponding entry in F  is
incremented by one.
*/
private static double[] nGramsFrequency(int n,
       StringBuffer Text)
{
 //Zeroed initialization
 double[] F = new double[N];
 int nText = Text.length();
 //System.out.println("Length  of working text = " + nText);
 int start = 0;
```

```
  int ind = 0;
  //Number of possible n-grams
  int nPossibleNGrams = nText-n+1;
  for(int c=0; c < nPossibleNGrams; c++)
   {
    String ngram =  Text.substring(start, start+n);
    //System.out.println("String ngram = " + ngram);
    ind = index(n,ngram);
    //if a given char of an n-gram is not in
    //the alphabet, its index is negative:
    //you must enlarge the alphabet else ignore it.
    //We ignore it.
    if (ind >= 0)   F[ind] = F[ind] + 1;
    start = start +1;
   }
  for(int i = 0; i < N; i++)
  F[i] = F[i]/nPossibleNGrams;
  double sum = 0;
  for(int i = 0; i < N; i++)
  sum = sum + F[i];
  //System.out.println("Frequencies add up to "+ sum);
  return F;
 }



 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
               double[] F1, double[] F2)
 {
  double sum = 0;
  double d = 0;
  String nGram = "";
  if (printMore1)
System.out.println("\nRelative frequency " +
"vectors to compare");
```

```
  for(int i = 0; i < N; i++)
   {
  double a = F1[i]-F2[i];
sum = sum + a*a;
d = Math.sqrt(sum);
if (printMore1)
{
 nGram = indexInverse(n, i);
 System.out.println( i + "  " +  nGram + "  " +
                  F1[i] + "  " + F2[i]);
}
   }
  return d;
 }



//=========================================
//=========== Ancestral tracer ===========
//=========================================



private static int memory = N;
//Number of nodes of the last generation
private static int numberOfFatherNodes = 0;
//Position of nodes of the previous generation in Record[]
private static  int[] PreviousGen = new int[memory];

//Nodes of the previous generation is found
//and the difference of each one of them
//to the  node under test is measured.
public static void
           findPreviousGeneration(int position)
{
int counter = 0;
//Number of bits in the node under test
int lengthThis = Record[position].length();
```

```
//The previous generation consists in those  nodes
//whose length = lengthThis-1


for(int i = position; i >= 0; i = i-2)
 {
  //Nodes go at even entries
  String previous = Record[i];
  //If node belongs in the previous generation
  if( previous.length() == lengthThis - 1)
 {
  PreviousGen[counter] = i;
  counter++;
  //System.out.println("Previous = " + Record[i]);
 }
}
numberOfFatherNodes = counter;
}


//The n-gram distance of the node under test to
//nodes of the previous generation is measured.
//That node with minimum distance is single out.
public static int   findDirectAncestor(int position)
{
 double minDistance = 1000000000;
 int ancestorPosition = 0;
 StringBuffer s1 =
 new StringBuffer(Record[position+1]);
 /*System.out.println("String under
  test = " + s1.toString());
  */
 freq1 = nGramsFrequency(n, s1);

 for(int i = 0; i < numberOfFatherNodes; i++)
 {
  int j = PreviousGen[i];
  StringBuffer s2 =
```

```
 new StringBuffer(Record[j+1]);

  //System.out.println("\n" +  n + "-grams of   text 2:");
  freq2 = nGramsFrequency(n, s2);
  Double dist = euclideanDistance(N, freq1,freq2);
  //System.out.println(n + "-distance = " + dist);
  if (dist < minDistance)
   {
  minDistance = dist;
  ancestorPosition = j;
   }
  }
return ancestorPosition;
}

 //print ancestors
private static void  printAncestors(int position)
{
 //Number of bits in the node under test
 //= number of generations
 int lengthThis = Record[position].length();
 if (printMore2)
  for (int i = 0; i < lengthThis; i++)
   {
 System.out.println("Direct ancestor       = "
             + Record[Ancestors[i]] );
   }
}


//A node is addressed by position.
//Its ancestors are found by means of the n-gram technology.
//One can verify the answer directly:
//the ancestor of 11101 is 1110 and
//that of 1010100 is 101010. And so on.
private static void findAncerstors(int position)
{
 //Number of bits in the node under test
```

```
 //= number of generations
 int lengthThis = Record[position].length();
 numberOfAncestors = lengthThis;
 int ancestorPosition = position;
 Ancestors[0] = ancestorPosition;
 for (int i = 0; i < lengthThis-1; i++)
  {
findPreviousGeneration(ancestorPosition);
ancestorPosition = findDirectAncestor(ancestorPosition);
Ancestors[i+1] = ancestorPosition;
  }
 printAncestors(position);
}



//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
public class drawingPane18 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00 =    scale*centerX;
 int y00 =   scale* centerY + verticalSize /2;
 //Target coordinates
 int x11 =  x00 + (int) ( scale*horizontalLength);
 int y11 =  y00;
 //Horizontal and basic vertical strokes
 int H = scale * horizontalLength;
 int V = scale * verticalLength;
```

```
 //Addresses are interpreted as paths
 String path = "";
 char direction;

 //Tree is drawn
 private void drawTree(Graphics2D g2d)
 {
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
```

```
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }


//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
 //Calculate the dimension of the strings
 //All strings have equal size.

 //Get   font in use
 Font font = g2d.getFont();
 //get metrics from the graphics
 FontMetrics metrics = g2d.getFontMetrics(font);
 //get the height of a line of text in this
 //font and render context (some padding is included)
 int hgt = metrics.getHeight()  + 9;
 //get the advance of my text in this font
 //and render context
 int lengthText = metrics.stringWidth(seed) + 2;

 //There are 2^n leaves
 verticalSize =
 (int) (hgt * Math.pow(2, numberOfGenerations));
 horizontalSize = lengthText +
 (int) (H* Math.pow(2, numberOfGenerations));

 //area = size of the graphic (width, height)
 Dimension area =
```

```
 new Dimension(verticalSize, horizontalSize);
 //area is set as the preferred size of the graphic
 drawingPane18.setPreferredSize(area);
 //The graphic demands a space for itself:
 //if the alloted space is not enough,
 //turn on scrolling bars.
 drawingPane18.revalidate();
 verticalLength = verticalSize/4;
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 color = Color.red;
 int counter = 0;
 //Seed is drawn
 g2d.drawString(seed,x00,verticalSize - y00);
 yCoordinate[counter] = verticalSize - y00;
 //counter++;
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
```

```
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
    }
  numberOfLeaves = counter;
 }

//Those strings that correspond with the ancestors
//are redrawn in another color.
private void drawAncestors(Graphics2D g2d, Color color)
{
g2d.setColor(color);
//Seed is drawn
g2d.drawString(seed,x00,verticalSize - y00);
//Strings are drawn:
for(int i = 0; i < numberOfAncestors; i++)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[Ancestors[i]];
 String s = Record[Ancestors[i]+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
```

```
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);

  }
}

//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
 g2d.setColor(Color.red);
 //Draw stem
 g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
 sizeOfTree(g2d);
 drawTree(g2d);
 drawStrings(g2d);


 if (reset)//Evolution anew
   {
 if (printMore2)
     System.out.println("\nThe direct ancestor of  " +
       Record[lastUsedPosition-1] + " is: ");
 findAncerstors(lastUsedPosition-1);
 color = Color.blue;
 drawAncestors(g2d,color);
}
  else //draw ancestors
   {
if (printMore2)
  System.out.println("\nThe direct ancestor of  " +
```

```
                    Record[place] + " is: ");
findAncerstors(place);
color = Color.blue;
drawAncestors(g2d,color);
   }
}

//Graphical  master : it is called automatically
//under starting and by repaint()
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  //Font to draw strings
  Font font = new Font("Arial", Font.PLAIN, 12);
  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
  drawBasic(g2d);
 }


}//End of inner class drawingPane18



/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("TestNearest");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
```

```
 //Constructor of the main application is used.
 JComponent newContentPane = new TestNearest3();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}

public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
 evolution();
 reset = true;
     createAndShowGUI();
     System.out.println( "\nLetters per n-gram = " + n );
     System.out.println( "Lenght of the complete" +
                         " alphabet = " + lengthAlf);
     System.out.println( "Number of n-grams = " + N);
     }
   });
 }



}//End of main class 108 TestNearest3
```

139, page 250. The code for a generative grammar of trees with 2 rules follows. Rule 1: AI $->$ (AN,FN)I. Rule 2: (S)I $->$ ((S)N, FN)I. Newick's notation. To test the code, we begin with tree (((AI,BI)J,CJ)K,(DK,EK)J) and generate all 7 trees that result from inserting branch FN with new leaf

F and internal node L.

```
//Program J139 Topologies2
//This program implements  rules 1 and 2
//to generate trees by the insertion of new branch FN:
//Rule 1:
//aI -> (aN,FN)I;
//a = leaf in tree;
//I = internal node already in tree;
//N = next Internal node to be inserted;
//F = nextLeaf;
//Rule 2:
// (S)I -> ((S)N,FN)I
//where (S) is a subtree. +
// N and I= internal nodes, F new leaf,
//FN new grafted branch.
//From tree ((AI,BI),IC) all tree possible
//children are generated.

//To test the code we insert branch FN,
//with new leaf F and new internal node N,
//to the tree (((AI,BI)J,CJ)K,(DK,EK)).
//Seven trees must be generated:
//5 by rule 1 and
//2 by rule 2.


public class Topologies2
{

 //Link's notation for trees
 private static   String   parent =
 "(((AI,BI)J,CJ)K,(DK,EK))";
 private static  String leaves =  "ABCDEFGRSTUVWXYZ";
 private static String internalNodes = "IJKLMNOP";

 private static boolean print = false;
```

```
 //Detect internal nodes, different than leaves
 private static boolean charIsInternalNode(char c)
 {
  String s = "" + c;
  if (internalNodes.contains(s)) return true;
  else return false;
 }

//Simplification from Link notation to Newick's notation:
 // (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
 String tree = "";
//length of string newTree
int l = newTree.length();
 for(int i = 0; i < l; i++ )
  {
   //newTree is scanned char by char
  char c = newTree.charAt(i);
  //Is c an internal node to be deleted?
  if (charIsInternalNode(c));
   else //add char c to output
    tree = tree + c;
  }
 return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
        char F, char N)
{
 String newTree = "";
 char I = parentTree.charAt(i);
```

```
 if (print)
 System.out.println("i = " + i + " I = " + I);
 //Subtree S of link (S)I must be read
 String S = "";
 //Counter of left parenthesis
 int counter = 0;
 String pl = "" + '(';
 String pr = "" + ')';
 //Position to be scanned
 int position = i;
 boolean go = true;
 //All right parentheses to the near left of I are captured
 while (go)
  {
   position = position -1;
   String a = "" + parentTree.charAt(position);

   if (print)
   System.out.println("a = " + a + " p = " + pr);
   if (a.equals(pr) )
   {
counter++;
S = a + S;
   }
   else go = false;
  }//End of while
 if (print)
 System.out.println("counter = " + counter);
 //The complete subtree is captured:
 //Parentheses must balanced
 while (counter > 0)
  {
   String a = "" + parentTree.charAt(position);
   if (a.equals(pr))   counter++;
   if (a.equals(pl))  counter--;
   S = a + S;
   if (print)
   System.out.println("Subtree = " + S);
```

```
   position = position -1;
   }
 //Replacement rule 2:
 //(S)I -> ((S)N,FN)I.  N= nextIntNode;
 //F = nextLeaf
 //Target to be replaced
 String target  =   S  + I;
 String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(target, replacement);
 //If I links to a complex structure
 return newTree;
}


//Trees are generates by a generative grammar
private static void generateTrees(String parentTree)
{
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(5);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(3);
 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = parentTree.length();
 for(int i = 0; i < l; i++ )
  {
    //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
```

```
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree, i,
              nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
          + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(link, replacement);
   }
 System.out.println(newTree + " -> "
                       + simplify(newTree));
}
  }
}


public static void main(String[] args)
{
 System.out.println("Parent tree = \n" + parent
 + " -> "    + simplify(parent));
 System.out.println("Offspring :  "   );
 generateTrees(parent);
}

}//End of main class J139 Topologies2
```

143, page 265. To understand how to assign addresses to leaves from Newick's notation, some case studies will help us. Helping us with previous drawings, we have generated some assignments:

((A,B),C) = ((00,01),1)

(((A,B),C), (D,E)) = (((000,001),01), (10,11))

(((A,D),(B,E)),(C,F)) = (((000,001),(010,011)),(10,11))

So, which are the rules?

We propose the following rules given that one reads a string from left to right:

1. A left parentheses means rise a level and add a zero to the right.
2. A right parenthesis means go down one level and delete a zero to the beginning of the binary string.
3. A semicolon means change from left to right, change the most right 0 by a 1.

144, page 266. A generator of trees at random:

```
//Program J144 Topologies3
//This program is a generator of
//random trees in Newick's notation.
//Up to 11 leaves are allowed.

import java.util.Random;


public class Topologies3
{

 //Link's notation for trees
 private static   String   parent =
 "((AI,BI),CI)";
 private static  String leaves =  "ABCDEFGHUVWXYZ";
 private static String internalNodes = "IJKLMNOPQRST";
 //Next leaf to be grafted
 private static  int nextLeafIndex = 2;
 //Next internal node to be inserted
 private static  int nextIntNodeIndex = 0;

 private static  Random r = new Random();
```

```
 private static boolean print = false;


 //Detect internal nodes, different than leaves
 private static boolean charIsInternalNode(char c)
 {
  String s = "" + c;
  if (internalNodes.contains(s)) return true;
  else return false;
 }

//Simplification from Link notation to Newick's notation:
 // (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
 String tree = "";
//length of string newTree
int l = newTree.length();
 for(int i = 0; i < l; i++ )
  {
   //newTree is scanned char by char
  char c = newTree.charAt(i);
  //Is c an internal node to be deleted?
  if (charIsInternalNode(c));
   else //add char c to output
    tree = tree + c;
  }
 return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
```

```
        char F, char N)
{
 String newTree = "";
 char I = parentTree.charAt(i);
 if (print)
 System.out.println("i = " + i + " I = " + I);
 //Subtree S of link (S)I must be read
 String S = "";
 //Counter of left parenthesis
 int counter = 0;
 String pl = "" + '(';
 String pr = "" + ')';
 //Position to be scanned
 int position = i;
 boolean go = true;
 //All right parentheses to the near left of I are captured
 while (go)
  {
   position = position -1;
   String a = "" + parentTree.charAt(position);

   if (print)
   System.out.println("a = " + a + " p = " + pr);
   if (a.equals(pr) )
   {
counter++;
S = a + S;
   }
   else go = false;
  }//End of while
 if (print)
 System.out.println("counter = " + counter);
 //The complete subtree is captured:
 //Parentheses must balanced
 while (counter > 0)
  {
   String a = "" + parentTree.charAt(position);
   if (a.equals(pr))   counter++;
```

```
   if (a.equals(pl))  counter--;
   S = a + S;
   if (print)
   System.out.println("Subtree = " + S);
   position = position -1;
   }
 //Replacement rule 2:
 //(S)I -> ((S)N,FN)I.  N= nextIntNode;
 //F = nextLeaf
 //Target to be replaced
 String target  =   S  + I;
 String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(target, replacement);
 //If I links to a complex structure
 return newTree;
}

//From parentTree, a tree in Newick's notation,
//a Child  tree is generated by a generative grammar.
//A random election is made among all possibilities.
private static String generateChild(String parentTree)
{

 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);


 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
```

```
 //length of parent tree
 int l = parentTree.length();
 //counter of links = 0;
 int counter = 0;
 //Links are counted
 for(int i = 0; i < l; i++ )
  {
   //parentTree is scanned char by char
   char I = parentTree.charAt(i);
//Is I an internal node that denotes a link?
if (charIsInternalNode(I)) counter++;
  }
 //A link is chosen
 int k = r.nextInt(counter);
 System.out.println("Random number = " + k);
 //new branch is grafted at k
 counter = 0;
 for(int i = 0; i < l; i++ )
  {
    //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  if (counter == k) //Is I the chosen link?
   {
//Check for the char before I
char a = parentTree.charAt(i-1);
//If I links to a complex structure
if (a == ')') newTree = useRule2(parentTree, i,
               nextLeaf, nextIntNode);
else //if I links to a leaf
 {
  //Link to be replaced
  link = "" + a + I;
  //Replacement rule 1:
  //aI -> (aN,FN)I;   N= nextIntNode;
  //F = nextLeaf
```

```
 replacement = "(" + a + nextIntNode + ","
          + nextLeaf + nextIntNode + ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(link, replacement);
 }
System.out.println(newTree + " -> "
                       + simplify(newTree));


 i = l;
   }
  counter++;
}
  }
 return newTree;
}


public static void main(String[] args)
{
 System.out.println("Parent tree = \n" + parent
 + " -> "    + simplify(parent));
 int limit = 11; //less than 11.
 for(int i = 0; i < limit; i++ )
 {
  parent = generateChild(parent);
 }
}

}//End of main class J144 Topologies3
```

147, page 273. A tree is drawn:

```
//Program J147 NwckToBin2


//This program makes  a translation of a tree in
//Link notation ( a variation of Newick's notation )
```

```
//into binary notation.
//The tree is drawn.



import java.awt.*;
import java.awt.event.*;
import java.util.Random;


import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;


public class NwckToBin2 extends JPanel
                         implements
                         MouseListener
{
 private static final long serialVersionUID = 1L;
 private JPanel drawingPane21;
 JScrollPane scroller;

 //How many leaves must the tree have?
 private static int numberOfLeaves = 8;


 private static String[] Input = new String[10000];

 static String[] Record =  new String[10000];
 //Last filled position  in Record[] and Input[];
 private static int lastUsedPosition;
```

```
//=======================================



//Vertical  size of the panel
private static int verticalSize;
private static int horizontalSize;
//Scale must be a power of 2
private static int scale = 1;

//Symbolic horizontal length of each stroke
private static int horizontalLength = 30;

private static int verticalLength;




//Link's notation for trees
private static   String   parent =
"((Aa,Bb),Cc)";
private static   String  NwTree = "";
private static  String leaves =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
private static String internalNodes =
"abcdefghijklmnopqrstuvwxyz";
//Next leaf to be grafted
private static  int nextLeafIndex = 2;
//Next internal node to be inserted
private static  int nextIntNodeIndex = 0;

private static  Random r = new Random();

private static boolean print = false;
private static boolean print2 = true;
```

```
//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
private static int[] yCoordinate = new int[1000];

//Reset = true for a new tree
private static Boolean reset = false;

//Set up the instructions.
JLabel instructionsLeft;

public NwckToBin2()
{
 super(new BorderLayout());


 JPanel instructionPanel = new JPanel(new GridLayout(0,1));

 //Set up the instructions.
 instructionsLeft = new JLabel(
    "Tree = " +  NwTree);
 JLabel instructionsRight = new JLabel(
  "Click right mouse button to reset. ");
 JLabel advertisment = new JLabel(
   "Nearest neighbord's principle + n-gram distance.");

 instructionPanel.setFocusable(true);
 instructionPanel.add(instructionsLeft);
 instructionPanel.add(instructionsRight);
 instructionPanel.add(advertisment);


 //Set up the drawing area.
```

```
 drawingPane21 = new drawingPane21();
 drawingPane21.setBackground(Color.white);
 drawingPane21.addMouseListener(this);

 //Put the drawing area in a scroll pane.
 JScrollPane scroller = new JScrollPane(drawingPane21);
 scroller.setPreferredSize(new Dimension(500,500));
 scroller.revalidate();

 //Lay out this demo.
 add(instructionPanel, BorderLayout.PAGE_END);
 add(scroller, BorderLayout.CENTER);
}



public void mouseClicked(MouseEvent e) { }



public void mousePressed(MouseEvent e) { }



public void mouseReleased(MouseEvent e)
{
 //Reset?
 if (SwingUtilities.isRightMouseButton(e))
  {
   treeMain();
   readInput();
   instructionsLeft.setText("Tree = " + NwTree);
   repaint();
   repaint();
  }
}
```

```
public void mouseEntered(MouseEvent e) { }


public void mouseExited(MouseEvent e) { }


//======================


 //=======================================
 //========= Generator of trees ==========
 //=======================================

 //A tree is generated in Link and Newick's notation
 //and then translated to binary notation.

 //Binary notation:
 //A vector of strings.
 //Even entries for binary addresses,
 //odd entries for contents.



 //Detect internal nodes, different than leaves
 private static boolean charIsInternalNode(char c)
 {
  String s = "" + c;
  if (internalNodes.contains(s)) return true;
  else return false;
 }

//Simplification from Link notation to Newick's notation:
 // (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
 String tree = "";
```

```
//length of string newTree
int l = newTree.length();
 for(int i = 0; i < l; i++ )
  {
   //newTree is scanned char by char
  char c = newTree.charAt(i);
  //Is c an internal node to be deleted?
  if (charIsInternalNode(c));
   else //add char c to output
    tree = tree + c;
  }
 return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
        char F, char N)
{
 String newTree = "";
 char I = parentTree.charAt(i);
 if (print)
 System.out.println("i = " + i + " I = " + I);
 //Subtree S of link (S)I must be read
 String S = "";
 //Counter of left parenthesis
 int counter = 0;
 String pl = "" + '(';
 String pr = "" + ')';
 //Position to be scanned
 int position = i;
 boolean go = true;
 //All right parentheses to the near left of I are captured
 while (go)
```

```
  {
   position = position -1;
   String a = "" + parentTree.charAt(position);

   if (print)
   System.out.println("a = " + a + " p = " + pr);
   if (a.equals(pr) )
   {
counter++;
S = a + S;
   }
   else go = false;
  }//End of while
 if (print)
 System.out.println("counter = " + counter);
 //The complete subtree is captured:
 //Parentheses must balanced
 while (counter > 0)
  {
   String a = "" + parentTree.charAt(position);
   if (a.equals(pr))   counter++;
   if (a.equals(pl))  counter--;
   S = a + S;
   if (print)
   System.out.println("Subtree = " + S);
   position = position -1;
   }
 //Replacement rule 2:
 //(S)I -> ((S)N,FN)I.  N= nextIntNode;
 //F = nextLeaf
 //Target to be replaced
 String target  =   S  + I;
 String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(target, replacement);
 //If I links to a complex structure
 return newTree;
```

```
}

//From parentTree ( a tree in Newick’s notation)
//a Child  tree is generated by a generative grammar.
//A random election is made among all possibilities.
private static String generateChild(String parentTree)
{
 if (print)
  System.out.println("Parent at generate = " + parentTree);
 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);
 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = parentTree.length();
 //counter of links = 0;
 int counter = 0;
 //Links are counted
 for(int i = 0; i < l; i++ )
  {
   //parentTree is scanned char by char
   char I = parentTree.charAt(i);
//Is I an internal node that denotes a link?
if (charIsInternalNode(I)) counter++;
  }


 //A link is chosen
 int k = r.nextInt(counter);
 if (print)
  System.out.println("Random number = " + k);
```

```
 //new branch is grafted at k
  counter = 0;


 for(int i = 0; i < l; i++ )
  {
    //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  if (counter == k)
   {
//Check for the char before I
char a = parentTree.charAt(i-1);
//If I links to a complex structure
if (a == ')') newTree = useRule2(parentTree, i,
               nextLeaf, nextIntNode);
else //if I links to a leaf
 {
  //Link to be replaced
  link = "" + a + I;
  //Replacement rule 1:
  //aI -> (aN,FN)I;   N= nextIntNode;
  //F = nextLeaf
 replacement = "(" + a + nextIntNode + ","
          + nextLeaf + nextIntNode + ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(link, replacement);
 }
 if (print)
  System.out.println(newTree + " -> "
             + simplify(newTree)
             + " -> " + toBinary(newTree));


 i = l;//End work
   }
```

```
  counter++;
}
  }
 return newTree;
}


//A tree at random is generated
private static String generateTree(String parentTree)
{
 nextLeafIndex = 2;
 //Next internal node to be inserted
 nextIntNodeIndex = 0;
 parent =   "((Aa,Bb),Cc)";
 if (print)
  System.out.println("nLeaves  = "
                             + numberOfLeaves);
 //The first 3 leaves come with the initial tree
 for(int i = 0; i < numberOfLeaves - 3; i++ )
  {
   parent = generateChild(parent);
  }
 return parent;
}


//Translation from Newick's to quasi-binary notation
private static String toBinary(String tree)
{
 String regulators = "(,)";
 //Output
 String treeToBi = "";
 //The binary address of leaf
 String address = "";
 //The name or content of the tree
 String content = "";
 String thisChar = "";
 String nextChar = "";
```

```
 String previousChar = "";
 String prepreviousChar = "";
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if ( i < l-1) nextChar = "" + tree.charAt(i+1);
   else nextChar = "end";
   if ( i > 0) previousChar = "" + tree.charAt(i-1);
   else previousChar = "start";
   //Is thisChar a regulator?
   if (regulators.contains(thisChar))
 {
   treeToBi =  treeToBi + thisChar;

   //This char = (
   if (thisChar.contentEquals("("))
   {
address = address + '0';
if (print)
System.out.println("Address = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
 {

  content = ""+ nextChar;
  treeToBi =  treeToBi + address + content;

}
   }


 //This char = ,
   if (thisChar.contentEquals(","))
   {
if (print)
```

```
System.out.println("Address at ,   = " + address);
int m = address.length();
//replace last 0 by a 1
address = address.substring(0, m-1) + '1';
if (print)
  System.out.println("Address at , 1 = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
  {

 content = ""+ nextChar;

 treeToBi =  treeToBi + address + content;

  }
   }


 //This char = )
  if (thisChar.contentEquals(")"))
   {
  prepreviousChar = "" + tree.charAt(i-2);
if ( (internalNodes.contains(previousChar)
& (leaves.contains(prepreviousChar))))
{
 content = "" + tree.charAt(i-2);


}
int m = address.length();
//Go one level down the tree
    address = address.substring(0, m-1);
   }


 } //End of   if (regulators.contains(thisChar))
   }//End of for(int i...
 return treeToBi;
```

```
}


//bTree in complex Newick's notation
//is used to generate Input[]
//with address and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Input[counter] = address;
    counter++;
 Input[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Input[i] +
 " bbbb " + Input[i+1]);
```

```
lastUsedPosition = counter;

}


public static void treeMain()
{
 if (print)
  System.out.println("Parent tree = \n" + parent
 + " -> "    + simplify(parent));
 String tree = generateTree(parent);
 NwTree = simplify(tree);
 String bTree = toBinary(tree);
 makeInput(bTree);
 if (print2)
 {
  System.out.println("\nTree in Newick's notation: \n"
         + tree);
  System.out.println("Tree in quasi-binary notation: \n"
               + bTree);
 }
}

//Input is read
private static void readInput()
{
 if (print)
System.out.println("nLeaves = " + numberOfLeaves);
  for (int i = 0; i < lastUsedPosition; i++)
  Record[i] = Input[i];
}




//=========================================
//============= Painting class ============
//=========================================
```

```
/** The component inside the scroll pane. */
public class drawingPane21 extends JPanel
{
 private static final long serialVersionUID = 1L;

 //Initialization of coordinates
 int centerX = 1;
 int centerY = 1;
 //Coordinates of the origin
 int x00;
 int y00;
 //Target coordinates
 int x11 ;
 int y11;
 //Horizontal and basic vertical strokes
 int H;
 int V;
 //Addresses are interpreted as paths
 String path = "";
 char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
```

```
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
 System.out.println("height of letters = " + hgt);

horizontalSize = lengthText +
                (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane21.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane21.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
 x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;
//Horizontal and basic vertical strokes
 H = scale * horizontalLength;
 V = scale * verticalLength;
}



 //Tree is drawn
```

```
 private void drawTree(Graphics2D g2d)
 {
 if (print)
  for(int i = 0; i < lastUsedPosition; i= i+2)
  System.out.println(Input[i]
  + " input " + Input[i+1]);
  //The tree is drawn:
  //A path from the origin of the tree to
  //each leaf is drawn
  for(int i = 0; i < lastUsedPosition; i = i+2)
   {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
```

```
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
  }



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
 int counter = 0;
 //Seed is not drawn: our tree is unrooted

 yCoordinate[counter] = verticalSize - y00;
 //counter++;
 //Strings are drawn:
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
   //Addresses will be used as paths to the places
   //where strings must be drawn
   path = Record[i];
   String s = Record[i+1];
   //Reinitialization
   Dimension center = new Dimension(x00,y00);
   Dimension target = center;
   V = scale * verticalLength;
   //Path is interpreted
   for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
```

```
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                       verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
    }
 }


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
 g2d.setColor(Color.red);
 //Draw stem
 g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
 sizeOfTree(g2d);
 drawTree(g2d);
 drawStrings(g2d);


 if (reset)//Evolution anew
   {
 treeMain();
 readInput();
 repaint();
}

}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
```

```
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  //Font to draw strings
  Font font = new Font("Arial", Font.PLAIN, 12);
  g2d.setFont(font);
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
  drawBasic(g2d);
 }


}//End of inner class drawingPane21



/**
 * Create the GUI and show it.  For thread safety,
 * this method should be invoked from the
 * event-dispatching thread.
*/
private static void createAndShowGUI()
{
 //Create and set up the window.
 JFrame frame = new JFrame("Visor of trees in Link notation");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //Create and set up the content pane.
 //Constructor of the main application is used.
 JComponent newContentPane = new NwckToBin2();
//content panes must be opaque
 newContentPane.setOpaque(true);
 frame.setContentPane(newContentPane);

 //Display the window.
 frame.pack();
 frame.setVisible(true);
}
```

```
public static void main(String[] args)
{
 //Schedule a job for the event-dispatching thread:
 //creating and showing this application's GUI.
 javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
   public void run()
    {
 //Built a tree in Newick's notation
 treeMain();
 //Input for the drawing machinery is prepared
 readInput();
 //Tree is drawn
     createAndShowGUI();
     }
   });
 }

}//End of main class 147 NwckToBin2
```

150, page 298. . Inner nodes get visualized.

Our case study is the tree (((A,F),(B,E)),(C,D)) = (((Ad,Fd)a,(Bc,Ec)a),(Cb,Db)a). We add a root in within the lowest link. This operation generates an address for each node as shown below:

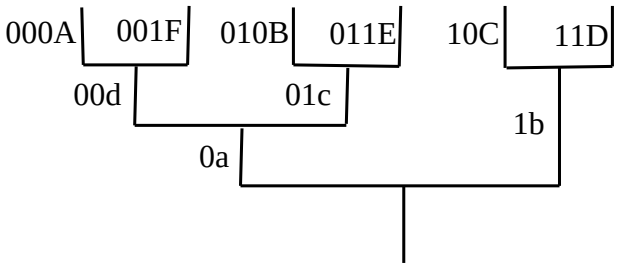


Figure 6.0. Inner nodes and binary addresses associated to the tree (((A,F),(B,E)),(C,D)) = (((Ad,Fd)a,(Bc,Ec)a),(Cb,Db)a).

Our tentative rules to generate addresses look as follow:

- ($->$ add a zero to the right
- Leaf or inner node $->$ print address and leaf or inner node.
- Inner node $->$ delete last bit from the right.
- , $->$ add a one to the right.
-) $->$ do nothing.
-), $->$ delete last bit from the right.

The code to test these rule follows. If amendments are necessary, they will be done directly there:

```
//Program J150 TreeOfTrees3
//Beginning with tree ((A,B),C) = ((AI,BI),CI)
//we generate all possible topologies of trees.
//The resultant data structure is that of a tree.
//It is shown here.


//Topologies are generated
//and one can select one of them
//for visualization.
//Inner nodes also depicted.


//Topology = the graph of the tree without lengths

//Formats = Link and Newick's notations.
//You can learn these notations by looking at drawings.




import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.util.Enumeration;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class TreeOfTrees3 extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;



 //Everything is built by recursion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
 "ABCDEFGHIJKLMNOPRSTUVWXYZ";
```

```
private static String internalNodes =
"abcdefghijklmnopqrstuvwxyz";

private static boolean print = false;
private static boolean print2 = true;


//Number of leaves that must go in all trees
private static int numberOfLeaves = 6;


private static String[] Record =  new String[10000];
//Last filled position  in Record[]
private static int lastUsedPosition;
private static String selectedTree = "";

 //DynamicTree is an inner service class that comes
 //at the end of this file.
 private static DynamicTree6 treePanel;
 private static String VISUALIZE_COMMAND = "Visualize";

 //Constructor of main class: GUI basics
 public TreeOfTrees3()
 {
  super(new BorderLayout());
  JPanel instructionPanel = new JPanel(new GridLayout(0,1));
  JLabel instructionsLeft = new JLabel(
     "Select  a tree and click the Visualize button.");

  instructionPanel.setFocusable(true);
  instructionPanel.add(instructionsLeft);
  add(instructionPanel, BorderLayout.PAGE_END);

  //A new node is added
  JButton visualizeButton = new JButton("Visualize");
  visualizeButton.setActionCommand(VISUALIZE_COMMAND);
  visualizeButton.addActionListener(this);
```

```
   // Create the components.
   treePanel = new DynamicTree6();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(1200, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
      //Selected tree must be unveiled
      selectedTree = treePanel.unveilTree();
      System.out.println("\nSelected tree   after " +
       "unveilTree() = \n"   + selectedTree);

  //Translation from one format into another
      //output = a vector with binary address and contents
      translate(selectedTree);
      DrawBiTree y = new DrawBiTree();
      y.createAndShowGUI();
 }
  }

//======================

  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
```

```
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Visual generator of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees3 newContentPane = new TreeOfTrees3();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}
```

```
//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //add char c to output
   tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
```

```
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
```

```
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //Trees are generated
  private static void generateTrees()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }




//=============== Translation in within formats ============




//Translation from Newick's to quasi-binary notation
  //Inner nodes are lacked
private static String toBinary(String tree)
{
 String regulators = "(,)";
 //Output
```

```
 String treeToBi = "";
 //The binary address of leaf
 String address = "";
 //The name or content of the tree
 String content = "";
 String thisChar = "";
 String nextChar = "";
 String previousChar = "";
 String prepreviousChar = "";
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if ( i < l-1) nextChar = "" + tree.charAt(i+1);
   else nextChar = "end";
   if ( i > 0) previousChar = "" + tree.charAt(i-1);
   else previousChar = "start";
   //Is thisChar a regulator?
   if (regulators.contains(thisChar))
 {
   treeToBi =  treeToBi + thisChar;

   //This char = (
   if (thisChar.contentEquals("("))
   {
address = address + '0';
if (print)
System.out.println("Address = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
 {

  content = ""+ nextChar;
  treeToBi =  treeToBi + address + content;

}
```

```
   }


 //This char = ,
   if (thisChar.contentEquals(","))
   {
if (print)
System.out.println("Address at ,   = " + address);
int m = address.length();
//replace last 0 by a 1
address = address.substring(0, m-1) + '1';
if (print)
  System.out.println("Address at , 1 = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
  {

 content = ""+ nextChar;

 treeToBi =  treeToBi + address + content;

  }
   }


 //This char = )
  if (thisChar.contentEquals(")"))
   {
  prepreviousChar = "" + tree.charAt(i-2);
if ( (internalNodes.contains(previousChar)
& (leaves.contains(prepreviousChar))))
{
 content = "" + tree.charAt(i-2);


}
int m = address.length();
//Go one level down the tree
```

```
    address = address.substring(0, m-1);
   }


 } //End of   if (regulators.contains(thisChar))
   }//End of for(int i...
 return treeToBi;
}


//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
```

```
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved.
private static String toQuasiBinaryInner(String tree)
{
//Output
String treeToBi = "";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
```

```
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + ’0’;
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }


 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "0";
   treeToBi =  treeToBi + address + thisChar;
   //if (m==1) treeToBi =   "0" + thisChar;
  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
```

```
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (internalNodes.contains(previousChar))
  address = address + '1';//Add a 1 to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "1";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
```

```
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "01";
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
   counter++;
 Record[counter] = thisChar;

counter++;
address = "";
 }
}

if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}


//Tree is translated
//from link notation to binary.
//Output in Record[]
```

```
public static void translate(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
 if (print)
 System.out.println("Tree  at translate = "    + tree);
 //String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

//A vector with addresses and names is composed
makeInputInner(bTree);



 if (print)
 System.out.println("lastUsedPosition = "
                      + lastUsedPosition);
}




public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
```

```
    {
      public void run()
      {
        createAndShowGUI();
        generateTrees();
      }
    });
  }
```

```
//=========================================================
//==================Inner service class====================
//=========================================================


class DynamicTree6 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree6()
 {
```

```
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode(seed);
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);

    JScrollPane scrollPane = new JScrollPane(tree);
    add(scrollPane);
  }


  /** Selected tree is unveiled */
  public String unveilTree()
  {
    DefaultMutableTreeNode selectedNode = null;
    TreePath selectedPath = tree.getSelectionPath();

    if (selectedPath == null) {
     selectedNode = rootNode;
    } else {
     selectedNode = (DefaultMutableTreeNode)
             (selectedPath.getLastPathComponent());
    }

    String tree =  selectedNode.toString();
    if (print)
    System.out.println("\nSelected tree at unveilTree() = "
                       + tree);
    return tree;
  }


  //Add a node
```

```
  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();
```

```
  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
System.out.println("ParentTree = " +  parentTree );

//===============

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
```

```
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                       + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
```

```
         * then the changed node is the child of the
         * node we've already gotten.
         * Otherwise, the changed node and the
         * specified node are the same.
         */

        int index = e.getChildIndices()[0];
        node = (DefaultMutableTreeNode)
                   (node.getChildAt(index));

        System.out.println(
          "The user has finished editing the node.");
        System.out.println("New value: " +
                    node.getUserObject());
      }

      public void treeNodesInserted(TreeModelEvent e)
      {
      }

      public void treeNodesRemoved(TreeModelEvent e)
      {
      }

      public void treeStructureChanged(TreeModelEvent e)
      {
      }
    }
}//End of DynamicTree6


//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
```

```
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick’s notation
//is included.

//Resultant trees are drawn.




class DrawBiTree extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane23;
JScrollPane scroller;




//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength;
```

```
  String  NwTree = "";


 boolean print = false;


//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.


//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];


//Set up the instructions.
JLabel instructionsLeft;

public DrawBiTree()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (Newick's notation) = " + simplify(selectedTree));

instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);
```

```
//Set up the drawing area.
drawingPane23 = new drawingPane23();
drawingPane23.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane23);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//========================================
//============= Painting class ============
//========================================


/** The component inside the scroll pane. */
 class drawingPane23 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
```

```
//Addresses are interpreted as paths
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane23.setPreferredSize(area);
```

```
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane23.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;
//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;
}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                              + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
```

```
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '0') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '1'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
```

```
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '0') //go up
 target.height = target.height + V;
 else // if direction == '1'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
```

```
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);



}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane23



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame =
   new JFrame("Tree (Link notation) = " + selectedTree);
```

```
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree

}//End of main class J150 TreeOfTrees3
```

155, page 331. Trees with links whose length takes values on real numbers.

```
//Program J155 TreeOfTrees5
//Beginning with tree ((A,B),C) = ((AI,BI),CI)
//we generate all possible topologies of trees.
//The resultant data structure is that of a tree.
//It is shown here.

//Topologies are generated
//and one can select one of them
//for visualization.
//Inner nodes also depicted.

//The corresponding tree is endowed with
//links of different length according to assignment
//at random.
//Lengths can  take on real values such as 123.456

//Topology = the graph of the tree without lengths
```

```
//Formats = Link and Newick's notations. And variants.
//You can learn these notations by looking at drawings.


import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class TreeOfTrees5 extends JPanel
                          implements ActionListener
{
```

```
private static final long serialVersionUID = 1L;



//Everything is built by recursion over this tree
private static   String   seed =
   "((Aa,Ba),Ca)";

private static  String leaves =
"ABCDEFGHIJKLMNOPRSTUVWXYZ";
private static String internalNodes =
"abcdefghijklmnopqrstuvwxyz";


private static boolean print = false;
private static boolean print2 = true;


//Number of leaves that must go in all trees
private static int numberOfLeaves = 6;


//Maximal allowed length
private static int maxLength = 10;


private static String[] Record =  new String[10000];
//Last filled position  in Record[]
private static int lastUsedPosition;
private static String selectedTree = "";

 //DynamicTree is an inner service class that comes
 //at the end of this file.
 private static DynamicTree8 treePanel;
 private static String VISUALIZE_COMMAND = "Visualize";

 private static Random r = new Random();
```

```
  //Constructor of main class: GUI basics
  public TreeOfTrees5()
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructionsLeft = new JLabel(
      "Select  a tree and click the Visualize button.");

   instructionPanel.setFocusable(true);
   instructionPanel.add(instructionsLeft);
   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree8();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(1200, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
     //Selected tree must be unveiled
     selectedTree = treePanel.unveilTree();
     System.out.println("\nSelected tree in Link notation"
      + " after unveilTree() = \n"   + selectedTree);
     selectedTree = addLenghts(selectedTree);
     System.out.println("\nSelected tree in " +
      "Link notation with random "
```

```
      + "lengths of Links = \n"   + selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      translate(selectedTree);
      DrawBiTree4 y = new DrawBiTree4();
      y.createAndShowGUI();
 }
  }

//======================


//Random lengths are added to a tree in Link notation
public static String addLenghts(String treeLinkN)
 {
  int l = leaves.length();

  //Lengths with digit values are generated
  /*
    int lengthsLeaves[]  = new int[l];
  //Last two entries are redundant
  int lengthsInnerNodes[]  = new int[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextInt(maxLength);
 lengthsInnerNodes[i] =  r.nextInt(maxLength);
   }
  */
  //Lengths with real values are generated
  double lengthsLeaves[]  = new double[l];
  //Last two entries are redundant
  double lengthsInnerNodes[]  = new double[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextDouble();
 lengthsInnerNodes[i] =  r.nextDouble();
   }
```

```
  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
  //Tree is scanned char by char
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    System.out.println(newTree.length() + " " + newTree);
    //Lengths are appended
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      double j = lengthsLeaves[k];
     newTree = newTree + ":" + j;
     }
    if (internalNodes.contains(thisChar))
    {
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerNodes[k];
     newTree = newTree + ":" + j;
    }
   }

   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
```

```
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Visual generator of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees5 newContentPane = new TreeOfTrees5();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}
```

```
//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
```

```
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
```

```
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}



  //Trees are generated
  private static void generateTrees()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }




//=============== Translation in within formats ============




//Translation from Newick's to quasi-binary notation
  //Inner nodes are lacked
```

```
private static String toBinary(String tree)
{
 String regulators = "(,)";
 //Output
 String treeToBi = "";
 //The binary address of leaf
 String address = "";
 //The name or content of the tree
 String content = "";
 String thisChar = "";
 String nextChar = "";
 String previousChar = "";
 String prepreviousChar = "";
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if ( i < l-1) nextChar = "" + tree.charAt(i+1);
   else nextChar = "end";
   if ( i > 0) previousChar = "" + tree.charAt(i-1);
   else previousChar = "start";
   //Is thisChar a regulator?
   if (regulators.contains(thisChar))
 {
   treeToBi =  treeToBi + thisChar;

   //This char = (
   if (thisChar.contentEquals("("))
   {
address = address + '+';
if (print)
System.out.println("Address = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
 {
```

```
  content = ""+ nextChar;
  treeToBi =  treeToBi + address + content;

}
   }


 //This char = ,
   if (thisChar.contentEquals(","))
   {
if (print)
System.out.println("Address at ,   = " + address);
int m = address.length();
//replace last 0 by a 1
address = address.substring(0, m-1) + '-';
if (print)
  System.out.println("Address at , 1 = " + address);
//If next char is a leaf
if (leaves.contains(nextChar))
  {

 content = ""+ nextChar;

 treeToBi =  treeToBi + address + content;

  }
   }


 //This char = )
  if (thisChar.contentEquals(")"))
   {
  prepreviousChar = "" + tree.charAt(i-2);
if ( (internalNodes.contains(previousChar)
& (leaves.contains(prepreviousChar))))
{
 content = "" + tree.charAt(i-2);
```

```
}
int m = address.length();
//Go one level down the tree
    address = address.substring(0, m-1);
   }


 } //End of   if (regulators.contains(thisChar))
   }//End of for(int i...
 return treeToBi;
}


//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;
```

```
counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
//Output
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
```

```
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "+";
```

```
   treeToBi =  treeToBi + address + thisChar;
   //if (m==1) treeToBi =   "0" + thisChar;
  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (numeric.contains(previousChar))
  address = address + '-';//Add a 1 to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}
```

```
//bTree in LL (links with lengths) notation
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}
```

```
if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method



//Tree is translated
//from link notation to binary.
//Output in Record[]: even entires= addresses, odd = names
public static void translate(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
 if (print)
 System.out.println("Tree  at translate = "    + tree);
 //String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
 System.out.println("Selected tree in " +
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }
```

```
//A vector with addresses and names is composed
makeInputInner(bTree);



 if (print)
 System.out.println("lastUsedPosition = "
                      + lastUsedPosition);
}



public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        createAndShowGUI();
        generateTrees();
      }
    });
  }




//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree8 extends JPanel
```

```
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree8()
 {
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode(seed);
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);

    JScrollPane scrollPane = new JScrollPane(tree);
    add(scrollPane);
  }


  /** Selected tree is unveiled */
  public String unveilTree()
  {
    DefaultMutableTreeNode selectedNode = null;
```

```
  TreePath selectedPath = tree.getSelectionPath();

  if (selectedPath == null) {
   selectedNode = rootNode;
  } else {
   selectedNode = (DefaultMutableTreeNode)
           (selectedPath.getLastPathComponent());
  }

  String tree =  selectedNode.toString();
  if (print)
  System.out.println("\nSelected tree at unveilTree() = "
                       + tree);
  return tree;
}


//Add a node
public DefaultMutableTreeNode
             addObject(DefaultMutableTreeNode parent,
    Object child, boolean shouldBeVisible) {
  DefaultMutableTreeNode childNode =
              new DefaultMutableTreeNode(child);

  if (parent == null) {
    parent = rootNode;
  }

  // It is key to invoke this on the TreeModel,
  //and NOT DefaultMutableTreeNode
  treeModel.insertNodeInto
            (childNode, parent, parent.getChildCount());

  // Make sure the user can see the lovely new node.
  if (shouldBeVisible) {
    tree.scrollPathToVisible
                  (new TreePath(childNode.getPath()));
  }
```

```
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
System.out.println("ParentTree = " +  parentTree );

//===============

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
```

```
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                       + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())
```

```
 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */

      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }
```

```
    public void treeNodesRemoved(TreeModelEvent e)
    {
    }

    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree8



//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
//is included.

//Resultant trees are drawn.




class DrawBiTree4 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane25;
JScrollPane scroller;
```

```
//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;



  String  NwTree = "";



 boolean print = false;



//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];
```

```
//Set up the instructions.
JLabel instructionsLeft;

public DrawBiTree4()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (pseudo-Newick's notation) = "
  + simplify(selectedTree));

instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);

//Set up the drawing area.
drawingPane25 = new drawingPane25();
drawingPane25.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane25);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//========================================
//============= Painting class ============
```

```
//=========================================


/** The component inside the scroll pane. */
 class drawingPane25 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
```

```
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane25.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane25.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}
```

```
//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                                 + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
```

```
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
```

```
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
```

```
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane25



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame =
   new JFrame("Tree (Link notation) = "
                   + simplify(selectedTree));
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree4();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree4
```

```
}//End of main class J155 TreeOfTrees5
```

157, page 331.

We know how to generate trees in various notations. Our aim now is to calculate the distance matrix associated to a specific tree. Actually, we need only the distances among leaves because our experimental data will be taken from living species that shall appear as leaves in our trees. To that aim, we shall discover first how to trace the path from one leaf to another.

Let us consider the following tree: ((Aa,(Bd,Fd)a),(Cb,(Dc,Ec)b)a).

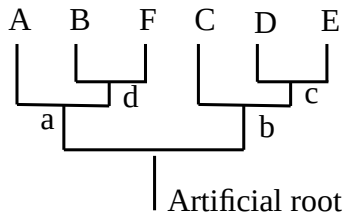


Figure 6.1. This is the tree ((Aa,(Bd,Fd)a),(Cb,(Dc,Ec)b)a). Can we discover from this string representation how to find the path from one leaf to another?

It seems easy to find the path from a leaf to node *a*. And also to find the path backwards:

```
Aa aA
Bda adB
Fda adF
Cba abC
Dcba abcD
Ecba abcE
```

To find the path from a leaf to another, we just glue the initial path to the final back in backwards order. Say, let us find the path from A to B. The answer must be AdB: the initial path is Aa, the target path is adB. If we concatenate these paths, we get AaadB that must be equal to AdB. We see that this can be done by a rule:

aa $->$ nothing.

The very same grammar serves us to find the path from C to E:

CbaabcE = CbbcE = CcE.

Nevertheless, this rule is not suitable to find the path from A to E. The answer must be AabcE. But if we concatenate the two paths, we get: AaabcE, which would be simplified into AbcE and we lack an *a*. So, when the two paths to be concatenated belong in different sectors of the (artificial) root, we need another rule:

aa $->$ a

Thus, we need an indicative of the sector of the root to which the leaf belongs. With this information, our paths from leaves to node *a* look like this:

```
Aa+ +aA
Bda+ +adB
Fda+ +adF
Cba-  -abC
Dcba-  -abcD
Ecba-  -abcE
```

Let us find the path from A to B. The answer must be AdB: the initial path is Aa+, the target path is +adB. If we concatenate these paths, we get Aa++adB that must be equal to AdB. We see that this can be done by the rule:

aa $->$ nothing.

The very same rule serves us to find the path from C to E:

Cba–abcE = CbaabcE = CbbcE = CcE.

Let us find the path from A to E. The answer must be AabcE. If we concatenate the corresponding two paths, we get: Aa+-abcE. We are done if we use the rule

a+-a $->$ a

Therefore, we are invited to try out the grammar composed of two rules:

aa $->$ nothing (two repeated symbols annihilate one another)
a+-a $->$ a (indicatives of different sectors conserve a copy of the surrounding symbol)).

The corresponding implementation of these ideas will be done step by step. As to the first one, the next code outputs the path to internal node a from each leaf.

```
//Program J157 TreeOfTrees6
//Beginning with tree ((A,B),C) = ((AI,BI),CI)
//we generate all possible topologies of trees.
//The resultant data structure is that of a tree.
//It is shown here.

//Topologies are generated
//and one can select one of them
//for visualization.
//Inner nodes are also depicted.

//The corresponding tree is endowed with
//links of different length according to assignment
//at random.
//Lengths can  take on real values such as 123.456

//Path  to  node a is calculated for each leaf.

//Topology = the graph of the tree without lengths

//Formats = Link and Newick's notations. And variants.
//You can learn these notations by looking at drawings.


import java.awt.BorderLayout;
```

```
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class TreeOfTrees6 extends JPanel
                          implements ActionListener
{

 private static final long serialVersionUID = 1L;



 //Everything is built by recursion over this tree
 private static   String   seed =
```

```
   "((Aa,Ba),Ca)";

private static  String leaves =
"ABCDEFGHIJKLMNOPRSTUVWXYZ";
private static String internalNodes =
"abcdefghijklmnopqrstuvwxyz";


private static boolean print = false;
private static boolean print2 = true;


//Number of leaves that must go in all trees
private static int numberOfLeaves = 6;




private static String[] Record =  new String[10000];
//Last filled position  in Record[]
private static int lastUsedPosition;
private static String selectedTree = "";


 //DynamicTree is an inner service class that comes
 //at the end of this file.
 private static DynamicTree9 treePanel;
 private static String VISUALIZE_COMMAND = "Visualize";

 private static Random r = new Random();

 //Constructor of main class: GUI basics
 public TreeOfTrees6()
 {
  super(new BorderLayout());
  JPanel instructionPanel = new JPanel(new GridLayout(0,1));
  JLabel instructionsLeft = new JLabel(
     "Select  a tree and click the Visualize button.");
```

```
   instructionPanel.setFocusable(true);
   instructionPanel.add(instructionsLeft);
   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree9();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(1200, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
      //Selected tree must be unveiled
      selectedTree = treePanel.unveilTree();
      System.out.println("\nSelected tree in Link notation"
       + " after unveilTree() = \n"   + selectedTree);
      //Lengths are added
      selectedTree = addLenghts(selectedTree);
      System.out.println("\nSelected tree in " +
       "Link notation with random "
      + "lengths of Links = \n"   + selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      translate(selectedTree);
      DrawBiTree5 y = new DrawBiTree5();
      y.createAndShowGUI();
```

```
 }
  }

//======================


//Random lengths are added to a tree in Link notation
public static String addLenghts(String treeLinkN)
 {
  int l = leaves.length();

  //Lengths with digit values are generated
  /*
    int lengthsLeaves[]  = new int[l];
  //Last two entries are redundant
  int lengthsInnerNodes[]  = new int[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextInt(maxLength);
 lengthsInnerNodes[i] =  r.nextInt(maxLength);
   }
  */
  //Lengths with real values are generated
  double lengthsLeaves[]  = new double[l];
  //Last two entries are redundant
  double lengthsInnerNodes[]  = new double[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextDouble();
 lengthsInnerNodes[i] =  r.nextDouble();
   }


  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
  //Tree is scanned char by char
```

```
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    if (print)
    System.out.println(newTree.length() + " " + newTree);
    //Lengths are appended
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      double j = lengthsLeaves[k];
     newTree = newTree + ":" + j;
     }
    if (internalNodes.contains(thisChar))
    {
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerNodes[k];
     newTree = newTree + ":" + j;
    }
   }

   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
```

```
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Visual generator of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
  TreeOfTrees6 newContentPane = new TreeOfTrees6();
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
```

```
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
```

```
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
```

```
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}

  //Trees are generated
  private static void generateTrees()
  {

//New leaves are incorporated one by one
for(int i = 0; i < numberOfLeaves - 3; i++)
 {
if (print)
System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
 }
  }



//=============== Translation in within formats ============



//bTree in (modified) Newick’s notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput /////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
```

```
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Outputs a word in reversed order.
private static String reverser(String word)
{
 return new StringBuilder(word).reverse().toString();
}


private static String cleanTree(String tree)
```

```
{
 System.out.println("Input tree at deleteNumerics = \n "
        + tree);
 String treeClean = "";
 String numeric = ".:0123456789";
 String plusMinus = "+-";
 String thisChar = "";
  boolean leaf = false;
 //Numerics are washed
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if (numeric.contains(thisChar));
    else treeClean = treeClean + thisChar;
  }
 System.out.println("Tree without numerics = \n"
+ treeClean);
 //+/- are deleted from inner nodes that accompany leaves
 l = treeClean.length();
 String treeOut = "";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
 {
  thisChar = "" + treeClean.charAt(i);
  if (leaves.contains(thisChar))
  leaf  = true;
  if ((leaf) &  (plusMinus.contains(thisChar)));
else treeOut = treeOut + thisChar;
  if (internalNodes.contains(thisChar))
  leaf  = false;
}

System.out.println("Tree at output of cleanTree = \n"
+ treeOut);
return treeOut;
```

```
}



//The path from each leaf to node a is found.
//The input contains branch's indicator, say:
//((++Aa,+-Ba),(-+Cb,((--++Dd,--+-Fd)--c,---Ec)-b)+a)
//Output for leaf A: Aa+.
//Output for leaf C: Cba-
//Output for leaf D: Ddcba-
private static String findPaths(String tree)
{
 System.out.println("Input tree at findPaths = \n" + tree);
 //length of String   tree
 int l = tree.length();
 String plusMinus = "+-";
 String thisChar;
 String nextChar;
 String path = "";
 //Branch indicative with respect to root, + else -
 String branch;
 String address ;
 //The number of links of a path to root
 int level = 0;
 String treeReversed = reverser(tree);
 System.out.println("Reversed tree = " + treeReversed);

//Each leaf has a path to a
for(int k= 0; k < leaves.length(); k++)
{
 String leaf = "" + leaves.charAt(k);
 //Candidate nodes to be part of the path to node a
 //must have  sign +/- as prefix.
 //Or as suffix in the reversed tree.
 int position = treeReversed.indexOf(leaf);
 if (position > -1) // if leaf is in tree
```

```
 {
  System.out.println("\nLeaf = " + leaf );
  address = "";
  boolean go = true;
  //branchIndicator and level are found
  for(int i = position+1 ; ((i < l) & (go)); i++ )
  {
   thisChar = "" + treeReversed.charAt(i);
   if   (plusMinus.contains(thisChar)  )
address = thisChar + address;
   else go = false;
  }
  level = address.length();
  branch = "" + address.charAt(0);
  if (print)
  System.out.println("Address  = " + leaf + address + level);



  //Path from leaf to node a is found in (not reversed) tree
  position = tree.indexOf(leaf);
  thisChar = "" + tree.charAt(position);
  nextChar = "" + tree.charAt(position + 1);
  //Output
  path = "" + leaf + nextChar;
  go = true;
  if  (nextChar.contentEquals("a") ) //leaf is neighbor of a
   {
  path = path + branch;
  if (print)
  System.out.println("Path short = " + path);
   }
  else  //level > 1
  {



  //Tree is scanned char by char to get path
```

```
  String iChar = "";
  for(int i = position + 2; ((i < l) & (go)); i++ )
  {
   iChar = "" + tree.charAt(i);


   int newLevel = 0;
   if (internalNodes.contains(iChar))//InternalNodes
{
     if   (iChar.contentEquals("a")  )
      {
   path  = path +  iChar;
   go = false; //end
      }
     else //Internal node different than a
     {
      if (print)
      System.out.println("Internal node not a) = " + iChar );
      int rP = l -i-1; //index of iChar in treeReversed
      String  nextNewChar = "" + treeReversed.charAt(rP+1);

      if ( plusMinus.contains(nextNewChar )) //candidate node
       {
       if (print)
     System.out.println("next candidate = " + iChar);
        String addressNew = "";

        //level of candidate node iChar  is inquired
        String thatChar = "";
        boolean go2 = true;
        for(int j = rP+1 ; ((j < l) & (go2)); j++ )
         {
       thatChar = "" + treeReversed.charAt(j);
       if   (plusMinus.contains(thatChar)  )
          addressNew = thatChar + addressNew;
       else go2 = false;
         }
        if (print)
```

```
        System.out.println("AddressNew " + iChar + addressNew);
       newLevel = addressNew.length();
       //A path links a node with another in a lower level
       if  ( (newLevel < level) &
        (!(nextChar.contentEquals(iChar))))
        {
       path = path + iChar;
       level = newLevel;
       if (print)
       System.out.println("Path newLevel= " + path);
        }
      } //End of if candidate node
     }//End of else Internal node different than a
}//End of if InternalNodes
   }//End of Tree is scanned char by char
  path = path + branch;
 }//End of  else level > 1
   System.out.println("Path completed  = " + path);
 }//End of if leaf is in tree
}//End of each leaf
return path;
}



//Paths from leaves to node a are found.
//Input tree in quasibinary notation with addresses and lengths
private static void pathFinder(String tree)
{
 //useless information is deleted
 tree = cleanTree(tree);
 //Add an indicator of the branch
 //with respect to the root: up(+) else down(-)
  tree =  findPaths(tree);
}
```

```
//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
System.out.println("Input tree at toQuasiBinaryInner = \n" + tre
//Output
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }
```

```
 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "+";
   treeToBi =  treeToBi + address + thisChar;

  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (numeric.contains(previousChar))
  address = address + '-';//Add a - to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }
```

```
 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree in LL (links with lengths) notation
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
```

```
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method



//Tree is translated
//from link notation to binary.
//Output in Record[]: even entires= addresses, odd = names
public static void translate(String tree)
{
 /*
```

```
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
 if (print)
 System.out.println("Tree  at translate = "    + tree);
 //String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
  System.out.println("Selected tree in " +
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

 //A vector with addresses and names is composed
 makeInputInner(bTree);
 //Paths from leaves to node a are found.
 //Reverse paths are also listed.
 pathFinder(bTree);

 if (print)
 System.out.println("lastUsedPosition = "
                      + lastUsedPosition);
}



public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
```

```
    {
      public void run()
      {
        createAndShowGUI();
        generateTrees();
      }
    });
  }
```

```
//=========================================================
//==================Inner service class====================
//=========================================================


class DynamicTree9 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree9()
 {
```

```
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode(seed);
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);

    JScrollPane scrollPane = new JScrollPane(tree);
    add(scrollPane);
  }


  /** Selected tree is unveiled */
  public String unveilTree()
  {
    DefaultMutableTreeNode selectedNode = null;
    TreePath selectedPath = tree.getSelectionPath();

    if (selectedPath == null) {
     selectedNode = rootNode;
    } else {
     selectedNode = (DefaultMutableTreeNode)
             (selectedPath.getLastPathComponent());
    }

    String tree =  selectedNode.toString();
    if (print)
    System.out.println("\nSelected tree at unveilTree() = "
                         + tree);
    return tree;
  }


  //Add a node
```

```
  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
                new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }


 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();


while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();
```

```
  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
System.out.println("ParentTree = " +  parentTree );

//=================

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
```

```
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);

   }
 System.out.println(newTree + " -> "
                       + simplify(newTree));
String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
```

```
         * then the changed node is the child of the
         * node we've already gotten.
         * Otherwise, the changed node and the
         * specified node are the same.
         */

        int index = e.getChildIndices()[0];
        node = (DefaultMutableTreeNode)
                   (node.getChildAt(index));

        System.out.println(
          "The user has finished editing the node.");
        System.out.println("New value: " +
                    node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }

    public void treeNodesRemoved(TreeModelEvent e)
    {
    }

    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree9



//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
```

```
//denotes a tree in Newick’s notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick’s notation
//is included.

//Resultant trees are drawn.




class DrawBiTree5 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane26;
JScrollPane scroller;




//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;
```

```
  String  NwTree = "";


 boolean print = false;


//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.


//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];


//Set up the instructions.
JLabel instructionsLeft;

public DrawBiTree5()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));

//Set up the instructions.
instructionsLeft = new JLabel(
  "Tree (pseudo-Newick's notation) = "
  + simplify(selectedTree));
```

```
instructionPanel.setFocusable(true);
instructionPanel.add(instructionsLeft);

//Set up the drawing area.
drawingPane26 = new drawingPane26();
drawingPane26.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane26);
scroller.setPreferredSize(new Dimension(500,500));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
 class drawingPane26 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
```

```
int H;
int V;
//Addresses are interpreted as paths
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);
```

```
//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,10*verticalSize);
//area is set as the preferred size of the graphic
drawingPane26.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane26.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 800;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                                + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
```

```
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
```

```
{
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}
```

```
//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane26


/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
```

```
JFrame frame =
   new JFrame("Tree (Link notation) = "
                  + simplify(selectedTree));
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree5();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree5

}//End of main class J157 TreeOfTrees6
```

164, page 420. Our aim was to test the basic requirement that the tree that best fits the distance matrix associated to a tree must be the tree itself. The Author found many examples of not perfect matching but all of them obey a simple rule: they exploit the freedom given by our encoding and depicted by the following drawing:

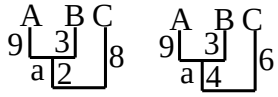


Figure 6.2. These two trees are different but generate the very same distance matrix because our matrices does not relate every pair of nodes but only pair of leaves. This causes a freedom represented in these trees by the equation 2 + 8 = 7 + 3. As a consequence, the solution to our requirement is unique but only up to this freedom.

Formally: if we begin with a distance matrix associated to a tree, *D*, to next find the tree that best fits *D* and calculate its associated distance matrix,

M then, we always have $M = D$.

Besides, approximate solutions are rapidly found but exact matching demands too many generations: for 5 leaves 50000 generations were enough.

168, page 420.

The following code improves the GUI performance: the tree that best fits the target matrix (associated to target tree) is shown and the best fitting tree is shown as response to the selection of a topology by the User. The code follows:

```
//Program J168 EvoTree2
/*
 * We generate a random tree topology
 * and attach random integer lengths to links.
 * In that way we get the target tree, T.
 * Next, we find the associated distance matrix D  to T.
 * Our problem is:
 *
 * Find tree t such that the absolute distance between
 * matrix D and the associated matrix to t is a minimum.
 * Absolute distance between M and N = sum |Mij-Nij|.
 *
 * We run the next test:
 * If we begin with a tree T, compute D, and then t,
 * we expect to return with the initial tree: t = T.
 *
 * To associate a distance matrix to a tree,
 * we calculate the distance in within
 * each pair of nodes (leaves + inner nodes).
 * To that aim, we begin with the first node and
 * walk over the tree until the second node
 * and add the lengths of all traversed links.
 * The resulting number is the  distance.
 *
 * To find the tree that best fit D,
 * we  scan all possible topologies,
 * each one of which is submitted to evolution
 * to finely tune lengths.
 *
```

```
 * Formats = Link and Newick’s notations. And variants.
 *
 * The performance of the GUI has been improved.
 */



import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class EvoTree2 extends JPanel
                          implements ActionListener
```

```
{

 private static final long serialVersionUID = 1L;



 //Everything is built by recursion over this tree
 private static   String   seed =
    "((Aa,Ba),Ca)";

 private static  String leaves =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 private static String internalNodes =
"abcdefghijklmnopqrstuvwxyz";

 //Next leaf to be grafted
 private static  int nextLeafIndex = 2;
 //Next internal node to be inserted
 private static  int nextIntNodeIndex = 0;


 static int maxNumberOfLeaves = leaves.length();
 //Number of leaves that must go in all trees > 3
 private static int numberOfLeaves = 5;

 private static String[] Record =  new String[10000];

 private static  int indexOfTopologies;
 //The distance to target of the best fitting tree
 //of the given topology
 private static double[] bestFitting =  new double[10000];
 private static String[] bestTreeOfTopology
                             = new String[10000];

 //Last filled position  in Record[]
 private static int lastUsedPosition;
 private static String selectedTree = "";
```

```
  //DynamicTree is an inner service class that comes
  //at the end of this file.
  private static DynamicTree12 treePanel;
  private static String VISUALIZE_COMMAND = "Visualize";

   //Length of the link that glues a leaf to the tree
  private static int lengthsLeaves[]  =
                new int[numberOfLeaves];
  //Length of the link that glues
  //an inner node to another one
  private static int lengthsInnerLinks[]  =
              new int[numberOfLeaves];


  private static Random r = new Random();
  private static int maxLength = 10;

  //Paths from each leaf to node a
  private static String[] Paths =
              new String[numberOfLeaves];

  private static String targetTree;
  private static double targetMatrix[][] =
     new double[numberOfLeaves][numberOfLeaves];


  //=======Evolution parameters============


//Population to evolve =
//lengths of links that attach leaves to tree
// + lengths of inner links.
//N individuals at all.

static private int N = 20;
static private double oldLengths[][][] =
```

```
 new double[N][2][maxNumberOfLeaves];

static private double newLengths[][][] =
new double[N][2][maxNumberOfLeaves];

static private double distance[] = new double[N];
static private double fitness[] = new double[N];

static private int numberOfGenerations = 2;

//Mutation rate per site (= entry) per generation
static private double    m = 0.5;

static private int best;
//The best tree for a given topology
static private String bestTree = "";

//=========Debugging help ===============

private static boolean print  = false;
private static boolean print2 = false;
private static boolean print3 = false;
private static boolean print4 = false;
//Turn on print5 firstly
private static boolean print5 = true;

//==============

private static String titleFrame = "";
private static String title1 = "";
private static String title2 = "";
private static EvoTree2 newContentPane = new EvoTree2();



  //=======Constructor of main class: GUI basics=====

  public EvoTree2()
```

```
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructions1 = new JLabel(
      "Select  a tree and click the Visualize button.");


   instructionPanel.setFocusable(true);
   instructionPanel.add(instructions1);

   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree12();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(700, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
     //Selected tree must be unveiled
     selectedTree = treePanel.unveilTree();
     System.out.println("\nSelected tree in Link notation"
      + " after unveilTree() = \n"   + selectedTree);
     /*Random lengths are added to topology
     makeLengths();
     //Lengths are added
```

```
      selectedTree = insertLengths(selectedTree);
      */
      int i =  findTheBest(selectedTree);
      selectedTree =  bestTreeOfTopology[i];
      double d = bestFitting[i];
      System.out.println( " Best fitting = \n " +
                                  selectedTree);
      System.out.println( " Best distance = \n "
       + d);


      decoderLengths(selectedTree);
  //Translation from one format into another
      //output = a vector with binary addresses and contents
      double[][] M =  associatedMatrix(selectedTree);
      printMatrix(M, "distance matrix associated to the " +
      "best tree for this topology");
      //Selected tree is drawn
      title1 = "Best tree for the topology = "
       + simplify(selectedTree);
      title2 = "Distance to target = " + d;
      DrawBiTree9 a = newContentPane.new DrawBiTree9();
      a.createAndShowGUI();

 }
  }

//======================



public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
```

```
     {
    String tree = randomTopology();
    makeLengths();
    targetTree = insertLengths(tree);
    System.out.println("Target tree: each link " +
    "has its own length = \n" + targetTree);
    targetMatrix = associatedMatrix(targetTree);

       createAndShowGUI();



       System.out.println("\nDistance to target = " +
         " Absolute distance between" +
        " target matrix  and \n" +
        "the distance matrix associated to " +
        "the tree with lengths of links that \n" +
        "is the best fitting  of the topology.\n" +
        "Perfect fitting: distance = 0 \n");
       System.out.println("\nNumber of Generations = "
           + numberOfGenerations);
       //To find the best fitting, all tree topologies are
       //scanned and each one of them is evolved for
       //the best particular fitting.
       //All this is done along the process of
       //generating all topologies
       generateTopologies();
       //Draw Target tree: this malfunctions.
       /*
       titleFrame = "Target tree = " + tree;
       selectedTree = targetTree;
       DrawBiTree9 b = newContentPane.new DrawBiTree9();
       b.createAndShowGUI();
       */

     }
   });
}
```

```
//From parentTree, a tree in Newick's notation,
//a Child  tree is generated by a generative grammar.
//A random election is made among all possibilities.
private static String generateChild(String parentTree)
{

 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);


 //link to be replaced
 String link = "";
 //String link is replaced by what:
 String replacement = "";
 //new tree generated by replacement
 String newTree = "";
 //length of parent tree
 int l = parentTree.length();
 //counter of links = 0;
 int counter = 0;
 //Links are counted
 for(int i = 0; i < l; i++ )
  {
   //parentTree is scanned char by char
   char I = parentTree.charAt(i);
 //Is I an internal node that denotes a link?
 if (charIsInternalNode(I)) counter++;
  }
 //A link is chosen
 int k = r.nextInt(counter);
 if (print2)
```

```
System.out.println("Random number = " + k);
//new branch is grafted at k
counter = 0;
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  if (counter == k) //Is I the chosen link?
   {
//Check for the char before I
char a = parentTree.charAt(i-1);
//If I links to a complex structure
if (a == ')') newTree = useRule2(parentTree, i,
               nextLeaf, nextIntNode);
else //if I links to a leaf
 {
  //Link to be replaced
  link = "" + a + I;
  //Replacement rule 1:
  //aI -> (aN,FN)I;   N= nextIntNode;
  //F = nextLeaf
 replacement = "(" + a + nextIntNode + ","
          + nextLeaf + nextIntNode + ")" + I;
 //Rule 1 is executed
 newTree = parentTree.replace(link, replacement);
 }
if (print2)
System.out.println(newTree + " -> "
                      + simplify(newTree));

 i = l;
   }
  counter++;
}
 }
```

```
  return newTree;
 }


 //A tree topology is generated at random.
 public static String randomTopology()
 {
  if (print2)
  System.out.println("Parent tree = \n" + seed
   + " -> "    + simplify(seed));
  String tree = seed;
  int limit = numberOfLeaves-3; //less than 11.
  for(int i = 0; i < limit; i++ )
  tree = generateChild(tree);
  System.out.println("Target topology = \n" + tree);
  return tree;
 }


//======================


//Random lengths are added to a tree in Link notation
public static void makeLengths()
 {
  //Lengths with digit values are generated
  for(int i = 0; i < numberOfLeaves; i++)
   {
 //Zero lengths are not allowed
 lengthsLeaves[i] = 1 + r.nextInt(maxLength - 1);
 lengthsInnerLinks[i] = 1 +  r.nextInt(maxLength - 1);
   }
 }


//The best fitting tree for chosen topology is found
public static int findTheBest(String tree)
{
```

```
 String selectedTopology = simplify(tree);
 String thisTopology = "";
 String thisTree = "";
 int bestPosition = -1;

for(int i = 0; i < indexOfTopologies; i++)
 {
  thisTree = bestTreeOfTopology[i];
  thisTopology = simplify(thisTree);
  if (selectedTopology.equals(thisTopology))
  {
   bestPosition = i;
   return bestPosition;
  }
 }
return  bestPosition;
}


  /*
  //Lengths with real values are generated
  double lengthsLeaves[]  = new double[l];
  //Last two entries are redundant
  double lengthsInnerLinks[]  = new double[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextDouble();
 lengthsInnerLinks[i] =  r.nextDouble();
   }
  */

//Random lengths are added to a tree in Link notation
public static String insertLengths(String treeLinkN)
 {
  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
```

```
  //Tree is scanned char by char
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    if (print)
    System.out.println(newTree.length() + " " + newTree);
    //Lengths are appended
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      double j = lengthsLeaves[k];
     newTree = newTree + ":" + j;
     }
    if (internalNodes.contains(thisChar))
    {
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerLinks[k];
     newTree = newTree + ":" + j;
    }
   }
   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
```

```
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Tree of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.

  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
```

```
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
```

```
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
```

```
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}

//Input: string tree with complete information.
//Work: lengthsLeaves[] and lengthsInnerLinks[]
//are read from tree.
private static void decoderLengths(String tree)
{
 if(print2)
 System.out.println(
"Input tree at recoverLengths = \n "   + tree  );
 //Delete :
 tree = tree.replaceAll(":", "");
 if(print2)
 System.out.println(
"Input tree without : = \n "   + tree  );
 int n = tree.length();
 String thisChar = "";
 String nextChar = "";
 String numbers = ".0123456789";
 boolean leaf = false;
 boolean node = false;
 String Leaf = "";
 String numberLeaf = "";
 String numberNode = "";
 String Node = "";
 int j ;
 double f;

//All chars in tree are scanned
 for(int i=0; i < n; i++)
  {
   thisChar = "" + tree.charAt(i);
   if (i < n-1) nextChar  = "" + tree.charAt(i+1);
   //Do we have a leaf?
```

```
   if (leaves.contains(thisChar))
    {
 leaf = true;
 Leaf = "" + thisChar;
    }

   //Do we have an internal node?
   if (internalNodes.contains(thisChar))
   {
 node = true;
 Node = "" + thisChar;
   }

   //Do we have a number that belongs in a leaf?
   if ( (leaf) & (numbers.contains(thisChar)))
{
 numberLeaf = numberLeaf + thisChar;
}

  //Do we have a number that belongs in a leaf?
   if ( (node) & (numbers.contains(thisChar)))
{
 numberNode = numberNode + thisChar;
}

   //Must we end accumulation of numbers of a leaf?
   if ( (leaf) & (internalNodes.contains(nextChar)))
{
  leaf = false;
  j = leaves.indexOf(Leaf);
  if (print4)
  System.out.println("Leaf = " + Leaf);
  f = new Double(numberLeaf);
  lengthsLeaves[j] = (int) Math.floor(f);
  numberLeaf = "";
}

   //Must we end accumulation of numbers of a node?
```

```
   if ( (node) & (!(numbers.contains(nextChar))))
{
  node = false;
  j = internalNodes.indexOf(Node);
  if (print4)
  System.out.println("Node = " + Node);
  f = new Double(numberNode);
  lengthsInnerLinks[j] = (int) Math.floor(f);
  numberNode = "";
}

  }
}


//Topologies are generated and
//the best fitting is filtered
private static void generateTopologies()
 {
  //New leaves are incorporated one by one
  for(int i = 0; i < numberOfLeaves - 3; i++)
   {
    if (print) System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
   }
  //The best topology is found
  double minD = bestFitting[0];
  int place = 0;
  for(int i = 1; i <   indexOfTopologies; i++)
 {
   if ( bestFitting[i] < minD )
{
 minD =  bestFitting[i];
     place = i;
}
 }

  System.out.println("================\n================\n");
```

```
  System.out.println("TARGET TOPOLOGY = \n" +
  simplify(targetTree));
  System.out.println("TARGET TREE = \n" + targetTree);
  printMatrix(targetMatrix, "Distance Matrix   " +
   "associated to the leaves  of target tree");
  String theBestTree = bestTreeOfTopology[place];
  System.out.println("\nBEST FITTING TOPOLOGY = \n" +
  simplify(theBestTree));
  System.out.println(
    "\nBEST DISTANCE AND FITTING TREE TO TARGET MATRIX  "
    + minD + "\n"    + theBestTree);

  //Distance from M(bestTree)
  //to target matrix is evaluated



  decoderLengths(theBestTree);
  double M[][] = associatedMatrix(theBestTree);
  double d= absDistance(numberOfLeaves,
             targetMatrix, M);
  printMatrix(M, "Distance Matrix associated to " +
                    "the leaves of the best tree");
  System.out.println("Distance to target matrix = "   + d);
  //Draw theBestTree
  titleFrame = "Best fitting tree";
  title1 =  simplify(theBestTree);
  selectedTree = theBestTree;
  DrawBiTree9 c = newContentPane.new DrawBiTree9();
  c.createAndShowGUI();
  c.setLocation(500,  100);
 }



//========= Translation in within formats ==========
```

```
//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput ////// " + bTree);
 int counter = 0;
 //length of String   tree
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);
```

```
lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Outputs a word in reversed order.
private static String reverser(String word)
{
 return new StringBuilder(word).reverse().toString();
}


private static String cleanTree(String tree)
{
 if (print2)
 System.out.println("Input tree at deleteNumerics = \n "
        + tree);
 String treeClean = "";
 String numeric = ".:0123456789";
 String plusMinus = "+-";
 String thisChar = "";
  boolean leaf = false;
 //Numerics are washed
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if (numeric.contains(thisChar));
    else treeClean = treeClean + thisChar;
  }
 if (print2)
 System.out.println("Tree without numerics = \n"
+ treeClean);
```

```
 //+/- are deleted from inner nodes that accompany leaves
 l = treeClean.length();
 String treeOut = "";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
 {
  thisChar = "" + treeClean.charAt(i);
  if (leaves.contains(thisChar))
  leaf  = true;
  if ((leaf) &  (plusMinus.contains(thisChar)));
else treeOut = treeOut + thisChar;
  if (internalNodes.contains(thisChar))
  leaf  = false;
}

 if (print2)
 System.out.println("Tree at output of cleanTree = \n"
+ treeOut);
return treeOut;

}



//The path from each leaf to node a is found.
//The input contains branch's indicator, say:
//((++Aa,+-Ba),(-+Cb,((--++Dd,--+-Fd)--c,---Ec)-b)+a)
//Output for leaf A: Aa+.
//Output for leaf C: Cba-
//Output for leaf D: Ddcba-
private static void findPaths(String tree)
{
 if (print2)
 System.out.println("Input tree at findPaths = \n" + tree);
 //length of String   tree
 int l = tree.length();
 String plusMinus = "+-";
```

```
 String thisChar;
 String nextChar;
 String path = "";
 //Branch indicative with respect to root, + else -
 String branch;
 String address ;
 //The number of links of a path to root
 int level = 0;
 String treeReversed = reverser(tree);
 if (print2)
 System.out.println("Reversed tree = " + treeReversed);

//Each leaf has a path to a
for(int k= 0; k < numberOfLeaves; k++)
{
 String leaf = "" + leaves.charAt(k);
 //Candidate nodes to be part of the path to node a
 //must have  sign +/- as prefix.
 //Or as suffix in the reversed tree.
 int position = treeReversed.indexOf(leaf);
 if (position > -1) // if leaf is in tree
 {
  if (print2)
  System.out.println("\nLeaf = " + leaf );
  address = "";
  boolean go = true;
  //branchIndicator and level are found
  for(int i = position+1 ; ((i < l) & (go)); i++ )
  {
   thisChar = "" + treeReversed.charAt(i);
   if   (plusMinus.contains(thisChar)  )
address = thisChar + address;
   else go = false;
  }
  level = address.length();
  if (print2)
  System.out.println("LeafAddressLevel  = " + leaf
                     + address + level);
```

```
  branch = "" + address.charAt(0);


  //Path from leaf to node a is found in (not reversed) tree
  position = tree.indexOf(leaf);
  thisChar = "" + tree.charAt(position);
  nextChar = "" + tree.charAt(position + 1);
  //Output
  path = "" + leaf + nextChar;
  go = true;
  if  (nextChar.contentEquals("a") ) //leaf is neighbor of a
   {
  path = path + branch;
  if (print)
  System.out.println("Path short = " + path);
   }
  else  //level > 1
  {

  //Tree is scanned char by char to get path
  String iChar = "";
  for(int i = position + 2; ((i < l) & (go)); i++ )
  {
   iChar = "" + tree.charAt(i);


   int newLevel = 0;
   if (internalNodes.contains(iChar))//InternalNodes
{
     if   (iChar.contentEquals("a")  )
      {
   path  = path +  iChar;
   go = false; //end
      }
     else //Internal node different than a
     {
      if (print)
      System.out.println("Internal node not a) = " + iChar );
```

```
       int rP = l -i-1; //index of iChar in treeReversed
       String  nextNewChar = "" + treeReversed.charAt(rP+1);

       if ( plusMinus.contains(nextNewChar )) //candidate node
        {
        if (print)
      System.out.println("next candidate = " + iChar);
         String addressNew = "";

         //level of candidate node iChar  is inquired
         String thatChar = "";
         boolean go2 = true;
         for(int j = rP+1 ; ((j < l) & (go2)); j++ )
          {
        thatChar = "" + treeReversed.charAt(j);
        if   (plusMinus.contains(thatChar)  )
           addressNew = thatChar + addressNew;
        else go2 = false;
          }
         if (print)
         System.out.println("AddressNew " +
                                      iChar + addressNew);
        newLevel = addressNew.length();
        //A path links a node with another in a lower level
        if  ( (newLevel < level) &
         (!(nextChar.contentEquals(iChar))))
         {
        path = path + iChar;
        level = newLevel;
        if (print)
        System.out.println("Path newLevel= " + path);
         }
       } //End of if candidate node
      }//End of else Internal node different than a
}//End of if InternalNodes
   }//End of Tree is scanned char by char

  path = path + branch;
```

```
 }//End of  else level > 1
 }//End of if leaf is in tree
 if (print2)
 System.out.println("Path completed  = " + path);
 Paths[k] = path;
}//End of each leaf
}



//Paths from leaves to node a are found
//Reverse paths are also calculated.
//Input tree in quasibinary notation
//with addresses and lengths
private static void pathFinder(String tree)
{
 //Useless information is deleted
 tree = cleanTree(tree);
 //Find the path to node a from each leaf.
 //Add an indicator of the branch
 //with respect to the root: up(+) else down(-)
  findPaths(tree);
}

//Input: a long raw string with path from leaf to node a
//plus path from node a to leaf.
//Editing is done to get true paths.
private static String grammar (String path)
{
 int n = path.length();
 String thisChar = "";
 String nextChar = "";
 String oldPath = path;
 String plusMinus = "a+-a";
 String minusPlus = "a-+a";
 String two = "";
 int N = path.length();
```

```
//Rule 1: two equal symbols produce nothing
 for(int j=1; j < N; j++)
  {
   for(int i=1; i < n-1;)
   {
    thisChar = "" + path.charAt(i);
    nextChar = "" + path.charAt(i+1);
    two = "" + thisChar + nextChar;
    if (thisChar.contentEquals(nextChar))
  {
   i = i + 2;
   path = path.replace(two, "");
  }
    else
  {
   i = i+1;
  }
  n = path.length();
   }
  }

 //Rule 2: a+-a -> a

 if  (path.contains(plusMinus))
      path = path.replace(plusMinus, "a");
 if   (path.contains(minusPlus))
     path = path.replace(minusPlus, "a");
 if (print2)
 System.out.println("input = " + oldPath +
                    " output = " + path);
 return path;
}

//The distance matrix among leaves is calculated
private static double[][] matrixBuilder()
{
double M[][] =
new double[numberOfLeaves][numberOfLeaves];
```

```
 //Path to node a from leaf
 String pathLeafToa = "";
 //Path to leaf from node a
 String pathaToLeaf = "";
 //Path from leaf to lead
 String pathLeafiToLeafj = "";
 String innerNode = "";
 double distance;
 for(int i = 0; i < numberOfLeaves; i++)
  for(int j = 0; j < numberOfLeaves; j++)
  {
   pathLeafToa = Paths[i];
   pathaToLeaf = reverser( Paths[j] );
   //Raw paths
   pathLeafiToLeafj = pathLeafToa + pathaToLeaf;
   //An editing grammar to get true paths
   pathLeafiToLeafj = grammar( pathLeafiToLeafj );
   if (i == j) M[i][j] = 0;
   else
    {
 String leaf = "" + pathLeafiToLeafj.charAt(0);
 int p = leaves.indexOf(leaf);
 distance = lengthsLeaves[p];
 int n = pathLeafiToLeafj.length();
 for(int k = 1; k < n-1; k++)
 {
  innerNode =  "" + pathLeafiToLeafj.charAt(k);
  p = internalNodes.indexOf(innerNode);
  distance = distance + lengthsInnerLinks[p];
 }
 leaf =  "" + pathLeafiToLeafj.charAt(n-1);
 p = leaves.indexOf(leaf);
 distance = distance + lengthsLeaves[p];
  M[i][j] = distance;
    }
  }
 return  M;
}
```

```
//Translation from Newick’s to quasi-binary notation:
//addresses are attached to tree in Newick’s notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
if (print2)
System.out.println("Input tree at toQuasiBinaryInner " +
                        "= \n" + tree);
//Output
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + ’+’;
```

```
 if (print)
System.out.println("Address = " + address);

}


//If this char is a leaf
if (leaves.contains(thisChar))
 {
  // System.out.println("leaf");
  treeToBi =  treeToBi + content + address + thisChar;
  content = "";
 }

if (numeric.contains(thisChar))
treeToBi =  treeToBi + thisChar;

//If this char is an inner node
if (internalNodes.contains(thisChar))
 {
  int m = address.length();
//delete last bit from the right
  if (m>0) address = address.substring(0, m-1) ;
  if (address.contentEquals("")) address = "+";
  treeToBi =  treeToBi + address + thisChar;

 }


//This char = , and goes after an inner node
if  (thisChar.contentEquals(","))
 {
  //System.out.println
//("comma : "  + previousChar + thisChar );
  treeToBi =  treeToBi + thisChar ;
  if (print)
System.out.println("Address at ,1   = " + address);
```

```
   if    (numeric.contains(previousChar))
  address = address + '-';//Add a - to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree  (in links with lengths notation)
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
```

```
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method


//Prints  a matrix and its title
private static void printMatrix(double Matrix[][],
```

```
                        String title)
{
System.out.println( "\n" + title );
int m = Matrix.length;
System.out.print(  "\t");
for ( int j = 0; j < m; j++ )
  System.out.print( leaves.charAt(j)+ "\t");
System.out.println();
for ( int i = 0; i < m; i++ )
 {
  System.out.print(leaves.charAt(i ) + "\t");
  for ( int j = 0; j < m; j++ )
 System.out.print( Matrix[ i ][j]+ "\t");
  System.out.println();
 }
}


//Tree is translated
//from link notation to binary.
//Output in Record[]: even entries= addresses, odd = names.
//Associated matrix is also calculated
public static double [][] associatedMatrix(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
//String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
 System.out.println("Input tree  at translate = "    + tree);
  System.out.println("Selected tree in " +
```

```
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

 //A vector with addresses and other with names are composed
 makeInputInner(bTree);
 //Paths from leaves to node a are found.
 //Reverse paths are also listed.
 pathFinder(bTree);
 //The distance matrix among leaves is calculated
 double M[][] = matrixBuilder();
 return M;
}


//=========================================================
//================== Evolution of fitting =================
//Lengths are added to given tree topology and evolved
//to find the best fitting of  topology
//to target distance matrix.
//Returned value at evolveLengths(String topology) =
//the distance between
//target matrix and the matrix associated to
//the best fitting of this topology.
//=========================================================



//Population of lengths is initialized at random.
//Lengths take values on digital numbers 0,...,9
private void initialize()
{
 for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = r.nextInt(maxLength);
```

```
}

//Recently generated new population becomes the old one
private static void newToOld()
{
for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = newLengths[i][j][k];

}

//Old lengths are copied to a service vector
private static void oldToService(int i)
{

for(int k = 0; k < numberOfLeaves; k++)
 {lengthsLeaves[k] =  (int) oldLengths[i][0][k];}

for(int k = 0; k < numberOfLeaves; k++)
{lengthsInnerLinks[k] =  (int) oldLengths[i][1][k];}
}


//The absolute distance among two matrices is calculated.
//Only triangular part of matrices are compared.
private static double absDistance(int n,
               double[][] M1, double[][] M2)
{
double abs = 0;
for(int i = 0; i < n; i++)
for(int j = i+1; j < n; j++)
 abs = abs + Math.abs(M1[i][j]-M2[i][j]);
return abs;
}


//Input : tree in link notation.
```

```
//Output:
//distance matrix associated to the leaves of the tree.
private static double[][] distanceMatrix(String tree)
{
String s = toQuasiBinaryInner(tree) ;
if (print4)
 System.out.println("\n Tree = " + s);
//Paths from leaves to node a are found.
//Reverse paths are also listed.
pathFinder(s);
//The distance matrix among leaves is calculated
double M[][] = matrixBuilder();
return M;
}



//A tree topology is received as input.
//Lengths from oldLengths are inserted into the topology
//to get a tree with lengths.
//The distance between target matrix and
//the associated distance matrix to tree is calculated.
//The minimal distance is found:
//this renders the fittest tree.
private double findDistances(String topology)
{
 for(int i = 0; i < N; i++)
 {
 oldToService(i);

  String tree = insertLengths(topology);
  if (print4)
 System.out.println("\n Tree = " + tree);
  double M[][] = distanceMatrix(tree);
  if (print4)
{
 printMatrix(M,
```

```
 "Distance Matrix associated to   leaves"
 + " of  tree");
}
  double d= absDistance(numberOfLeaves,targetMatrix, M);
  if (print3)
  System.out.println("\n Individual " + i + ":\n" + tree);
  //Distance from the associated distance matrix of the tree
  //to target matrix is evaluated
  distance[i] = d;
  if (print4)
  System.out.println(i + " Dist to tarrrrrrget = "
                                  + distance[i]);


 }//End of for(int i...

 //The fittest individual minimizes distance function
 double min = distance[0];
 best = 0;
 for(int i = 1; i < N; i++)
 {
  if (distance[i] < min )
  {
   min = distance[i];
       best = i;
  }
 }
 //lengths are copied to  service vectors
 oldToService(best);
 //Best tree for the given topology
 bestTree = insertLengths(topology);



 if (print4)
 {
  System.out.println("\n Distance + BestTreeeeeee  = " +
  + min + " \n"  + bestTree
```

```
            + "  "  + "\n");
 }
 return min;
}

//Selection is implemented
private void makeFitnesses()
{
 double maxD = distance[0];
 double minD = distance[0];
 double b = 1 ;
 //Min and max distances are found
 for(int i = 1; i < N; i++)
  {
if ( distance[i] > maxD ) maxD =  distance[i];
if ( distance[i] < minD ) minD =  distance[i];
  }

 if (minD == maxD) //All   equally fit
   {
  for(int i = 0; i < N; i++)
fitness[i] = 1;
   }
 else
  {
 //Distance to target determine fitness:
//fitness[maxD] = 0;    fitness[minD] = 1;
 for(int i = 0; i < N; i++)
 {
  b = 1;
  b = b/(minD - maxD);
  double a = -b*maxD;
  fitness[i] = a + b * distance[i];
  if (print3)
   {
  System.out.println("Individual = "
            + i + " has distance " +
         distance[i] + " fitness " + fitness[i]);
```

```
   }
}
  }
}
//Distances are translated to
//probabilities of not reproducing.
//N individuals are sampled at random from the population
//according to those probabilities
private void reproduction()
{
 makeFitnesses();
 //The best is cloned
 for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[0][0][j] = oldLengths[best][0][j];
 newLengths[0][1][j] = oldLengths[best][1][j];
}

 //N-1 children are sampled
 for(int i = 1; i < N; i++)
  {
 //flag
 boolean go = true;
   while (go)
    {
 //An individual is chosen at random
 int k = r.nextInt(N);
 //k is reproduced according to its fitness
 double q = r.nextDouble();
 if (print4)
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] );
 if ( q < fitness[k]) //make a copy of k
  {
   for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[i][0][j] = oldLengths[k][0][j];
```

```
 newLengths[i][1][j] = oldLengths[k][1][j];
}
 go = false;//Pass to making a new individual
   if (print4)
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] + "->" + i);
  }

    }

  }//End of for //N-1 children are sampled
}



//Individuals are mutated
private void mutation()
{
 //The old best, i = 0, is conserved intact
 for(int i = 1; i < N; i++)
   {
//Mutation in links that connect leaves
for(int k = 0; k < numberOfLeaves; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][0][k] = r.nextInt(maxLength);

 }
//Mutation in links that connect inner nodes
for(int k = 0; k < numberOfLeaves-2; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][1][k] = r.nextInt(maxLength);

 }
```

```
   }
}


//Individual l is recombined to another
//that is chosen at random
private static void recInd(int l)
{
 //leaf else inner node?
 int a = r.nextInt(2);
 //Partner at random
 int k = r.nextInt(N);

 if (a == 0)
  {
   //Random site for recombination
   int site = r.nextInt(numberOfLeaves);
   //Take a part from l
   for(int j = 0; j < site; j++)
  newLengths[l][0][j] = oldLengths[l][0][j];
   //Another part from k
   for(int j = site; j < numberOfLeaves; j++)
  newLengths[l][0][j] = oldLengths[k][0][j];
  }
 else
 {
  int site = r.nextInt(numberOfLeaves-2);
  for(int j = 0; j < site; j++)
  newLengths[l][1][j] = oldLengths[l][1][j];
  for(int j = site; j < numberOfLeaves; j++)
 newLengths[l][1][j] = oldLengths[k][1][j];
}
}



//Recombination master
private static void recombination()
```

```
{
if (print) System.out.println("\nRECOMBINATION\n");
newToOld();
//Individual l>0 is a recombinant version
for(int l=1; l< N; l++)
{
  recInd( l );
}
}



//Lengths are evolved to find the best fitting
//of the given topology to target matrix
private double evolveLengths(String topology)
{
 if(print4)
 System.out.println("\n========Evolution of lengths " +
  "for topology = "   + topology );
 initialize();
 double min = -1;
 for(int gen = 0; gen < numberOfGenerations; gen++)
 {
  if (print4)
System.out.println("\n=== Gen = " + gen);
  min =  findDistances(topology);
  reproduction();
  mutation();
  recombination();
  newToOld();
 }
 return min;
}
```

```
//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree12 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree12()
 {
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode(seed);
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);

    JScrollPane scrollPane = new JScrollPane(tree);
```

```
    add(scrollPane);
  }


  /** Selected tree is unveiled */
  public String unveilTree()
  {
    DefaultMutableTreeNode selectedNode = null;
    TreePath selectedPath = tree.getSelectionPath();

    if (selectedPath == null) {
     selectedNode = rootNode;
    } else {
     selectedNode = (DefaultMutableTreeNode)
             (selectedPath.getLastPathComponent());
    }

    String tree =  selectedNode.toString();
    if (print)
    System.out.println("\nSelected tree at unveilTree() = "
                         + tree);
    return tree;
  }


  //Add a node
  public DefaultMutableTreeNode
              addObject(DefaultMutableTreeNode parent,
      Object child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
               new DefaultMutableTreeNode(child);

    if (parent == null) {
      parent = rootNode;
    }

    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
```

```
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                     (new TreePath(childNode.getPath()));
    }
    return childNode;
  }



 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   if (print)
 System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();
   indexOfTopologies = 0;

while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
if (print)
System.out.println("ParentTree = " +  parentTree );

//=================
```

```
//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
parentTree.replace(link, replacement);
```

```
   }
 if ((4+counterGen)== numberOfLeaves)
 {

  //Lengths are added and evolved to find the
  //best fitting of this topology
  //to target distance matrix.
  //Returned value: the distance between
  //target matrix and the matrix associated to
  //the best fitting of this topology.

  double distance = evolveLengths(newTree);
  bestFitting[indexOfTopologies] = distance;

  System.out.println("\n====================" +
   "\nIndex + Topology + " +
   "distance to target + best tree: \n"
  + indexOfTopologies
   +  ":  "  +
  newTree + " -> "   + simplify(newTree)
 + " " +  distance  + "\n" + bestTree + "\n");

  bestTreeOfTopology[indexOfTopologies]
                      = bestTree;

 if (print5)
 {
//Distance from M(bestTree)
 //to target matrix is evaluated
 double M[][] = distanceMatrix(bestTree);
  if (print5)
{
 printMatrix(M,
 "Distance Matrix associated to leaves"
 + " of  tree");
}
  double d= absDistance(numberOfLeaves,
                  targetMatrix,
```

```
M);

System.out.println("Dist to target = " + d);
 }
  indexOfTopologies++;
 }

String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
       * specified node are the same.
       */
```

```
      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)
                 (node.getChildAt(index));

      System.out.println(
        "The user has finished editing the node.");
      System.out.println("New value: " +
                  node.getUserObject());
    }

    public void treeNodesInserted(TreeModelEvent e)
    {
    }

    public void treeNodesRemoved(TreeModelEvent e)
    {
    }

    public void treeStructureChanged(TreeModelEvent e)
    {
    }
  }
}//End of DynamicTree12



//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).

//A generator of random trees in Newick's notation
```

```
//is included.

//Resultant trees are drawn.




class DrawBiTree9 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane29;
JScrollPane scroller;




//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;



  String  NwTree = "";
```

```
 boolean print = false;


//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.



//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];



//Set up the instructions.
JLabel instruct1;
JLabel instruct2;

public DrawBiTree9()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));


//Set up the instructions.
instruct1 = new JLabel(title1);
instruct2 = new JLabel(title2);

instructionPanel.setFocusable(true);
instructionPanel.add(instruct1);
instructionPanel.add(instruct2);
```

```
//Set up the drawing area.
drawingPane29 = new drawingPane29();
drawingPane29.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane29);
scroller.setPreferredSize(new Dimension(500,900));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
 class drawingPane29 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
int V;
//Addresses are interpreted as paths
```

```
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);


//area = size of the graphic (width, height)
Dimension area =
```

```
 new Dimension(horizontalSize,12*verticalSize);
//area is set as the preferred size of the graphic
drawingPane29.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane29.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 600;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                                 + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
Dimension center = new Dimension(x11,y11);
V = scale * verticalLength;
//Path is interpreted
```

```
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
int counter = 0;
//Seed is not  drawn: we lack one
```

```
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                       verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
private void drawBasic(Graphics2D g2d)
{
```

```
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane29


/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
JFrame frame =
   new JFrame(titleFrame);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree9();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree9

}//End of main class J168 EvoTree2
```

171, page 420. A distance matrix is accepted as input:

```
//Program J171 EvoTree3
/*
 * We generate a random tree topology
 * and attach random integer lengths to links.
 * In that way we get the target tree, T.
 * Next, we find the associated distance matrix D  to T.
 * Our problem is:
 *
 * Find tree t such that the absolute distance between
 * matrix D and the associated matrix to t is a minimum.
 * Absolute distance between M and N = sum |Mij-Nij|.
 *
 * We run the next test:
 * If we begin with a tree T, compute D, and then t,
 * we expect to return with the initial tree: t = T.
 *
 * To associate a distance matrix to a tree,
 * we calculate the distance in within
 * each pair of nodes (leaves + inner nodes).
```

```
 * To that aim, we begin with the first node and
 * walk over the tree until the second node
 * and add the lengths of all traversed links.
 * The resulting number is the  distance.
 *
 * To find the tree that best fit D,
 * we  scan all possible topologies,
 * each one of which is submitted to evolution
 * to finely tune lengths.
 *
 * Formats = Link and Newick’s notations. And variants.
 *
 * The performance of the GUI has been improved.
 * An automatic halting option has been added to be applied
 * if perfect matching is found.
 * The program accepts for input
 * a distance matrix with integer values.
 * A test for being a distance matrix is run.
 */



import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Enumeration;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.event.TreeModelEvent;
```

```
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;

import java.awt.*;



import javax.swing.JComponent;


public class EvoTree3 extends JPanel
                           implements ActionListener
{

 private static final long serialVersionUID = 1L;

 //The program accepts a distance matrix as input
 //else
 //it generates a random tree, calculates the distance
 //matrix associated to leaves and posits it
 //as target matrix.
 private static boolean inputMatrix = true;
 private static boolean randomTree = (!(inputMatrix));
//Number of leaves that must go in all trees > 3
private static int numberOfLeaves = 6;

//Input distance matrix:
//Example 5.4, pag 99, Isaev (2001)
//Introduction to Mathematical Methods in Bioinformatics
private static double targetMatrix[][]
 = {{0,8,3,14,10,12},
   {8,0,9,10,6,8},
   {3,9,0,15,11,13},
   {14,10,15,0,10,8},
   {10,6,11,10,0,8},
```

```
  {12,8,13,8,8,0}};


//Everything is built by recursion over this tree
private static   String   seed =
   "((Aa,Ba),Ca)";

private static  String leaves =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
private static String internalNodes =
"abcdefghijklmnopqrstuvwxyz";

//Next leaf to be grafted
private static  int nextLeafIndex = 2;
//Next internal node to be inserted
private static  int nextIntNodeIndex = 0;


static int maxNumberOfLeaves = leaves.length();


private static String[] Record =  new String[10000];

private static  int indexOfTopologies;
//The distance to target of the best fitting tree
//of the given topology
private static double[] bestFitting =  new double[10000];
private static String[] bestTreeOfTopology
                           = new String[10000];

//Last filled position  in Record[]
private static int lastUsedPosition;
private static String selectedTree = "";


 //DynamicTree is an inner service class that comes
 //at the end of this file.
 private static DynamicTree13 treePanel;
```

```
  private static String VISUALIZE_COMMAND = "Visualize";

   //Length of the link that glues a leaf to the tree
  private static int lengthsLeaves[]  =
                new int[numberOfLeaves];
  //Length of the link that glues
  //an inner node to another one
  private static int lengthsInnerLinks[]  =
              new int[numberOfLeaves];


  private static Random r = new Random();
  //The control parameter of the random generator for int
  private static int maxLength = 10;

  //Paths from each leaf to node a
  private static String[] Paths =
              new String[numberOfLeaves];

  private static String targetTree;



  //=======Evolution parameters============



//N individuals at all.
static private int N = 10;
static private double oldLengths[][][] =
 new double[N][2][maxNumberOfLeaves];

static private double newLengths[][][] =
new double[N][2][maxNumberOfLeaves];

static private double distance[] = new double[N];
static private double fitness[] = new double[N];
```

```
static private int numberOfGenerations = 10;

//Mutation rate per site (= entry) per generation
static private double    m = 0.5;

static private int best;
//The best tree for a given topology
static private String bestTree = "";

//=========Debugging help ===============

private static boolean print  = false;
private static boolean print2 = false;
private static boolean print3 = false;
private static boolean print4 = false;
//Turn on print5 firstly
private static boolean print5 = true;

//==============

private static String titleFrame = "";
private static String title1 = "";
private static String title2 = "";
private static EvoTree3 newContentPane = new EvoTree3();



  //=======Constructor of main class: GUI basics=====

  public EvoTree3()
  {
   super(new BorderLayout());
   JPanel instructionPanel = new JPanel(new GridLayout(0,1));
   JLabel instructions1 = new JLabel(
      "Select  a tree and click the Visualize button.");


   instructionPanel.setFocusable(true);
```

```
   instructionPanel.add(instructions1);

   add(instructionPanel, BorderLayout.PAGE_END);

   //A new node is added
   JButton visualizeButton = new JButton("Visualize");
   visualizeButton.setActionCommand(VISUALIZE_COMMAND);
   visualizeButton.addActionListener(this);

   // Create the components.
   treePanel = new DynamicTree13();
   // Lay everything out.
   treePanel.setPreferredSize(new Dimension(700, 800));
   instructionPanel.add(visualizeButton);
   add(treePanel, BorderLayout.CENTER);
  }//End of constructor

  //Response to selecting visualization
  public void actionPerformed(ActionEvent e)
   {
String command = e.getActionCommand();
    if (VISUALIZE_COMMAND.equals(command))
 {
      //Selected tree must be unveiled
      selectedTree = treePanel.unveilTree();
      System.out.println("\nSelected tree in Link notation"
       + " after unveilTree() = \n"   + selectedTree);
      /*Random lengths are added to topology
      makeLengths();
      //Lengths are added
      selectedTree = insertLengths(selectedTree);
      */
      int i =  findTheBest(selectedTree);
      selectedTree =  bestTreeOfTopology[i];
      double d = bestFitting[i];
      System.out.println( " Best fitting = \n " +
                                  selectedTree);
      System.out.println( " Best distance = \n "
```

```
       + d);


       decoderLengths(selectedTree);
  //Translation from one format into another
       //output = a vector with binary addresses and contents
       double[][] M =  associatedMatrix(selectedTree);
       printMatrix(M, "distance matrix associated to the " +
       "best tree for this topology");
       //Selected tree is drawn
       title1 = "Best tree for the topology = "
        + simplify(selectedTree);
       title2 = "Distance to target = " + d;
       DrawBiTree10 a = newContentPane.new DrawBiTree10();
       a.createAndShowGUI();

 }
  }

//======================

  //Test for being a distance matrix.
  //MaxLenght is calulated.
 private static boolean
        isMatrixDistanceMatrix(double[][] M)
{
 int l = M.length;
 boolean isD = true;
 //Is M(u,u) = 0?
 for(int u = 0; u < l; u++)

if (M[u][u] ==  0) ;
 else
  {
isD = false;
return isD;
  }
```

```
 //Is M(u,v) = M(v,u)?
 for(int u = 0; u < l; u++)
 for(int v = u+1; v < l; v++)
if (M[u][v] == M[v][u] ) ;
 else
{
 isD = false;
 return isD;
}

 //Is M(u,v)>0 for u not v?
 for(int u = 0; u < l; u++)
 for(int v = u+1; v < l; v++)
if (M[u][v] >  0) ;
 else
{
 isD = false;
 return isD;
}



 //Is M(u,v) < M(u,w) + M(w,v)?
 for(int u = 0; u < l; u++)
  for(int v = 0; v < l; v++)
   for(int w = 0; w < l; w++)

if (M[u][v] <= M[u][w]+M[w][v]  ) ;
 else
{
 isD = false;
 return isD;
}
 double m = 0;
 for(int u = 0; u < l; u++)
 for(int v = u+1; v < l; v++)
 if (M[u][v] > m ) m = M[u][v];
 maxLength = (int) m;
```

```
 return true;
}


public static void main(String[] args)
  {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application’s GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
     public void run()
     {
      //If input is a distance matrix:
      if (inputMatrix)
      {
       printMatrix(targetMatrix, "Input Matrix");
       boolean isD = isMatrixDistanceMatrix(targetMatrix);
         if (isD) System.out.println("Matrix " +
        "is a distance matrix");
         else
          {
        System.out.println("Matrix " +
             "is NOT a distance matrix");
           }
      }
    else//if a random tree is generated -> Matrix
     {
     String tree = randomTopology();
     makeLengths();
     targetTree = insertLengths(tree);
     System.out.println("Target tree: each link " +
     "has its own length = \n" + targetTree);
     targetMatrix  =
         new double[numberOfLeaves][numberOfLeaves];
     targetMatrix = associatedMatrix(targetTree);
     }
     createAndShowGUI();
```

```
    System.out.println("\nDistance to target = " +
         " Absolute distance between" +
        " target matrix  and \n" +
        "the distance matrix associated to " +
        "the tree with lengths of links that \n" +
        "is the best fitting  of the topology.\n" +
        "Perfect fitting: distance = 0 \n");
     System.out.println("\nNumber of Generations = "
          + numberOfGenerations);
     //To find the best fitting, all tree topologies are
     //scanned and each one of them is evolved for
     //the best particular fitting.
     //All this is done along the process of
     //generating all topologies
     generateTopologies();
     //Draw Target tree: this malfunctions.
     /*
      titleFrame = "Target tree = " + tree;
       selectedTree = targetTree;
       DrawBiTree10 b = newContentPane.new DrawBiTree10();
       b.createAndShowGUI();
      */
     }
   });
 }


//From parentTree, a tree in Newick's notation,
//a Child  tree is generated by a generative grammar.
//A random election is made among all possibilities.
private static String generateChild(String parentTree)
{

 nextLeafIndex++;
 char nextLeaf= leaves.charAt(nextLeafIndex);
 //Next internal node to be inserted
 nextIntNodeIndex++;
 char nextIntNode = internalNodes.charAt(nextIntNodeIndex);
```

```
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
//counter of links = 0;
int counter = 0;
//Links are counted
for(int i = 0; i < l; i++ )
 {
  //parentTree is scanned char by char
  char I = parentTree.charAt(i);
//Is I an internal node that denotes a link?
if (charIsInternalNode(I)) counter++;
 }
//A link is chosen
int k = r.nextInt(counter);
if (print2)
System.out.println("Random number = " + k);
//new branch is grafted at k
counter = 0;
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node that denotes a replacement?
if (charIsInternalNode(I))
 {
  if (counter == k) //Is I the chosen link?
   {
//Check for the char before I
char a = parentTree.charAt(i-1);
//If I links to a complex structure
```

```
 if (a == ')') newTree = useRule2(parentTree, i,
                nextLeaf, nextIntNode);
 else //if I links to a leaf
  {
   //Link to be replaced
   link = "" + a + I;
   //Replacement rule 1:
   //aI -> (aN,FN)I;   N= nextIntNode;
   //F = nextLeaf
  replacement = "(" + a + nextIntNode + ","
           + nextLeaf + nextIntNode + ")" + I;
  //Rule 1 is executed
  newTree = parentTree.replace(link, replacement);
  }
 if (print2)
 System.out.println(newTree + " -> "
                       + simplify(newTree));

  i = l;
    }
   counter++;
 }
  }
 return newTree;
}


//A tree topology is generated at random.
public static String randomTopology()
{
 if (print2)
 System.out.println("Parent tree = \n" + seed
  + " -> "    + simplify(seed));
 String tree = seed;
 int limit = numberOfLeaves-3; //less than 11.
 for(int i = 0; i < limit; i++ )
 tree = generateChild(tree);
 System.out.println("Target topology = \n" + tree);
```

```
  return tree;
 }


//======================


//Random lengths are added to a tree in Link notation
public static void makeLengths()
 {
  //Lengths with digit values are generated
  for(int i = 0; i < numberOfLeaves; i++)
   {
 //Zero lengths are not allowed
 lengthsLeaves[i] = 1 + r.nextInt(maxLength - 1);
 lengthsInnerLinks[i] = 1 +  r.nextInt(maxLength - 1);
   }
 }


//The best fitting tree for chosen topology is found
public static int findTheBest(String tree)
{
 String selectedTopology = simplify(tree);
 String thisTopology = "";
 String thisTree = "";
 int bestPosition = -1;

for(int i = 0; i < indexOfTopologies; i++)
 {
  thisTree = bestTreeOfTopology[i];
  thisTopology = simplify(thisTree);
  if (selectedTopology.equals(thisTopology))
  {
   bestPosition = i;
   return bestPosition;
  }
 }
```

```
return  bestPosition;
}


  /*
  //Lengths with real values are generated
  double lengthsLeaves[]  = new double[l];
  //Last two entries are redundant
  double lengthsInnerLinks[]  = new double[l];
  for(int i = 0; i < numberOfLeaves; i++)
   {
 lengthsLeaves[i] = r.nextDouble();
 lengthsInnerLinks[i] =  r.nextDouble();
   }
  */

//Random lengths are added to a tree in Link notation
public static String insertLengths(String treeLinkN)
 {
  String thisChar = "";
  String newTree = "";
  //length of String   tree
  int m = treeLinkN.length();
  //Tree is scanned char by char
  for(int i = 0; i < m; i++ )
   {
    thisChar = "" + treeLinkN.charAt(i);
    newTree = newTree + thisChar;
    if (print)
    System.out.println(newTree.length() + " " + newTree);
    //Lengths are appended
    if (leaves.contains(thisChar))
     {
      int k = leaves.indexOf(thisChar);
      double j = lengthsLeaves[k];
     newTree = newTree + ":" + j;
     }
    if (internalNodes.contains(thisChar))
```

```
    {
     int k = internalNodes.indexOf(thisChar);
     double j = lengthsInnerLinks[k];
     newTree = newTree + ":" + j;
    }
   }
   return newTree;
 }


  //Data about the tree is printed to the console
  public void report()
  {
System.out.println("The root has " +
      treePanel.rootNode.getChildCount() + " children");
System.out.println("The tree's depth is " +
      treePanel.rootNode.getDepth());
System.out.println("The tree has " +
      treePanel.rootNode.getLeafCount() + " leaves");
 System.out.println("'Root' is really a root? " +
      treePanel.rootNode.isRoot());
 System.out.println("Root's userObject: " +
      treePanel.rootNode.toString());
  }


  /**
   * Create the GUI and show it. For thread safety,
   * this method should be
   * invoked from the event-dispatching thread.
   */
private static void createAndShowGUI()
 {
  // Create and set up the window.
  JFrame frame =
new JFrame("Tree of tree topologies.");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  // Create and set up the content pane.
```

```
  // content panes must be opaque
  newContentPane.setOpaque(true);
  frame.setContentPane(newContentPane);

  // Display the window.
  frame.pack();
  frame.setVisible(true);
 }


//Detect internal nodes, different than leaves
private static boolean charIsInternalNode(char c)
{
 String s = "" + c;
 if (internalNodes.contains(s)) return true;
 else return false;
}

//Simplification from Link notation to Newick's notation:
// (((AJ,DJ)I,BI),CI) -> (((A,D),B),C)
private static String simplify(String newTree)
{
String tree = "";
String numeric = ".:0123456789";
//length of string newTree
int l = newTree.length();
for(int i = 0; i < l; i++ )
 {
  //newTree is scanned char by char
 char c = newTree.charAt(i);
 String s = "" + c;
 //Is c an internal node to be deleted?
 if (charIsInternalNode(c));
  else //Is c numeric: delete it!
 if (numeric.contains(s)); //add char c to output
 else tree = tree + c;
 }
```

```
return tree;
}

//Rule 2 is used to generate a new tree:
//(S)I -> ((S)N,FN)I.
//S = subtree, I = internal node, F = new leaf,
//N = new internal node.
//Rule 2 grafts new branch FN to link (S)I
//i is the position of char I in string parentTree
private static String useRule2(String parentTree, int i,
       char F, char N)
{
String newTree = "";
char I = parentTree.charAt(i);
if (print)
System.out.println("i = " + i + " I = " + I);
//Subtree S of link (S)I must be read
String S = "";
//Counter of left parenthesis
int counter = 0;
String pl = "" + '(';
String pr = "" + ')';
//Position to be scanned
int position = i;
boolean go = true;
//All right parentheses to the near left of I are captured
while (go)
 {
  position = position -1;
  String a = "" + parentTree.charAt(position);

  if (print)
  System.out.println("a = " + a + " p = " + pr);
  if (a.equals(pr) )
  {
counter++;
S = a + S;
  }
```

```
  else go = false;
 }//End of while
if (print)
System.out.println("counter = " + counter);
//The complete subtree is captured:
//Parentheses must balanced
while (counter > 0)
 {
  String a = "" + parentTree.charAt(position);
  if (a.equals(pr))   counter++;
  if (a.equals(pl))  counter--;
  S = a + S;
  if (print)
  System.out.println("Subtree = " + S);
  position = position -1;
  }
//Replacement rule 2:
//(S)I -> ((S)N,FN)I.  N= nextIntNode;
//F = nextLeaf
//Target to be replaced
String target  =   S  + I;
String replacement =  "("+ S +   N + ","
          + F + N+ ")" + I;
//Rule 1 is executed
newTree = parentTree.replace(target, replacement);
//If I links to a complex structure
return newTree;
}

//Input: string tree with complete information.
//Work: lengthsLeaves[] and lengthsInnerLinks[]
//are read from tree.
private static void decoderLengths(String tree)
{
 if(print2)
 System.out.println(
"Input tree at recoverLengths = \n "   + tree  );
 //Delete :
```

```
 tree = tree.replaceAll(":", "");
 if(print2)
 System.out.println(
"Input tree without : = \n "   + tree  );
 int n = tree.length();
 String thisChar = "";
 String nextChar = "";
 String numbers = ".0123456789";
 boolean leaf = false;
 boolean node = false;
 String Leaf = "";
 String numberLeaf = "";
 String numberNode = "";
 String Node = "";
 int j ;
 double f;

//All chars in tree are scanned
 for(int i=0; i < n; i++)
  {
   thisChar = "" + tree.charAt(i);
   if (i < n-1) nextChar  = "" + tree.charAt(i+1);
   //Do we have a leaf?
   if (leaves.contains(thisChar))
    {
 leaf = true;
 Leaf = "" + thisChar;
    }

   //Do we have an internal node?
   if (internalNodes.contains(thisChar))
   {
 node = true;
 Node = "" + thisChar;
   }

   //Do we have a number that belongs in a leaf?
   if ( (leaf) & (numbers.contains(thisChar)))
```

```
{
 numberLeaf = numberLeaf + thisChar;
}

  //Do we have a number that belongs in a leaf?
   if ( (node) & (numbers.contains(thisChar)))
{
 numberNode = numberNode + thisChar;
}

   //Must we end accumulation of numbers of a leaf?
   if ( (leaf) & (internalNodes.contains(nextChar)))
{
  leaf = false;
  j = leaves.indexOf(Leaf);
  if (print4)
  System.out.println("Leaf = " + Leaf);
  f = new Double(numberLeaf);
  lengthsLeaves[j] = (int) Math.floor(f);
  numberLeaf = "";
}

   //Must we end accumulation of numbers of a node?
   if ( (node) & (!(numbers.contains(nextChar))))
{
  node = false;
  j = internalNodes.indexOf(Node);
  if (print4)
  System.out.println("Node = " + Node);
  f = new Double(numberNode);
  lengthsInnerLinks[j] = (int) Math.floor(f);
  numberNode = "";
}

  }
}
```

```
//Topologies are generated and
//the best fitting is filtered
private static void generateTopologies()
 {
  //New leaves are incorporated one by one
  for(int i = 0; i < numberOfLeaves - 3; i++)
   {
    if (print) System.out.println("Leaf = " + i);
treePanel.reproductionOfLeaves();
   }
  //The best topology is found
  double minD = bestFitting[0];
  int place = 0;
  for(int i = 1; i <   indexOfTopologies; i++)
 {
   if ( bestFitting[i] < minD )
{
 minD =  bestFitting[i];
     place = i;
}
 }

  System.out.println("================\n================\n");

  //Was input = null and a random tree was generated?
  if (randomTree)
  {
  System.out.println("TARGET TOPOLOGY = \n" +
  simplify(targetTree));
  System.out.println("TARGET TREE = \n" + targetTree);
  }
  printMatrix(targetMatrix, "Target Distance Matrix  ");
  String theBestTree = bestTreeOfTopology[place];
  System.out.println("\nBEST FITTING TOPOLOGY = \n" +
  simplify(theBestTree));
  System.out.println(
    "\nBEST DISTANCE AND FITTING TREE TO TARGET MATRIX  "
    + minD + "\n"    + theBestTree);
```

```
  //Distance from M(bestTree)
  //to target matrix is evaluated



  decoderLengths(theBestTree);
  double M[][] = associatedMatrix(theBestTree);
  double d= absDistance(numberOfLeaves,
             targetMatrix, M);
  printMatrix(M, "Distance Matrix associated to " +
                    "the leaves of the best tree");
  System.out.println("Distance to target matrix = "   + d);
  //Draw theBestTree
  titleFrame = "Best fitting tree";
  title1 =  simplify(theBestTree);
  selectedTree = theBestTree;
  DrawBiTree10 c = newContentPane.new DrawBiTree10();
  c.createAndShowGUI();
  c.setLocation(500,  100);
 }



//========= Translation in within formats ==========



//bTree in (modified) Newick's notation
//is used to generate Record[]
//with addresses and contents.
public static void makeInput(String bTree)
{
 if (print)
 System.out.println("bTree at makeInput ////// " + bTree);
 int counter = 0;
 //length of String   tree
```

```
 int l = bTree.length();
 String thisChar = "";
 String address = "";
 String binNumbers = "01";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if (leaves.contains(thisChar))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;

counter++;
address = "";
   }
  }

 if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
        + lastUsedPosition);
}



//Outputs a word in reversed order.
private static String reverser(String word)
{
 return new StringBuilder(word).reverse().toString();
```

```
}


private static String cleanTree(String tree)
{
 if (print2)
 System.out.println("Input tree at deleteNumerics = \n "
        + tree);
 String treeClean = "";
 String numeric = ".:0123456789";
 String plusMinus = "+-";
 String thisChar = "";
  boolean leaf = false;
 //Numerics are washed
 //length of String   tree
 int l = tree.length();
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
  {
   thisChar = "" + tree.charAt(i);
   if (numeric.contains(thisChar));
    else treeClean = treeClean + thisChar;
  }
 if (print2)
 System.out.println("Tree without numerics = \n"
+ treeClean);
 //+/- are deleted from inner nodes that accompany leaves
 l = treeClean.length();
 String treeOut = "";
 //Tree is scanned char by char
 for(int i = 0; i < l; i++ )
 {
  thisChar = "" + treeClean.charAt(i);
  if (leaves.contains(thisChar))
  leaf  = true;
  if ((leaf) &  (plusMinus.contains(thisChar)));
else treeOut = treeOut + thisChar;
  if (internalNodes.contains(thisChar))
```

```
  leaf  = false;
}

 if (print2)
 System.out.println("Tree at output of cleanTree = \n"
+ treeOut);
return treeOut;

}



//The path from each leaf to node a is found.
//The input contains branch’s indicator, say:
//((++Aa,+-Ba),(-+Cb,((--++Dd,--+-Fd)--c,---Ec)-b)+a)
//Output for leaf A: Aa+.
//Output for leaf C: Cba-
//Output for leaf D: Ddcba-
private static void findPaths(String tree)
{
 if (print2)
 System.out.println("Input tree at findPaths = \n" + tree);
 //length of String   tree
 int l = tree.length();
 String plusMinus = "+-";
 String thisChar;
 String nextChar;
 String path = "";
 //Branch indicative with respect to root, + else -
 String branch;
 String address ;
 //The number of links of a path to root
 int level = 0;
 String treeReversed = reverser(tree);
 if (print2)
 System.out.println("Reversed tree = " + treeReversed);
```

```
//Each leaf has a path to a
for(int k= 0; k < numberOfLeaves; k++)
{
 String leaf = "" + leaves.charAt(k);
 //Candidate nodes to be part of the path to node a
 //must have  sign +/- as prefix.
 //Or as suffix in the reversed tree.
 int position = treeReversed.indexOf(leaf);
 if (position > -1) // if leaf is in tree
 {
  if (print2)
  System.out.println("\nLeaf = " + leaf );
  address = "";
  boolean go = true;
  //branchIndicator and level are found
  for(int i = position+1 ; ((i < l) & (go)); i++ )
  {
   thisChar = "" + treeReversed.charAt(i);
   if   (plusMinus.contains(thisChar)  )
address = thisChar + address;
   else go = false;
  }
  level = address.length();
  if (print2)
  System.out.println("LeafAddressLevel  = " + leaf
                     + address + level);
  branch = "" + address.charAt(0);


  //Path from leaf to node a is found in (not reversed) tree
  position = tree.indexOf(leaf);
  thisChar = "" + tree.charAt(position);
  nextChar = "" + tree.charAt(position + 1);
  //Output
  path = "" + leaf + nextChar;
  go = true;
  if  (nextChar.contentEquals("a") ) //leaf is neighbor of a
   {
```

```
  path = path + branch;
  if (print)
  System.out.println("Path short = " + path);
   }
  else  //level > 1
  {

  //Tree is scanned char by char to get path
  String iChar = "";
  for(int i = position + 2; ((i < l) & (go)); i++ )
  {
   iChar = "" + tree.charAt(i);


   int newLevel = 0;
   if (internalNodes.contains(iChar))//InternalNodes
{
     if   (iChar.contentEquals("a")  )
      {
   path  = path +  iChar;
   go = false; //end
      }
     else //Internal node different than a
     {
      if (print)
      System.out.println("Internal node not a) = " + iChar );
      int rP = l -i-1; //index of iChar in treeReversed
      String  nextNewChar = "" + treeReversed.charAt(rP+1);

      if ( plusMinus.contains(nextNewChar )) //candidate node
       {
       if (print)
     System.out.println("next candidate = " + iChar);
        String addressNew = "";

        //level of candidate node iChar  is inquired
        String thatChar = "";
        boolean go2 = true;
```

```
        for(int j = rP+1 ; ((j < l) & (go2)); j++ )
         {
       thatChar = "" + treeReversed.charAt(j);
       if   (plusMinus.contains(thatChar)  )
          addressNew = thatChar + addressNew;
       else go2 = false;
         }
        if (print)
        System.out.println("AddressNew " +
                                    iChar + addressNew);
       newLevel = addressNew.length();
       //A path links a node with another in a lower level
       if  ( (newLevel < level) &
        (!(nextChar.contentEquals(iChar))))
        {
       path = path + iChar;
       level = newLevel;
       if (print)
       System.out.println("Path newLevel= " + path);
        }
      } //End of if candidate node
     }//End of else Internal node different than a
}//End of if InternalNodes
   }//End of Tree is scanned char by char

  path = path + branch;

 }//End of  else level > 1
 }//End of if leaf is in tree
 if (print2)
 System.out.println("Path completed  = " + path);
 Paths[k] = path;
}//End of each leaf
}



//Paths from leaves to node a are found
```

```
//Reverse paths are also calculated.
//Input tree in quasibinary notation
//with addresses and lengths
private static void pathFinder(String tree)
{
 //Useless information is deleted
 tree = cleanTree(tree);
 //Find the path to node a from each leaf.
 //Add an indicator of the branch
 //with respect to the root: up(+) else down(-)
  findPaths(tree);
}

//Input: a long raw string with path from leaf to node a
//plus path from node a to leaf.
//Editing is done to get true paths.
private static String grammar (String path)
{
 int n = path.length();
 String thisChar = "";
 String nextChar = "";
 String oldPath = path;
 String plusMinus = "a+-a";
 String minusPlus = "a-+a";
 String two = "";
 int N = path.length();
//Rule 1: two equal symbols produce nothing
 for(int j=1; j < N; j++)
  {
   for(int i=1; i < n-1;)
   {
    thisChar = "" + path.charAt(i);
    nextChar = "" + path.charAt(i+1);
    two = "" + thisChar + nextChar;
    if (thisChar.contentEquals(nextChar))
  {
   i = i + 2;
   path = path.replace(two, "");
```

```
  }
    else
  {
   i = i+1;
  }
  n = path.length();
   }
  }

 //Rule 2: a+-a -> a

 if  (path.contains(plusMinus))
      path = path.replace(plusMinus, "a");
 if   (path.contains(minusPlus))
     path = path.replace(minusPlus, "a");
 if (print2)
 System.out.println("input = " + oldPath +
                    " output = " + path);
 return path;
}

//The distance matrix among leaves is calculated
private static double[][] matrixBuilder()
{
double M[][] =
new double[numberOfLeaves][numberOfLeaves];
 //Path to node a from leaf
 String pathLeafToa = "";
 //Path to leaf from node a
 String pathaToLeaf = "";
 //Path from leaf to lead
 String pathLeafiToLeafj = "";
 String innerNode = "";
 double distance;
 for(int i = 0; i < numberOfLeaves; i++)
  for(int j = 0; j < numberOfLeaves; j++)
  {
   pathLeafToa = Paths[i];
```

```
   pathaToLeaf = reverser( Paths[j] );
   //Raw paths
   pathLeafiToLeafj = pathLeafToa + pathaToLeaf;
   //An editing grammar to get true paths
   pathLeafiToLeafj = grammar( pathLeafiToLeafj );
   if (i == j) M[i][j] = 0;
   else
    {
 String leaf = "" + pathLeafiToLeafj.charAt(0);
 int p = leaves.indexOf(leaf);
 distance = lengthsLeaves[p];
 int n = pathLeafiToLeafj.length();
 for(int k = 1; k < n-1; k++)
 {
  innerNode =  "" + pathLeafiToLeafj.charAt(k);
  p = internalNodes.indexOf(innerNode);
  distance = distance + lengthsInnerLinks[p];
 }
 leaf =  "" + pathLeafiToLeafj.charAt(n-1);
 p = leaves.indexOf(leaf);
 distance = distance + lengthsLeaves[p];
  M[i][j] = distance;
    }
  }
 return  M;
}



//Translation from Newick's to quasi-binary notation:
//addresses are attached to tree in Newick's notation
//Inner nodes are conserved. Lengths are conserved.
private static String toQuasiBinaryInner(String tree)
{
if (print2)
System.out.println("Input tree at toQuasiBinaryInner " +
                         "= \n" + tree);
//Output
```

```
String treeToBi = "";
String numeric = ".:0123456789";
//The binary address of leaf
String address = "";
//The name or content of the tree
String content = "";
String thisChar = "";
String nextChar = "";
String previousChar = "";
//length of String   tree
int l = tree.length();
//Tree is scanned char by char
for(int i = 0; i < l; i++ )
{
thisChar = "" + tree.charAt(i);
if ( i < l-1) nextChar = "" + tree.charAt(i+1);
else nextChar = "end";
if ( i > 0) previousChar = "" + tree.charAt(i-1);
else previousChar = "start";


//This char = (
if (thisChar.contentEquals("("))
 {
  content = content + thisChar;
  address = address + '+';
  if (print)
 System.out.println("Address = " + address);

 }


 //If this char is a leaf
 if (leaves.contains(thisChar))
  {
   // System.out.println("leaf");
   treeToBi =  treeToBi + content + address + thisChar;
   content = "";
```

```
  }

 if (numeric.contains(thisChar))
 treeToBi =  treeToBi + thisChar;

 //If this char is an inner node
 if (internalNodes.contains(thisChar))
  {
   int m = address.length();
//delete last bit from the right
   if (m>0) address = address.substring(0, m-1) ;
   if (address.contentEquals("")) address = "+";
   treeToBi =  treeToBi + address + thisChar;

  }


 //This char = , and goes after an inner node
 if  (thisChar.contentEquals(","))
  {
   //System.out.println
 //("comma : "  + previousChar + thisChar );
   treeToBi =  treeToBi + thisChar ;
   if (print)
 System.out.println("Address at ,1   = " + address);

   if    (numeric.contains(previousChar))
  address = address + '-';//Add a - to the right
   if (print)
 System.out.println("Address at ,2   = " + address);
  }


 if (thisChar.contentEquals(")"))
  {
treeToBi =  treeToBi + thisChar ;
    //Diphthonge   ),
if   (nextChar.contentEquals(","))
```

```
{
     int m = address.length();
 //delete last bit from the right
 if (m >0)   address = address.substring(0, m-1) + "-";
   }
  }

}//End of for(int i...
return treeToBi;
}


//bTree  (in links with lengths notation)
//is used to generate Record[]
//with addresses and contents.
//Names of inner nodes are visible
public static void makeInputInner(String bTree)
{
int counter = 0;
//length of String   tree
int l = bTree.length();
String thisChar = "";
String address = "";
String binNumbers = "+-";
String numeric = ".:0123456789";
//String tree is scanned char by char
for(int i = 0; i < l; i++ )
{
 thisChar = "" + bTree.charAt(i);
if (binNumbers.contains(thisChar))
   address = address + thisChar;
if ( (leaves.contains(thisChar)) ||
(internalNodes.contains(thisChar)))
{
 Record[counter] = address;
     counter++;
 Record[counter] = thisChar;
```

```
 counter++;
 address = "";
    }

if (numeric.contains(thisChar))
 {
Record[counter-1] = Record[counter-1] + thisChar;
 }

}


if (print)
for(int i = 0; i < counter; i = i+2 )
  System.out.println(Record[i] +
 " bbbb " + Record[i+1]);

lastUsedPosition = counter;
if (print)
System.out.println("lastUsedPosition ==== "
      + lastUsedPosition);
}//End of method


//Prints  a matrix and its title
private static void printMatrix(double Matrix[][],
                        String title)
{
System.out.println( "\n" + title );
int m = Matrix.length;
System.out.print(  "\t");
for ( int j = 0; j < m; j++ )
  System.out.print( leaves.charAt(j)+ "\t");
System.out.println();
for ( int i = 0; i < m; i++ )
 {
  System.out.print(leaves.charAt(i ) + "\t");
  for ( int j = 0; j < m; j++ )
```

```
 System.out.print( Matrix[ i ][j]+ "\t");
  System.out.println();
 }
}


//Tree is translated
//from link notation to binary.
//Output in Record[]: even entries= addresses, odd = names.
//Associated matrix is also calculated
public static double [][] associatedMatrix(String tree)
{
 /*
 System.out.println("Tree  at translate = " + tree);
 //String NwTree = simplify(tree);
 String bTree = toBinary(tree);
 //A vector with addresses and names is composed
 makeInput(bTree);
 */
//String NwTree = simplify(tree);
 String bTree = toQuasiBinaryInner(tree);

 if (print2)
 {
 System.out.println("Input tree  at translate = "    + tree);
  System.out.println("Selected tree in " +
  "pseudo-Newick's notation = \n" + simplify(tree));
  System.out.println("Tree in quasi-binary notation: \n"
              + bTree);
 }

 //A vector with addresses and other with names are composed
 makeInputInner(bTree);
 //Paths from leaves to node a are found.
 //Reverse paths are also listed.
 pathFinder(bTree);
 //The distance matrix among leaves is calculated
 double M[][] = matrixBuilder();
```

```
 return M;
}


//=========================================================
//================== Evolution of fitting =================
//Lengths are added to given tree topology and evolved
//to find the best fitting of  topology
//to target distance matrix.
//Returned value at evolveLengths(String topology) =
//the distance between
//target matrix and the matrix associated to
//the best fitting of this topology.
//=========================================================



//Population of lengths is initialized at random.
//Lengths take values on digital numbers 0,...,9
private void initialize()
{
 for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = r.nextInt(maxLength);

}

//Recently generated new population becomes the old one
private static void newToOld()
{
for(int i = 0; i < N; i++)
 for(int j = 0; j < 2; j++)
 for(int k = 0; k < maxNumberOfLeaves; k++)
 oldLengths[i][j][k] = newLengths[i][j][k];

}
```

```
//Old lengths are copied to a service vector
private static void oldToService(int i)
{

for(int k = 0; k < numberOfLeaves; k++)
 {lengthsLeaves[k] =  (int) oldLengths[i][0][k];}

for(int k = 0; k < numberOfLeaves; k++)
{lengthsInnerLinks[k] =  (int) oldLengths[i][1][k];}
}


//The absolute distance among two matrices is calculated.
//Only triangular part of matrices are compared.
private static double absDistance(int n,
               double[][] M1, double[][] M2)
{
double abs = 0;
for(int i = 0; i < n; i++)
for(int j = i+1; j < n; j++)
 abs = abs + Math.abs(M1[i][j]-M2[i][j]);
return abs;
}


//Input : tree in link notation.
//Output:
//distance matrix associated to the leaves of the tree.
private static double[][] distanceMatrix(String tree)
{
String s = toQuasiBinaryInner(tree) ;
if (print4)
 System.out.println("\n Tree = " + s);
//Paths from leaves to node a are found.
//Reverse paths are also listed.
pathFinder(s);
//The distance matrix among leaves is calculated
double M[][] = matrixBuilder();
```

```
return M;
}



//A tree topology is received as input.
//Lengths from oldLengths are inserted into the topology
//to get a tree with lengths.
//The distance between target matrix and
//the associated distance matrix to tree is calculated.
//The minimal distance is found:
//this renders the fittest tree.
private double findDistances(String topology)
{
 for(int i = 0; i < N; i++)
 {
  //Data preparation
  oldToService(i);
  //Lengths of links are added to tree topology
  String tree = insertLengths(topology);
  if (print4)
 System.out.println("\n Tree = " + tree);
  double M[][] = distanceMatrix(tree);
  if (print4)
   {
 //The tree generates a distance matrix
 //among pair of leaves
 printMatrix(M,
 "Distance Matrix associated to   leaves"
 + " of  tree");
}
  //The absolute distance from M to target matrix:
  double d= absDistance(numberOfLeaves,targetMatrix, M);
  if (print3)
  System.out.println("\n Individual " + i + ":\n" + tree);
  //Distance from the associated distance matrix of the tree
  //to target matrix is evaluated
```

```
 distance[i] = d;
 if (print4)
 System.out.println(i + " Dist to tarrrrrrget = "
                                  + distance[i]);
 //If solution has been found, halt.
 if (d == 0)
 {
  System.out.println( "\n\nSOLUTION HAS BEEN FOUND: \n"
  + " distance to target = " + d
  + " \nSolution tree = " + tree);
  printMatrix(M, "Distance matrix associated to tree");
  printMatrix(targetMatrix, "Target matrix");
  System.exit(0);
 }


}//End of for(int i...

//The fittest individual minimizes distance function
double min = distance[0];
best = 0;
for(int i = 1; i < N; i++)
{
 if (distance[i] < min )
 {
  min = distance[i];
      best = i;
 }
}
//lengths are copied to  service vectors
oldToService(best);
//Best tree for the given topology
bestTree = insertLengths(topology);



if (print4)
{
```

```
  System.out.println("\n Distance + BestTreeeeeee  = " +
  + min + " \n"  + bestTree
             + "  "  + "\n");
 }
 return min;
}

//Selection is implemented
private void makeFitnesses()
{
 double maxD = distance[0];
 double minD = distance[0];
 double b = 1 ;
 //Min and max distances are found
 for(int i = 1; i < N; i++)
  {
if ( distance[i] > maxD ) maxD =  distance[i];
if ( distance[i] < minD ) minD =  distance[i];
  }

 if (minD == maxD) //All   equally fit
   {
  for(int i = 0; i < N; i++)
fitness[i] = 1;
   }
 else
  {
 //Distance to target determine fitness:
//fitness[maxD] = 0;    fitness[minD] = 1;
 for(int i = 0; i < N; i++)
 {
  b = 1;
  b = b/(minD - maxD);
  double a = -b*maxD;
  fitness[i] = a + b * distance[i];
  if (print3)
   {
  System.out.println("Individual = "
```

```
            + i + " has distance " +
          distance[i] + " fitness " + fitness[i]);
   }
}
  }
}
//Distances are translated to
//probabilities of not reproducing.
//N individuals are sampled at random from the population
//according to those probabilities
private void reproduction()
{
 makeFitnesses();
 //The best is cloned
 for(int j = 0; j < numberOfLeaves; j++)
{
 newLengths[0][0][j] = oldLengths[best][0][j];
 newLengths[0][1][j] = oldLengths[best][1][j];
}

 //N-1 children are sampled
 for(int i = 1; i < N; i++)
  {
 //flag
 boolean go = true;
   while (go)
    {
 //An individual is chosen at random
 int k = r.nextInt(N);
 //k is reproduced according to its fitness
 double q = r.nextDouble();
 if (print4)
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] );
 if ( q < fitness[k]) //make a copy of k
  {
   for(int j = 0; j < numberOfLeaves; j++)
```

```
{
 newLengths[i][0][j] = oldLengths[k][0][j];
 newLengths[i][1][j] = oldLengths[k][1][j];
}
 go = false;//Pass to making a new individual
   if (print4)
 System.out.println("Individual = "
                      + k + " with fitness " +
 fitness[k] + "->" + i);
  }

    }

  }//End of for //N-1 children are sampled
}



//Individuals are mutated
private void mutation()
{
 //The old best, i = 0, is conserved intact
 for(int i = 1; i < N; i++)
   {
//Mutation in links that connect leaves
for(int k = 0; k < numberOfLeaves; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][0][k] = r.nextInt(maxLength);

 }
//Mutation in links that connect inner nodes
for(int k = 0; k < numberOfLeaves-2; k++)
 {
double q = r.nextDouble();
if (q < m) //mutate
 newLengths[i][1][k] = r.nextInt(maxLength);
```

```
 }
   }
}


//Individual l is recombined to another
//that is chosen at random
private static void recInd(int l)
{
 //leaf else inner node?
 int a = r.nextInt(2);
 //Partner at random
 int k = r.nextInt(N);

 if (a == 0)
  {
   //Random site for recombination
   int site = r.nextInt(numberOfLeaves);
   //Take a part from l
   for(int j = 0; j < site; j++)
  newLengths[l][0][j] = oldLengths[l][0][j];
   //Another part from k
   for(int j = site; j < numberOfLeaves; j++)
  newLengths[l][0][j] = oldLengths[k][0][j];
  }
 else
 {
  int site = r.nextInt(numberOfLeaves-2);
  for(int j = 0; j < site; j++)
  newLengths[l][1][j] = oldLengths[l][1][j];
  for(int j = site; j < numberOfLeaves; j++)
 newLengths[l][1][j] = oldLengths[k][1][j];
}
}
```

```
//Recombination master
private static void recombination()
{
if (print) System.out.println("\nRECOMBINATION\n");
newToOld();
//Individual l>0 is a recombinant version
for(int l=1; l< N; l++)
{
  recInd( l );
}
}
```

```
//Lengths are evolved to find the best fitting
//of the given topology to target matrix
private double evolveLengths(String topology)
{
 if(print4)
 System.out.println("\n========Evolution of lengths " +
  "for topology = "   + topology );
 initialize();
 double min = -1;
 for(int gen = 0; gen < numberOfGenerations; gen++)
 {
  if (print4)
System.out.println("\n=== Gen = " + gen);
  min =  findDistances(topology);
  reproduction();
  mutation();
  recombination();
  newToOld();
 }
 return min;
}
```

```
//=========================================================
//==================Inner service class====================
//=========================================================



class DynamicTree13 extends JPanel
{

 private static final long serialVersionUID = 1L;

 protected DefaultMutableTreeNode rootNode;
 protected DefaultTreeModel treeModel;
 protected JTree tree;
 //Next leaf to be grafted
 char nextLeaf = leaves.charAt(3);
 //Next internal node to be inserted
 char nextIntNode = internalNodes.charAt(1);
 //counter of generations
 int counterGen = 0;


 //Constructor
 public DynamicTree13()
 {
    super(new GridLayout(1, 0));

    rootNode = new DefaultMutableTreeNode(seed);
    treeModel = new DefaultTreeModel(rootNode);

    tree = new JTree(treeModel);
    tree.setEditable(true);
    tree.getSelectionModel().setSelectionMode(
        TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);
```

```
  JScrollPane scrollPane = new JScrollPane(tree);
  add(scrollPane);
}


/** Selected tree is unveiled */
public String unveilTree()
{
  DefaultMutableTreeNode selectedNode = null;
  TreePath selectedPath = tree.getSelectionPath();

  if (selectedPath == null) {
   selectedNode = rootNode;
  } else {
   selectedNode = (DefaultMutableTreeNode)
           (selectedPath.getLastPathComponent());
  }

  String tree =  selectedNode.toString();
  if (print)
  System.out.println("\nSelected tree at unveilTree() = "
                      + tree);
  return tree;
}


//Add a node
public DefaultMutableTreeNode
             addObject(DefaultMutableTreeNode parent,
    Object child, boolean shouldBeVisible) {
  DefaultMutableTreeNode childNode =
              new DefaultMutableTreeNode(child);

  if (parent == null) {
    parent = rootNode;
  }
```

```
    // It is key to invoke this on the TreeModel,
    //and NOT DefaultMutableTreeNode
    treeModel.insertNodeInto
              (childNode, parent, parent.getChildCount());

    // Make sure the user can see the lovely new node.
    if (shouldBeVisible) {
      tree.scrollPathToVisible
                    (new TreePath(childNode.getPath()));
    }
    return childNode;
  }



 //Leaves are found and children are generated.
 private DefaultMutableTreeNode reproductionOfLeaves()
  {
   if (print)
 System.out.println("counterGen = " + counterGen );

   DefaultMutableTreeNode node = null;

   Enumeration<?> e = rootNode.depthFirstEnumeration();
   indexOfTopologies = 0;

while  (e.hasMoreElements())
 {
  //Nodes are found
  node = (DefaultMutableTreeNode) e.nextElement();

  //Nodes are checked: are you a leaf?
  if(node.isLeaf())
   {
String parentTree = node.toString();
if (print)
System.out.println("ParentTree = " +  parentTree );
```

```
//=================

//Next leaf to be grafted
char nextLeaf = leaves.charAt(3+counterGen);
//Next internal node to be inserted
char nextIntNode =
internalNodes.charAt(1+counterGen);
//link to be replaced
String link = "";
//String link is replaced by what:
String replacement = "";
//new tree generated by replacement
String newTree = "";
//length of parent tree
int l = parentTree.length();
for(int i = 0; i < l; i++ )
 {
   //parentTree is scanned char by char
char I = parentTree.charAt(i);
//Is I an internal node for a replacement?
if (charIsInternalNode(I))
 {
  //Check for the char before I
  char a = parentTree.charAt(i-1);
  //If I links to a complex structure
  if (a == ')') newTree = useRule2(parentTree,
           i,   nextLeaf, nextIntNode);
  else //if I links to a leaf
   {
//Link to be replaced
link = "" + a + I;
//Replacement rule 1:
//aI -> (aN,FN)I;   N= nextIntNode;
//F = nextLeaf
replacement = "(" + a + nextIntNode + ","
        + nextLeaf + nextIntNode + ")" + I;
//Rule 1 is executed
newTree =
```

```
parentTree.replace(link, replacement);

   }
 if ((4+counterGen)== numberOfLeaves)
 {

  //Lengths are added and evolved to find the
  //best fitting of this topology
  //to target distance matrix.
  //Returned value: the distance between
  //target matrix and the matrix associated to
  //the best fitting of this topology.

  double distance = evolveLengths(newTree);
  bestFitting[indexOfTopologies] = distance;

  System.out.println("\n====================" +
   "\nIndex + Topology + " +
   "distance to target + best tree: \n"
  + indexOfTopologies
   +  ":  "  +
  newTree + " -> "   + simplify(newTree)
 + " " +  distance  + "\n" + bestTree + "\n");

  bestTreeOfTopology[indexOfTopologies]
                      = bestTree;

 if (print5)
 {
//Distance from M(bestTree)
 //to target matrix is evaluated
 double M[][] = distanceMatrix(bestTree);
  if (print5)
{
 printMatrix(M,
 "Distance Matrix associated to leaves"
 + " of  tree");
}
```

```
  double d= absDistance(numberOfLeaves,
                   targetMatrix,
M);

System.out.println("Dist to target = " + d);
 }
  indexOfTopologies++;
 }

String child = newTree;
    addObject(node, child, true);
}

 }//End of for(int i...


   }//End of if(node.isLeaf())

 }//End of while
counterGen++;
return null;
  }


  //Listener
  class MyTreeModelListener implements TreeModelListener
  {
    public void treeNodesChanged(TreeModelEvent e)
    {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)
         (e.getTreePath().getLastPathComponent());

      /*
       * If the event lists children,
       * then the changed node is the child of the
       * node we've already gotten.
       * Otherwise, the changed node and the
```

```
         * specified node are the same.
         */

        int index = e.getChildIndices()[0];
        node = (DefaultMutableTreeNode)
                   (node.getChildAt(index));

        System.out.println(
          "The user has finished editing the node.");
        System.out.println("New value: " +
                    node.getUserObject());
      }

      public void treeNodesInserted(TreeModelEvent e)
      {
      }

      public void treeNodesRemoved(TreeModelEvent e)
      {
      }

      public void treeStructureChanged(TreeModelEvent e)
      {
      }
    }
}//End of DynamicTree13



//=========================================
//==Inner class  for drawing selected tree ==
//=========================================


//This program translates a string that
//denotes a tree in Newick's notation
//to our binary notation
//(assign a zero to the left, a one to the right).
```

```
//A generator of random trees in Newick’s notation
//is included.

//Resultant trees are drawn.




class DrawBiTree10 extends JPanel

{
private static final long serialVersionUID = 1L;
private JPanel drawingPane30;
JScrollPane scroller;




//=======================================



//Vertical  size of the panel
 int verticalSize;
 int horizontalSize;
//Scale must be a power of 2
 int scale = 1;

//Symbolic horizontal length of each stroke
 int horizontalLength = 30;

int verticalLength = 10;
```

```
  String  NwTree = "";


 boolean print = false;


//This is our encoding for a tree in our binary notation:
//We use a vector of strings:
//Even entries are the paths in the tree  of the nodes,
//odd ones are the strings, names or contents.


//Y-coordinates of points where strings are drawn
 int[] yCoordinate = new int[1000];


//Set up the instructions.
JLabel instruct1;
JLabel instruct2;

public DrawBiTree10()
{
super(new BorderLayout());


JPanel instructionPanel = new JPanel(new GridLayout(0,1));


//Set up the instructions.
instruct1 = new JLabel(title1);
instruct2 = new JLabel(title2);

instructionPanel.setFocusable(true);
instructionPanel.add(instruct1);
```

```
instructionPanel.add(instruct2);

//Set up the drawing area.
drawingPane30 = new drawingPane30();
drawingPane30.setBackground(Color.white);

//Put the drawing area in a scroll pane.
JScrollPane scroller = new JScrollPane(drawingPane30);
scroller.setPreferredSize(new Dimension(500,900));
scroller.revalidate();

//Lay out this demo.
add(instructionPanel, BorderLayout.PAGE_END);
add(scroller, BorderLayout.CENTER);
}


//=========================================
//============= Painting class ============
//=========================================


/** The component inside the scroll pane. */
 class drawingPane30 extends JPanel
{
private static final long serialVersionUID = 1L;

//Initialization of coordinates
int centerX = 1;
int centerY = 1;
//Coordinates of the origin
int x00;
int y00;
//Target coordinates
int x11 ;
int y11;
//Horizontal and basic vertical strokes
int H;
```

```
int V;
//Addresses are interpreted as paths
String path = "";
char direction;



//Calculate the size of a box to hold the tree
//Scrolling controls are managed
private void sizeOfTree(Graphics2D g2d)
{
//Calculate the dimension of the strings
//All strings have equal size.

//Get   font in use
Font font = g2d.getFont();
//get metrics from the graphics
FontMetrics metrics = g2d.getFontMetrics(font);
//get the advance of my text in this font
//and render context
int lengthText = metrics.stringWidth("Aa") + 2;
//get the height of a line of text in this
//font and render context (some padding is included)
int hgt = metrics.getHeight()  + 8;
if (print)
System.out.println("height of letters = " + hgt);

//Horizontal and basic vertical strokes
H = scale * horizontalLength;
V = scale * verticalLength;

horizontalSize = lengthText +
              (int) (H *  numberOfLeaves);
verticalSize =
 (int) (hgt *   numberOfLeaves);
```

```
//area = size of the graphic (width, height)
Dimension area =
 new Dimension(horizontalSize,12*verticalSize);
//area is set as the preferred size of the graphic
drawingPane30.setPreferredSize(area);
//The graphic demands a space for itself:
//if the alloted space is not enough,
//turn on scrolling bars.
drawingPane30.revalidate();
verticalLength = 2*verticalSize;
//Coordinates of the origin
x00 =    scale*centerX;
y00 =   scale* centerY + verticalSize  - 600;
//Target coordinates
x11 =  x00 + (int) ( scale*horizontalLength);
y11 =  y00;

}



//Tree is drawn
private void DrawTree2(Graphics2D g2d)
{
//The tree is drawn:
//A path from the origin of the tree to
//each leaf is drawn
 if (print)
 System.out.println("LastUsedPositoon at DrawTree2=  "
                              + lastUsedPosition);
 for(int i = 0; i < lastUsedPosition; i = i+2)
  {
//Addresses will be used as paths to  be drawn
path = Record[i];
if (print)
System.out.println( i + " Path = " + path);
//Reinitialization
Dimension center = new Dimension(x11,y11);
```

```
V = scale * verticalLength;
//Path is interpreted
for(int j = 0; j < path.length(); j++)
 {
  direction = path.charAt(j);
  int x0 =  center.width;
  int y0 =  center.height;
  int x1 =  x0;
  int y1;
  if (direction == '+') //go up
   {
    y1 =  y0 + V;
    center.height = center.height + V;
   }
   else // if direction == '-'  =  go down
   {
     y1 =  y0 - V;
     center.height = center.height - V;
    } //End of if + else
    //Draw vertical stroke
    g2d.drawLine(x0, verticalSize -y0, x1,
     verticalSize- y1);
    //Draw horizontal stroke
    g2d.drawLine(x1, verticalSize - y1,
        x1 + H,  verticalSize - y1);
    //Move center
    center.width = center.width + H ;
    //Vertical stroke is divided in two
    V = V/2;
    }//End of for j
 }  //End of for: numberOfLeaves
}



//Strings are drawn
private void drawStrings(Graphics2D g2d)
{
```

```
int counter = 0;
//Seed is not  drawn: we lack one
//g2d.drawString(seed,x00,verticalSize - y00);
yCoordinate[counter] = verticalSize - y00;
//counter++;
//Strings are drawn:
for(int i = 0; i < lastUsedPosition; i = i+2)
{
 //Addresses will be used as paths to the places
 //where strings must be drawn
 path = Record[i];
 String s = Record[i+1];
 //Reinitialization
 Dimension center = new Dimension(x00,y00);
 Dimension target = center;
 V = scale * verticalLength;
 //Path is interpreted
 for(int j = 0; j < path.length(); j++)
{
 direction = path.charAt(j);
 if (direction == '+') //go up
 target.height = target.height + V;
 else // if direction == '-'  =  go down
 target.height = target.height - V;
 target.width = target.width + H ;
 V = V/2;
}//End of for j
//String is drawn
g2d.drawString(s, target.width + 1,
                      verticalSize - target.height-1);
//y-coordinates are recorded
yCoordinate[counter] = verticalSize - target.height-4;
 counter++;
  }
}


//Basic routine of the drawing procedure
```

```
private void drawBasic(Graphics2D g2d)
{
g2d.setColor(Color.red);
//Stem is drawn: our trees is artificially rooted.
g2d.drawLine(x00, verticalSize - y00,
      x11,  verticalSize - y11);
sizeOfTree(g2d);
DrawTree2(g2d);
drawStrings(g2d);
}

//Graphical  master : it is called automatically
//at starting and by repaint()
protected void paintComponent(Graphics g)
{
super.paintComponent(g);
Graphics2D g2d = (Graphics2D)g;
//Font to draw strings
Font font = new Font("Arial", Font.PLAIN, 12);
g2d.setFont(font);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  RenderingHints.VALUE_ANTIALIAS_ON);
drawBasic(g2d);
}


}//End of double inner class drawingPane30



/**
* Create the GUI and show it.  For thread safety,
* this method should be invoked from the
* event-dispatching thread.
*/
 void createAndShowGUI()
{
//Create and set up the window.
```

```
JFrame frame =
   new JFrame(titleFrame);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

//Create and set up the content pane.
//Constructor of the main application is used.
JComponent newContentPane = new DrawBiTree10();
//content panes must be opaque
newContentPane.setOpaque(true);
frame.setContentPane(newContentPane);

//Display the window.
frame.pack();
frame.setVisible(true);
}

}//End of  inner class DrawBiTree10

}//End of main class J171 EvoTree3
```

Bibliography

[1] BRASIER M, N MCLOUGHLIN, O GREEN AND D WACEY (2006) A fresh look at the fossil evidence for early Archaean cellular life. Phil. Trans. R. Soc. B. 361, 887–902. doi:10.1098/rstb.2006.1835
http://physwww.mcmaster.ca/~higgsp/3D03/BrasierArchaeanFossils.pdf
Verified 12/XI/2013 11

[2] CASANE D, LAURENTI P ((Feb 2013)) ”Why Coelacanths are not ”Living fossils”: a Review of Molecular and Morphological Data”. BioEssays. doi:10.1002/bies.201200145.
http://onlinelibrary.wiley.com/doi/10.1002/bies.201200145/full
Verified 12/XI/2013 12

[3] GREENWALT D E, GOREVA Y, SILJESTRÖM S, ROSE T, HARBACH R (2013) Hemoglobin-derived porphyrins preserved in a Middle Eocene blood-engorged mosquito. Proc. Natl Acad. Sci. USA pnas.1310885110.
http://www.pnas.org/content/early/2013/10/08/1310885110.abstract
Verified 12/XI/2013 12

[4] LIPSCOMB D (1998) Basics of Cladistic Analysis.
http://www.gwu.edu/~clade/faculty/lipscomb/Cladistics.pdf
Verified 14/XI/2013 207

[5] ORACLE (2013) How to use trees.
http://docs.oracle.com/javase/tutorial/uiswing/components/tree.html
Verified 13/XI/2013 19

[6] PALUMBI S, LESSIOS H (2005) Evolutionary animation: How do molecular phylogenies compare to Mayr’s reconstruction of speciation patterns in the sea? 6566 – 6572 , PNAS, May 3, 2005, vol. 102, suppl. 1

http://www.pnas.org/content/102/suppl_1/6566.full
Verified 13/XI/2013 15, 489

[7] RENEKER J, LYONS E, CONANT G, PIRES J, FREELING M, SHYU C AND KORKIN D (2012) "Long identical multispecies elements in plant and animal genomes". PNAS 109 (19): 1183–1191. doi:10.1073/pnas.1121356109.
http://www.pnas.org/content/109/19/E1183.abstract
Verified 12/XI/2013 12

[8] SALEMI (2009) Programs to display and manipulate phylogenetic trees. Pag 162. In: *The Phylogenetic Handbook*. Edited by Lemey P, Salemi M, Vandamme A. Cambrige University Press.
240

[9] SILVA (2013) Ribosomal RNA sequence data .
http://www.arb-silva.de/
Verified 12/XI/2013 12

[10] SWOFFORD D, SULLIVAN J (2009) Phylogeny inference based on parsimony and othermethid using PAUP. Pag 268. In: *The Phylogenetic Handbook*. Edited by Lemey P, Salemi M, Vandamme A. Cambrige University Press. 241

[11] UCBASE 2.0 (2013) .
http://www.dsb.unimo.it/UCbase/index.html
Verified 12/XI/2013 12

[12] UNDERSTANDING EVOLUTION TEAM (2013) Phylogenetic systematics, evolutionary trees.
http://evolution.berkeley.edu/evolibrary/article/0_0_0/phylog
Verified 14/XI/2013 207

[13] WAGGONER B, B R SPEER (2013) Bacteria: Fossil Record http://www.ucmp.berkeley.edu/bacteria/bacteriafr.html.
http://www.ucmp.berkeley.edu/bacteria/bacteriafr.html
Verified 12/XI/2013 11