JAVA FOR THE STUDY OF EVOLUTION

VOLUME IX

THE IMMUNOLOGICAL DISTANCE

José del Carmen Rodríguez Santamaría
The EvolJava Community

Contents

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 ninth volume: The immune system is a biochemical machinery that distinguishes self from non-self. Its way of operation is abstracted here into a technology that is known as n-gram analysis. We show how it is used in linguistics to identify languages and to measure language divergence, and in evolutionary genetics to propose distances between pairs of complete proteomes, which can then be plugged into a phylogenetic tree.

Bogotá, Colombia, *José Rodríguez*
May 2013

Introduction

The immune system can distinguish self from non-self. Step by step and along the first three chapters we elaborate this discriminating capability into a technology that is known as **n-gram analysis** of texts and that we apply to build language recognizers. In chapter IV we show how we can apply it to the study of the **proteome**, a complete set of proteins of a species. Chapter V presents a hindsight.

Chapter 1

Metrics

`Measuring differences`

1 ***Purpose.*** *A **distance** measures how different in a given respect are the elements of a given set. A distance associates a real number to every pair of elements, the different, the larger is that number. DNA and protein molecules can be given many diverse distances, as we show below. Next, we begin our study of immunological distances, which are induced by the way the immune system works. In fact, this system discriminates self from non-self, so it ipso facto defines a distance.*

1.1 Distances

To measure differences among objects, we can use distances. These are functions that assign a non-negative number, a distance, to a pair of elements of a given set. The design of a distance must obey a simple rule: the different are the objects, the larger shall be the associated number. This means that one has an instinctive way of measuring differences and the purpose of design is to propose a function that somehow copies the intuitive rule. But, how can one know that this creative work is correct?

2 ***Testing a proposal for a distance***

If one wants to make sure that a proposed distance is correct, then we have a test: every distance d must comply with the following conditions:

1. $d(a,a) = 0$: there is no distance in within one object and itself.
2. $d(a,b) = d(b,a)$: the distance between a and b is as large as that between b and a.

3. $d(a,b) \leq d(a,c) + d(c,b)$: the direct distance is less or equal than any other indirect one.

There are many forms as we can measure differences among DNA strings. To see that, let us consider the next strings

```
a = ATTATTCTGGC
b = ATTATTCTGGC
c = ATTATTATGGC
d = ATTATTA
```

3 ***The head to head alignment***

Before measuring distances between two sequences, one must align them. Sequences can be aligned in various ways. The default one is the **head to head alignment**, in which we align the letters site by site beginning from their origin at the left. Thus, the alignment is determined by the first letters, which are posited head to head. All others letters are compared shoulder to shoulder. If the strings have different length, we fill shorter strings with blanks and count a blank as a symbol. Unless otherwise specified, we adopt the head to head alignment.

4 ***The discrete distance***

To measure the difference between two strings, we compare them letter by letter. If all letters are the same, we declare the strings to be equal and define their distance as 0. If it happens that we find at least a mismatch, the distance is defined as 1.

The **discrete distance** is defined by

$$discrete(x,y) = \begin{cases} 0 \quad if \quad \text{x=y} \\ 1 \; otherwise. \end{cases}$$

So, this distance assigns a zero when the two strings coincide and 1 when they differ in at least one site.

For the strings that have been listed above, we have $discrete(a,b) = 0$ while $discrete(a,c) = discrete(c,d) = 1$.

5 ***Challenge.*** *Test discrete to see if it fits into our requirements of a distance.*

6 ***The mismatch distance***

The **mismatch distance**, *mismatch*, counts the number of mismatches that happen when comparing two strings letter by letter. So:

$mismatch(a,b) = 0.$
$mismatch(a,c) = 1.$
$mismatch(a,c) = 1.$
$mismatch(a,d) = 5.$
$mismatch(c,d) = 4.$

7 ***Challenge.*** *Test mismatch to see if it fits into our requirements of a distance.*

8 ***The first mismatch distance***

To measure the difference between two strings along the **first mismatch distance**, we compare them letter by letter. If all letters are the same, we declare the strings to be equal and define their *firstMismatch* distance, as zero. If it happens that we find a mismatch, let f the site where it happens. The distance in this case is defined as $1/(1+f)$. We used the default convention of beginning counting with 1 (in Java, counting begins with 0). So, for *a* and *c*, $firstMismatch(a,c) = 1/8$ because the first mismatch happens at place number 7. On the other hand, $firstMismatch(c,d) = 1/9$.

9 ***Challenge.*** *Test $firstMismatch$ to see if it fits into our requirements of a distance.*

10 ***The best alignments***

Let us consider the sequences TTTATATGGG and ATAT.
The head to head alignment between these two strings is the next:

```
TTTATATGGG
ATAT
```

The mismatch distance is 3. The next alignment renders a mismatch distance equal to 2

```
TTTATATGGG
    ATAT
```

These two string perfectly match one another if they are aligned as follows:

```
TTTATATGGG
   ATAT
```

Thus, we define the **best alignment** as an alignment that for the mismatch distance and for a given pair of strings produces the minimal distance when it is restricted to the shorter substring (so, we add no blanks).

11 ***Challenge.*** *Give examples that show that the best alignment is not unique.*

12 ***The loop distance***

Let us consider the next two strings ATATTCGTATTA and ATATTATTA. If we align them as follows

```
ATATTCGTATTA
ATAT    TATTA
```

we see that they perfectly match one another. We want to define the loop distance between these two strings as zero. So, the **loop distance** is based on the mismatch distance but includes the best alignment given that strings can be divided in various substrings. Now, the name of this distance is due to the fact that one can imagine that the unmatched subsequences form loops or handles to enable the asked alignment in a three dimensional space. For a more realistic fitting of the physics of DNA molecules, we might include other two concepts:

13 ***Complementing procedure***

One might be interested not in comparing two sequences but in comparing a given sequence with the complement of the other. The **complementing procedure** interchanges A with T and C with G. The complement of b is denoted C(b). Thus, for a= TTTATATGGG and b= ATAT, we find c(b) = TATA and then use a distance to measure the distance between a and c(b). For the case of the best alignment distance, we get a distance equal to zero:

```
a    = TTTATATGGG
c(b) =   TATA
```

14 ***Challenge.*** *Let us propose the distance* $cmplmnt(a,b) = mismatch(a,c(b))$. *Test over it our requirements of a distance. If not, made an amendment to concoct a correct definition and change the name distance by one more appropriate.*

15 ***Reversing procedure***

Real DNA molecules are oriented as rivers and a downstream direction is different that the up stream one. When two real DNA molecules match one another they do so in contrary directions. This motivates our next definition. The **reversing procedure** rewrites a sequence in reverse order. The reverse of `b` is denoted as `r(b)`. Thus, for `b= ATCG`, we find `r(b) = GCTA`. We also can take the complement of `r(b)` to get `cr(b) = CGAT`. Now, we can use a distance to measure the distance between `a= TTTATATGGG` and and `cr(b)`. For the case of the best alignment, we get

```
a     = TTTATATGGG
cr(b) =  CGAT
```

which renders a distance of 2.

16 ***Challenge.*** *Let us propose the distance* $rev(a,b) = mismatch(a, cr(b))$. *Test it to see if it fits into our requirements of a distance. If not, made an amendment to concoct a correct definition and change the name distance by one more appropriate.*

17 ***Comments***

- The listed forms of measuring differences among DNA sequences might be inspired by mathematical curiosity or by biophysical insight. For instance, the first mismatch distance is mathematical in origin and plays an important role in the theory of languages. But if one involves complementing, reversing and loop distances, one is thinking of real DNA molecules that swing in a liquid milieu and try to bind one with another following the complementarity dictated by the H-bonds: `A` and `T` form one binding pair while `C` and `G` form the other.

- The mismatch distance of best alignments might correlate with the binding energy of the given molecules because that energy depends in principle from the number of complemented base pairs. The binding energy measures the strength of the bonds among molecules.

- Our mathematical definitions are targeted to nude DNA that has been separated from a cell. But nude DNA has nothing to do with biology: nude DNA lacks genes and regulatory sequences that are the elementary concepts that connect molecular biology with life.

- If nude DNA has nothing to do with life, why are we interested in studying DNA distances? The reason is that DNA is the carrier of the software that instructs extant life to produce and unfold more life. Thus, we *predict* that if there is a systematic difference among members of different species, that difference must be reflected in the code that they rely upon and also in the nude DNA that serves as carrier of genetic information.

- To conclude, our primary interest resides in the species and we use molecular distances as tools, whose power and fairness *are not given a priori but must be investigated*.

18 ***Exercise.*** *Figure out how to use distances to implement an orthographic corrector.*

1.2 Immunological distances

Nature has a natural system to distinguish self from non-self and so it defines a discrete distance: the distance between self and self is zero but between self and non self is one. We have in mind the immune system so, let us review that part of it that is important for us.

19 ***The immune system***

The body routinely launches attacks against invading microbes, infected cells, and tumors while ignoring healthy tissues. Were not for this work, one would rapidly die. How is that achieved? The central point is to distinguish self from non self, a task that is fulfilled by the **immune system**.

The fundamental biological directive is that non-self is harmful for the self and so non-self must be prosecuted and destroyed. Why this is so is by no means obvious. To say the least, horizontal transfer in the microbial world and sexual recombination are facts that speak in favor of the thesis that genomes might be happily complemented. Additionally, we have in our intestinal track some bacteria that help in the synthesis of useful compounds. All these subtleties and many more facts make of the immune system one of the most complex objects of Biology. Actually, its complexity seems no lower than that of the system of security of any developed country.

The accepted explanation of how the immune systems distinguishes self from non-self is given by the clonal deletion theory that proposes a learning procedure (NIH, 2013):

1. The original setting of the immune system of the embryo is that everything, self and non-self, must be destroyed.

2. Immune cells give rise to clones over the life of the organism. Each immune cell has a target chosen at random but the population of immune cells is large enough that targeting is almost exhaustive. If a clone is exposed to something early in its life, the representative cells activate an internal self-destructive pathway, and the immune cell dies. If an immune cell matures, it and its clone will get ready to attack its target if it only appears.

3. So, self is what kills young immune clones and non self is what comes next after their maturation.

4. This system works if the embryo is protected by the immune system of the mother from the attacks of every invader.

This theory immediately explains rheumatism as an auto-immune illness: the inner tissues of bones are not exposed to the immune system early in life, when the immune system is in its process of learning. But when late in life a lesion to the bone happens, preferentially at the joins by the accumulation of residues, some proteins that never have been seen by the immune system make their appearance and thus they are taken as alien and so the tissues that contain them are prosecuted to disintegration.

The foreign molecular entities that trigger the operation of the immune system are preferentially proteins, which are encoded by DNA. Nevertheless, the immune system does not deal with entire proteins or enzymes but with small portions of it. Let us see one aspect of how this happens:

An antigen is a molecule that awakes an immune answer. An antibody is a molecule synthesized by the immune system to mark a foreign molecule, an antigen. Now, the unique function of an antibody is its high-affinity binding to a *limited region* of an antigen. This binding involves hydrophobic forces, ionic forces, and van de Waals forces but no covalent bonds are involved (Darnell et al, 1986). The site of an antigen at which an antibody binds is called a determinant. Observe that we have no reason to believe that a determinant corresponds to one uncut DNA substring. Most surely, they correspond to various substrings that end near one to another in space after the antigenic protein folds as a globular object. Nevertheless, every kind of proteins are continuously recycled and one of the first steps is to cut protein to pieces up to 6 or 7 amino acids long. These small strokes are precisely the objects that are revised by the scanning cells of the immune system.

We have now a simple theory:

Immunological recognizing theory*: reading of small DNA or protein segments suffices to discriminate self from non-self.*

This theory is simple enough to be tested with Java immediately. The theory is very interesting because it functions in one and only case: all determinants that result from folding in space of separated portions of the primary structure also come accompanied with determinants that correspond with connected substrings and that will appear in the small strokes of the digested protein that are scanned by the immune system.

20 *Testing the theory with human languages*

Our *immunological recognizing theory* is about proteins and DNA but these as such have no magic at all. With this we mean that every protein or DNA string is a text and that a theory that explains how the immune system discriminates self from non-self by reading protein or DNA fragments must also be useful to discriminate self from non-self in other cases related with texts, no matter whatever nature they might possess. So, the theory must also function if we use written texts of human languages as testbed for our theory. Once we get convinced of its correctness, we will pursue an application to the study of DNA and evolution.

To begin with, our insight seems quite reasonable since after some initial training, one can discriminate at once if a text is written in English or in German. How is that possible? Because one looks for substrings with typical combinations of letters. Say, the substring *freu* does not happen in English, it belongs in German. Thus, we immediately know that the word *freud* is German.

21 *English vs. German*

Let us devise a program in Java that discriminates whether or not a text is written in English. We use texts in English and in German.

To run this and other programs in Java, you need to install Java and Eclipse and to learn how to install a downloaded project. To see how this is done, please, refer to volume V of this series, chapter I, numerals 1.2-1.6 beginning with pag. 21. You have the license of using and misusing all of our material. Very specially, you are encouraged to use latex sources to tailor at pleasure your own pedagogic material. If you want to enjoy the exercises, you need to work out the first volume completely.

The purpose of the program we want to develop is to discriminate whether a given text is written or not in English. We mimic the function of the immune system that works over substrings n-symbols long. Officially, these substring are called **n-grams**. The program learns what English is by examining the n-grams of an input text that is, of course, in English. This training is used next to decide whether or not a new text is in English.

Specifically, the program contains a procedure with four steps:

1. Recording of self: Up to some n, all possible n-grams are generated and recorded in the boolean vector SelfEng. This is done trickily: we generate a boolean vector SelfEng and at the beginning, every entry of SelfEng is set to TRUE. And this is all to it. The idea behind this is that each n-gram is given an index, a number that identifies it in the boolean vector SelfEng. So, recording of Self is done as follows: the index of each n-gram of the trainingText is calculated and the corresponding entry in SelfEng is set to false. This mimics the training stage of the immune system.

2. Test over an English text: Because of the finiteness of the learning text, even another text in English might generate alien n-grams. So, a second text in English is tested against the former one. This defines a calibration stage: the proportion of alien n-grams is reported.

3. Test over a German text: A third text, in German, is analyzed: its n-grams are compared against those in the list that defines the self. The proportion of alien n-grams is calculated. When texts are long, this proportion approximately doubles that of the second text in English.

4. Formulation of the discriminating criterion: a classifying threshold is defined as the arithmetic average of the two former proportions. If the proportion of alien n-grams of a new text is less than that of the threshold, the text is in English, otherwise it is in another language.

22 ***Te code for our English vs. German classifier is the following.*** *In the case of memory problems somewhere along our simulations, one can make suitable adjustments: to shorten the dimensions of arrays, to restrict alphabets to, say, lower cases letters, to use the hard disk instead of RAM. How to use the disk is explained in Vol 1 chapter 11, program A158.*

```
/*Program I22 English
```

```
The purpose of this program is to discriminate
whether or not a given text is written  in English.
We mimic the function of the immune system,
whose work is based in n-grams, that are
string n-symbols long.

The program has four steps:

1. Definition of self:
Up to some n, all possible n-grams are generated
and recorded in the boolean vector
SelfEng. This is done trickily: we generate a
boolean vector SelfEng and at the beginning,
every entry of SelfEng is set to TRUE.
And this is all to it.
The idea behind this is that each n-gram
is given an index, a number that
identifies it in the boolean vector SelfEng.
So, recording of Self is done as follows:
the index of each n-gram of the trainingText is
calculated and the corresponding entry in SelfEng is
set to false.
This mimics the training stage of the immune system.

2.Test over an English text: Because of the finiteness
of the learning text,
even another text in English might generate alien
n-grams. So, a second text in English is tested against
the former one. This defines a calibration stage:
the proportion of alien n-grams is reported.


3. Test over a German text:
A third text, in German, is analyzed:
its n-grams are compared against those in the list
that defines the self.  The proportion of
alien n-grams is calculated. When text are long,
this proportion
approximately doubles
```

```
 that of the second text in English.

 4.Formulation of the discriminating criterion:
 a classifying threshold is defined as the arithmetic
 average of the two former proportions. If the proportion
 of alien n-grams of a new text is less than
 that of the threshold, the text is in English, otherwise
 it is in another language.
*/

public class English
{
 //Number of letters in the n-grams
 private static int n;
 //Number of n-grams
 private static int N;
 //DataBase: if n>3,  the dimension must be increased
 private static Boolean SelfEnglish[] = new Boolean[450000];
 //Training  text. A StringBuffer is a heavy-duty String.
 private static StringBuffer trainingText ;
 //Test  text
 private static StringBuffer testerEng, testerGer;
 //English + German chars (not all)
 private static String Alphabet;
 private static  int lengthAlf;
 private static int nTrainingSample, nTesterEngSample,
                                        nTesterGerSample;
 //Diverse proportions
 private static double propSelf, propTest;
 //Option to print more information
 private static Boolean printAll;

 //Training text is read
 private static void readTrainingText()
 {
  trainingText =  new StringBuffer("Common words such as " +
   "determiners, conjunctions and prepositions seem good " +
   "clues for guessing a language (Johnson 1993). From " +
   "a corpus of documents in a certain language, the " +
   "most frequent words are determined and placed in a " +
```

```
  "model for this language. Each word has a significance " +
  "score based on the frequency of that particular word " +
  "in the corpus. By dividing this frequency by the total " +
  "frequency of all words, we give each word a probability, " +
  "and we can see the language model as a probability " +
  "distribution. Table 1 shows the most frequent common " +
  "words for 10 European languages, as obtained in " +
  "experiments described by Grefenstette (1995).The " +
  "disadvantage of this technique is that, though common " +
  "words occur enough in larger texts, they might not " +
  "occur in a shorter input text.");

}

//A second  text in English is read
private static void readCalText()
{
 testerEng =  new StringBuffer("Rank order statistics: " +
  "This technique, as described by Cavnar and Trenkle " +
  "(1994), determines how far out of place an N-gram is" +
  " in the language models from its place in a document " +
  "model. Since this technique operates on the relative " +
  "indices of features (words or N-grams), these need " +
  "to be sorted in order of frequency. How this technique " +
  "is applied to N-gram features is shown in figure 2. For " +
  "each N-gram in a document model, its counterpart in a " +
  "language model is located, and then calculated how far " +
  "out of place it is. If an N-gram is not in the language" +
  "model, it takes a maximum out-of-place value, which is " +
  "equal to the amount of N-grams in the model. The sum of" +
  " all of the out-of-place values for all N-grams is the " +
  "distance measure for the document from the language. " +
  "The language model with the smallest distance from the " +
  "document represents the language of the document.");
}

//Test text in German is read
private static void readTestingText()
{
   testerGer =  new StringBuffer("Sprachidentifizierung für " +
```

```
  "Handys. Die Entwickler des japanischen Mobilfunkkonzerns " +
  "KDDI haben ein System vorgestellt, mit dem sich der Nutzer " +
  "per Stimmanalyse gegenüber seinem Handy identifizieren " +
  "kann. Die Funktion soll als Alternative zum Passwort " +
  "für Bankgeschäfte oder andere Dienste zum Einsatz " +
  "kommen, teilte das Unternehmen heute in Tokio mit. Der " +
  "Prototyp nahm das Stimmmuster auf und schickte dieses " +
  "digitalisiert an einen Server, wo die Analyse vorgenommen " +
  "wird. Anhand früherer Spracheingaben wird die Identität " +
  "festgestellt, hieß es. Der Prozess nimmt etwa zehn " +
  "Sekunden in Anspruch.");
}


//SelfEng contains all possible n-grams.
private static void initializeSelf(long N)
{
 for(int i = 0; i< N; i++)
  SelfEnglish[i] = true;
}

//Returns the index   of the n-gram 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.
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;
```

```
  correct = false;
 }
   }
  if (printAll)
  System.out.println( "ngram = " + ngram + " index = "
                                              + ind);
  return ind;
 }


 //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);
 c1 = Alphabet.charAt(p);

 indl = indl - p*k;
}
else c1 = Alphabet.charAt(0);
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

   }
  /*
```

```
 if (printAll)
 System.out.println( "ngram = " + ngram +
                  " index = "   + ind);*/
 return ngram;
}



//The boolean vector SelfEnglish records the identity
//of the training text. At the beginning,
//every entry of SelfEnglish is set to TRUE.
//Each n-gram is given an index, a number that
//identifies it in the boolean vector SelfEnglish.
//The index of each n-gram of the trainingText is
//calculated and the corresponding entry in SelfEmglish is
//set to false.
//The variable nTrainingSample decides
//how many chars of trainingText are taken.
private static void DefinitionOfSelf(int n,
                             int nTrainingSample,
                        StringBuffer trainingText)
{
 initializeSelf(N);
 int l1 = trainingText.length();
 int m = Math.min(l1, nTrainingSample);
 int start = 0;
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  trainingText.substring(start,
                                start+n);
ind = index(n,ngram);
SelfEnglish[ind] = false;
start = start +1;
  }

}

//The n-grams of testText are compared against
//those of the trainingText, which were kept at
```

```
//SelfEnglish. The proportion of non-self n-grams
//is reported.
private static double runTest(StringBuffer testText,
                                int nSample)
{
 if (printAll)
 System.out.println( "Alien n-grams");
 int counter = 0;
 int l1 = testText.length();
 int m = Math.min(l1, nSample);
 int start = 0;
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  testText.substring(start,
                               start+n);
ind = index(n,ngram);
if ( SelfEnglish[ind] == true)
{
 counter++ ;
 if (printAll)
 {
  System.out.print(" alien");
 }
}
if (printAll)  System.out.println();
start = start +1;
  }

 double prop = counter;
 prop = prop / (m-n+1);
 return prop;
}

//The training text is studied
private static void learning()
{
 //length of the Alphabet
 lengthAlf = Alphabet.length();
 System.out.println( "Lenght of the alphabet = " +
```

```
                                    lengthAlf);
 System.out.println("\nDEFINITION OF SELFENGLISH");
 //Length of Self
 N = (int) Math.pow(lengthAlf, n); ;
 System.out.println( "Number of n-grams in vector SelfEng = " + N);
 readTrainingText();
 int h = trainingText.length();
 System.out.println( "Lenght of trainingText = " + h);
 System.out.println( "Max numb of chars taken from" +
         " trainngText = " +  nTrainingSample);
 DefinitionOfSelf(n,nTrainingSample, trainingText );
}

//A second text in English is studied
private static void calibration()
{
 System.out.println("\nTEST OVER AN ENGLISH TEXT");
 readCalText();
 int l = testerEng.length();
 System.out.println( "Lenght of testerEng = " + l);
 System.out.println( "Max numb of chars taken from" +
          " testerEng = " +  nTesterEngSample);
 propSelf = runTest(testerEng, nTesterEngSample);
 System.out.println("Proportion of alien n-grams in " +
                           "testerEng = " + propSelf);
}

//A third text in German is classified
private static void testing()
{
 System.out.println("\nTEST OVER A GERMAN TEXT");
 readTestingText();
 int m = testerGer.length();
 System.out.println( "Lenght of testerGer = " + m);
 System.out.println("Max numb of chars taken from" +
           " testerGer =   " +nTesterGerSample);
 propTest = runTest(testerGer, nTesterGerSample);
 System.out.println("Proportion of alien n-grams in " +
                      "testerGer = " + propTest);
}
```

```
 //A condition for discriminating languages is proposed
 private static void conclusion()
 {
  double threshold = (propSelf + propTest)/2;
  System.out.println( "\nDISCRIMINATING RULE: ");
  System.out.print( "A text is written in English" +
                    " if  the proportion of alien " +
     n + "-grams \nis less than "+ threshold + ". " +
                     "Otherwise it is in German.");
 }



 //Test for correctness:
 //The index of a string with a numeric alphabet
 //is actually a number. Its index
 //must be that number in base ten.
 private static void corretnessTest()
 {
  Alphabet = "ab";
  //Alphabet = "0123";
  //Alphabet = "0123456789";
  lengthAlf = Alphabet.length();
  //Length of n-grams
  n = 3;
  int limit = (int) Math.pow(lengthAlf, n );
  for(int i= 0; i < limit; i++)
   {
//String corresponding to index i
String s = indexInverse(n,i);
//index corresponding to string s
int ind = index(n,s);
System.out.println( "String in base " + lengthAlf +" ="
        +  s +  " Index = Number in base 10 = " + ind);
   }
 }


 //Main method
```

```
 public static void main(String[] args)
 {
  Alphabet = "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
"aäbcdefghijklmnoöpqrstuüvwxyzß: ,.-()0123456789";
  //To print all, change to true
  printAll = false;
  //A test for correctness is done.
  Boolean Test = false;
  if (Test) corretnessTest();
   else
{
     //letters per n-gram
     n=3;
     System.out.println( "Letters per n-gram = " + n);
     //How many chars of trainingText must be taken
     nTrainingSample = 800;
     //How many chars of testerEng must be taken
     nTesterEngSample = 800;
     //How many chars of testerGer must be taken
     nTesterGerSample = 10;
     learning();
     calibration();
     testing();
     conclusion();
}
 }
}
```

23 ***Exercise.*** *Play with the code and try to understand it. Notice the form as we try to mimic the immune system. In first place, we do not generate a list of random n-grams, instead, we generate all possible n-grams. Next we use an input English text that is used as teaching material. In general, well written texts are not very redundant and so every piece of text has valuable information and that is why its n-grams are approximately random, i.e. from a given sublist, one cannot predict what will be the next item. In second place, the n-grams of the input text are not recorded anyhow except with a flag in the corresponding index of the vector SelfEng.*

The next remarks are for those that want to understand the indexation procedure that is used in the previous program.

24 ***Indexation of strings***

The purpose of the previous program was to discriminate whether or not a text is in English. The idea was to mimic the immune system that studies n-grams, small strokes of texts. A natural approach would be to make a list of random n-grams and erase -if that is possible- those that belong in the teaching text, the original one. Nevertheless, we generate all possible n-grams up to certain n and then we erase the n-grams of the teaching text. Were the teaching text sufficiently long (a library with various millions chars), a new text could be decided to be not in English if at least one alien n-gram is found. Nevertheless, one can use as teaching text a document 200 words long if we divide it in two teaching texts. The first is the original training text properly and the second is to sample the fraction of alien n-grams that another text in English has with respect to the training one. A text in another language has a fraction of alien n-grams that is expected to be greater than that of the second teaching text. So, we define a threshold proportion as the average of the two proportions of alien n-grams, the first in the contrasting English text and the second in the text in the foreign language.

One can propose the next discriminating rule: an input text is not in English if the fraction of alien n-grams in the test text is greater than the threshold proportion. Otherwise it is in English. This is the **hologramic technique** for language discrimination. The name is due to the resemblance of our procedure with that to make holograms with laser beams, a beam is divided in two parts, the first hits the target and reflected light interferes with the second part over a transducer.

Our program generates no list of n-grams. And more to the point, there is no explicit comparison among n-grams. So, what is done in there?

The key point is that one can replace a n-gram with a number if the correspondence is fair: one number for one n-gram and one n-gram for one number. To fix ideas, let us consider the alphabet formed only with two symbols, say “ab”. Then, a list with all possible 3-grams is:

```
aaa
aab
aba
abb
baa
bab
bba
bbb
```

Let us do now the next trick: replace `a` by `0` and `b` by `1`. We get the next list:

```
000
001
010
011
100
101
110
111
```

This is a list of the first 8 numbers in base 2. So, we can translate them to base 10:

```
000 -> 0
001 -> 1
010 -> 2
011 -> 3
100 -> 4
101 -> 5
110 -> 6
111 -> 7
```

The whole picture now emerges: we got a way to replace in a fair form every n-gram by a number:

```
aaa -> 0
aab -> 1
aba -> 2
abb -> 3
baa -> 4
bab -> 5
bba -> 6
bbb -> 7
```

This correspondence in its general form is an indexation procedure. But, what is its most general form? It corresponds to any desired alphabet when its chars are interpreted as numbers in the natural order: the first char is zero, the second is a symbol for 1, the third for 2 and so on. So, an alphabet is a numeric system in base equal the number of chars that it contains.

In short: look at the alphabet as at a numeric system and at a string as at a number. That is all to it. The associated number to a n-gram is the *index*, which is calculated in our program by the function `index()`. The inverse function, which

assigns a n-gram to a number is implemented by the function `indexInverse()`. Actually, these functions have two arguments. The first is the number of chars per n-gram, so it is n. The second is the string or number that enters as input to be translated.

We can now understand the program:

An initial Boolean vector `SelfEnglish[]` with dimension equal to the number of possible n-grams is allotted in memory. At the beginning, all its entries are set to true. Next, in the process of learning, the index of each n-gram of the input text is calculated and the corresponding entry in SelfEnglish[] is set to false, just as the immune systems does it. When a new text is given for scrutiny, each n-gram is indexed and the corresponding entry in SelfEnglish[] is checked out: if it is false, the n-gram is self otherwise it is alien. Proportions of alien n-grams are calculated and compared, as it was previously explained.

25 ***Exercise.*** *In the previous program turn the Boolean option to make a test (approximately in line 372,* `boolean Test = true`*) and run various tests by changing the alphabet. By using numerical alphabets for different bases, one can test that the* `index()` *and* `indexInverse()` *functions are correctly implemented.*

Why do we need tests? The reason is that no ones has enough intelligence to preview the output of an arbitrary code, so one must actually run it to see what it encodes for. In more operational words: in general, the cheapest way to fairly know what a program encodes for is to run it and see what it produces.

In practice, even the most simple tasks demand many trials before a correct code can be composed.

26 ***Exercise.*** *Run some elementary tests to check intuitive expectations as the next ones. Try to explain why they fail or why they are verified:*

a) If the length of the English training text is augmented, the proportion of false positives (English n-grams that are classified as aliens) produced by a second text in English diminishes.

b) If the length of the second English text is augmented, the proportion of its false positives augments.

c) If the length of the German text is augmented, the proportion of aliens augments.

d) Saturation is expected: after an initial period of learning, no surprises are expected.

The behavior of the program when its parameters are varied seems interesting enough to awake a better comprehension that would allow to single out the more efficient discriminating rule -if only that makes sense.

27 ***Code to study the dependency of outputs on input parameters.***

```
/*Program I27 EnglishGraphics

 The purpose of this program is to discriminate
 whether a given text is written or not in English.
 We mimic the function of the immune system,
 whose work is based in n-grams, that are
 string n-symbols long.
 Output is graphical.

 The program has four steps:

 1. Definition of self:
 Up to some n, all possible n-grams are generated
 and recorded in the boolean vector
 SelfEng. This is done trickily: we generate a
 boolean vector SelfEng and at the beginning,
 every entry of SelfEng is set to TRUE.
 And this is all to it.
 The idea behind this is that each n-gram
 is given an index, a number that
 identifies it in the boolean vector SelfEng.
 So, recording of Self is done as follows:
 the index of each n-gram of the trainingText is
 calculated and the corresponding entry in SelfEng is
 set to false.
 This mimics the training stage of the immune system.

 2.Test over an English text: Because of the finiteness
 of the learning text,
```

```
 even another text in English might generate alien
 n-grams. So, a second text in English is tested against
 the former one. This defines a calibration stage:
 the proportion of alien n-grams is reported.


 3. Test over a German text:
 A third text, in German, is analyzed:
 its n-grams are compared against those in the list
 that defines the self.  The proportion of
 alien n-grams is calculated. When text are long,
 this proportion
 approximately doubles
 that of the second text in English.

 4.Formulation of the discriminating criterion:
 a classifying threshold is defined as the arithmetic
 average of the two former proportions. If the proportion
 of alien n-grams of a new text is less than
 that of the threshold, the text is in English, otherwise
 it is in another language.

 Various graphics are displayed for diverse values
 of the number of chars taken from the training text.
 Each  graphic represents
 the proportion  of alien n-grams that one finds in
 a testing text as a function
 of the length in chars of that testing text.
 Two testing texts are contrasted: one in English
 the other in German.

*/


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
```

```
public class EnglishGraphics extends JFrame
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static EnglishGraphics p = new EnglishGraphics();

//The painting class is declared and instantiated
private static Canvas c = p.new Canvas();

 //Number of letters in the n-grams
 private static int n;
 //Number of n-grams
 private static int N;
 //DataBase: if n=4,  the dimension must be increased
 private static Boolean SelfEnglish[] = new Boolean[45000000];
 //Training  text. A StringBuffer is a heavyduty String.
 private static StringBuffer trainingText ;
 //Test  texts
 private static StringBuffer testerEng, testerGer;
 //English + German chars (not all)
 private static String Alphabet =
                  "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß: ,.-()0123456789";
 private static  int lengthAlf;
 private static int nTrainingText, nTesterEngText,
                  nTesterGerText, nTrainingSample,
                  nTesterEngSample, nTesterGerSample;
 private static double propSelf, propTest;
 private static Boolean printAll;
 private static int amplificationY;
 private static double scale;
 private static int graphNumber;


 private static int Points[][] = new int[100][3];
```

```
private static int xmax, ymax;
private static int deltaX, deltaY, nPoints;
private static int deltaLearningText, deltaTestingText;
private static boolean undone = true;


//Main method
public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //letters per n-gram
 // if n>3,  the size of SelfEnglish[] must be increased
 n=4;
 //The size of learning text is incremented by this
 deltaLearningText = 30;
 //The size of testing text is incremented by this
 deltaTestingText = 50;
 //Graphical scale = 1 for normal size.
 scale = 0.15;
 //To print all, change to true
 printAll = false;
 //length of the Alphabet
 lengthAlf = Alphabet.length();
 System.out.println( "\nLetters per n-gram = " + n );
 System.out.println( "Lenght of the alphabet = " +
                                    lengthAlf);
 //Length of Self
 N = (int) Math.pow(lengthAlf, n); ;
 System.out.println( "Number of n-grams in vector " +
                                "SelfEng = " + N);
 //The coordinates of points for graphics are set to zero.
 initialize();
 readTexts();
 //The JFrame is constructed with a title
 JFrame f = new JFrame( "Language identifier" );
 //specifications of JFrame f
 //Control is passed to the painting class c (Canvas)
 f.add( c, BorderLayout.CENTER );
 f.add( new JLabel("English (RED) vs. German (BLUE)"),
                             BorderLayout.SOUTH );
```

```
 f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
 f.setSize( 950, 250 );
 f.setVisible( true );
}


//Reading training text in English
private static void readTrainingText()
{
 trainingText =  new StringBuffer("Common words such " +
  "as determiners, conjunctions and prepositions" +
  " seem good clues for guessing a language " +
  "(Johnson 1993). From a corpus of documents" +
  " in a certain language, the most frequent " +
  "words are determined and placed in a model for " +
  "this language. Each word has a significance " +
  "score based on the frequency of that particular " +
  "word in the corpus. By dividing this frequency" +
  " by the total frequency of all words, we give " +
  "each word a probability, and we can see the " +
  "language model as a probability distribution. " +
  "Table 1 shows the most frequent common words " +
  "for 10 European languages, as obtained in " +
  "experiments described by Grefenstette (1995). " +
  "The disadvantage of this technique is that, " +
  "though common words occur enough in larger " +
  "texts, they might not occur in a shorter " +
  "input text.");
}

//Reading calibration text: a second  text in English.
private static void readCalText()
{
 testerEng =  new StringBuffer("Rank order statistics: " +
  "This technique, as described by Cavnar and " +
  "Trenkle (1994), determines how far out of " +
  "place an N-gram is in the language models " +
  "from its place in a document model. Since " +
  "this technique operates on the relative " +
  "indices of features (words or N-grams), these " +
```

```
  "need to be sorted in order of frequency. " +
  "How this technique is applied to N-gram features " +
  "is shown in figure 2. For each N-gram in a " +
  "document model, its counterpart in a language " +
  "model is located, and then calculated how far " +
  "out of place it is. If an N-gram is not in " +
  "the language model, it takes a maximum " +
  "out-of-place value, which is equal to the " +
  "amount of N-grams in the model. The sum of" +
  " all of the out-of-place values for all " +
  "N-grams is the distance measure for the  " +
  "document from the language.The language model" +
  " with the smallest distance from the document " +
  "represents the language of the document.");
}

//Reading test text in German
private static void readTestingText()
{
  testerGer =  new StringBuffer("Sprachidentifizierung " +
  "für Handys. Die Entwickler des japanischen " +
  "Mobilfunkkonzerns KDDI haben ein System " +
  "vorgestellt, mit dem sich der Nutzer " +
  "per Stimmanalyse gegenüber seinem Handy " +
  "identifizieren kann. Die Funktion soll als " +
  "Alternative zum Passwort für Bankgeschäfte" +
  " oder andere Dienste zum Einsatz " +
  "kommen, teilte das Unternehmen heute in Tokio" +
  " mit. Der Prototyp nahm das Stimmmuster " +
  "auf und schickte dieses digitalisiert an " +
  "einen Server, wo die Analyse vorgenommen " +
  "wird. Anhand früherer Spracheingaben wird " +
  "die Identität festgestellt, hieß es. Der " +
  "Prozess nimmt etwa zehn  Sekunden in Anspruch.");
}


private static void readTexts()
{
 readTrainingText();
```

```
  nTrainingText = trainingText.length();
  if (undone )
    System.out.println( "Lenght of trainingText = " +
                                nTrainingText);
  readCalText();
  readTestingText();
  nTesterEngText= testerEng.length();
  if (undone )
   System.out.println( "Lenght of testerEng = " +
                                nTesterEngText);
  nTesterGerText = testerGer.length();
  if (undone )
   System.out.println( "Lenght of testerGer = " +
                                nTesterGerText);
}


 //Mandatory initialization
 private static void initialize()
 {
  for(int i = 0; i < 100; i++)
for(int j = 0; j < 2; j++)
  Points[i][j] = 0;
 }

 //SelfEng contains all possible n-grams.
 private static void initializeSelf(long N)
 {
  for(int i = 0; i< N; i++)
   SelfEnglish[i] = true;
 }

 //Returns the index   of the n-gram 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.
 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;
 correct = false;
}
  }
 if (printAll)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}


//The boolean vector Self records the identity
//of the training text.
//At the beginning, every entry of Self is set to TRUE.
//Each n-gram is given an index, a number that
//identifies it in the boolean vector Self.
//The index of each n-gram of the trainingText is
//calculated and the corresponding entry in Self is
//set to false.
//The variable nTrainingSample decides
//how many chars of trainingText are taken.
private static void DefinitionOfSelf(int n,
                             int nTrainingSample,
                        StringBuffer trainingText)
{
 initializeSelf(N);
 nTrainingText = trainingText.length();
 int m = Math.min(nTrainingText, nTrainingSample);
 int start = 0;
 int ind = 0;
```

```
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  trainingText.substring(start,
                                start+n);
ind = index(n,ngram);
SelfEnglish[ind] = false;
start = start +1;
  }

}

//
//The n-grams of testText are compared against
//those of the trainingText, whose structure was  kept at
//SelfEnglish. The proportion of non-self n-grams
//is reported. nSample chars are taken from testText.
private static double runTest(StringBuffer testText,
                                int nSample)
{
 if (printAll)
 System.out.println( "Alien n-grams");
 int counter = 0;
 int l1 = testText.length();
 int m = Math.min(l1, nSample);
 int start = 0;
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  testText.substring(start,
                                start+n);
ind = index(n,ngram);
if ( SelfEnglish[ind] == true)
{
 counter++ ;
 if (printAll)
 {
  System.out.print(" alien");
 }
}
if (printAll)  System.out.println();
```

```
start = start +1;
  }

 double prop = counter;
 prop = prop / (m-n+1);
 return prop;
}

//The training text is studied
private static void learning(int n,
        int nTrainingSample,
        StringBuffer trainingText)
{
 if (undone )
 System.out.println( "\nNumb of chars taken from" +
         " trainngText = " +  nTrainingSample);
 DefinitionOfSelf(n,nTrainingSample, trainingText );
}

//A second text in English is studied
private static double calibration(StringBuffer testText,
        int nSample)
{
 propSelf = runTest(testText, nSample);

 return propSelf;
}

//A text in German is classified as self else not-self
private static double testing(StringBuffer testText,
        int nSample)
{
 propTest = runTest(testText, nSample);
 return propTest;
}

//
private static void work()
{
 learning(n,nTrainingSample, trainingText );
```

```
 if (undone )
  {
   System.out.println("LT = Length of testing text");
   System.out.println("PropEng = proportion of alien " +
                         "n-grams in  testerEng");
   System.out.println("PropGer = proportion of alien " +
                          "n-grams in  testerGer");
   System.out.println("LT  \t PropEng \t        PropGer " +
  "         \t  Difference");
  }
 boolean go = true;
 nPoints = 0;
 for(int h = 0; go; h++)
 {
  //How many chars of testerEng must be taken
  nTesterEngSample = deltaTestingText*h;
  propSelf = calibration(testerEng,nTesterEngSample);
  //How many chars of testerGer must be taken
  nTesterGerSample = deltaTestingText*h;
  propTest = testing(testerGer, nTesterGerSample);
  double difference = propTest - propSelf;
  if (undone )
   System.out.println( nTesterEngSample + "\t"
  + propSelf+ "\t"+ propTest+ "\t" + difference);
  Points[h][0] = nTesterEngSample ;
  amplificationY = 100;
  Points[h][1] =
  (int) Math.round(propSelf*amplificationY);
  Points[h][2] =
  (int) Math.round(propTest*amplificationY);
  if ( (nTesterEngSample > testerEng.length()) ||
  (nTesterGerSample > testerGer.length() ))
  go = false;
  nPoints = nPoints +1;
 }
}
```

```
 //This is the painting class
 public class Canvas extends JPanel
 {

private static final long serialVersionUID = 1L;

    //Constructor
    public Canvas()
    {
     //initializes the frame;
     repaint();
    }



    //Data of a graphic is normalized to be placed
    //in a given rectangle
    private void rescale( int[][] PointsD)
    {
     //scale factors are defined
     double  fx;
     double  fy;
     xmax = Math.min(nTesterEngSample,nTesterGerSample);
     ymax = amplificationY;
     double rxmax = xmax;
     double rymax = ymax;
     fx = 150 * scale / rxmax;
     fy = -170 * scale / rymax;
     int tx = 0;
     int ty = (int) (scale * 170);
     //Scaling formulae
     //System.out.println("Coordinates in the graphic");
     for(int i = 1; i < nPoints; i++)
     {
      Points[i][0] =  (int) (PointsD[i][0]*fx +tx);
      Points[i][1] =  (int)  (PointsD[i][1]*fy +ty);
      Points[i][2] =  (int)  (PointsD[i][2]*fy +ty);

      /*System.out.println( Points[i][0] + "\t"
       + Points[i][1]+ "\t"+ Points[i][2]);*/
```

```
 }
}


//The graphic is drawn
private void draw(Graphics g )
{
 //Axes
 g.setColor(Color.BLACK);
 g.drawLine(deltaX, 0, deltaX, (int) (170*scale));
 g.drawLine(deltaX, (int) (170*scale),
     deltaX + (int ) (150*scale),
                (int) (170*scale));
 g.setColor(Color.RED);
 //Prop of alien n-grams in English Text
 for(int count = 1; count < nPoints-1; count++)
  g.drawLine(Points[count][0]+deltaX,
      Points[count][1]+deltaY,
         Points[count+1][0]+deltaX,
         Points[count+1][1]+deltaY);
 g.setColor(Color.BLUE);
 //Prop of alien n-grams in German Text
 for(int count = 1; count < nPoints-1; count++)
   g.drawLine(Points[count][0]+deltaX,
      Points[count][2]+deltaY,
      Points[count+1][0]+deltaX,
      Points[count+1][2]+deltaY);
 //Graphic number
 g.drawString(""+ graphNumber,   10 + deltaX,
                (int) (scale * 186 + 10));
}

//A footnote is drawn
private void footnote(Graphics g )
{
 g.drawString("Length of Training text =  " +
       deltaLearningText + "*graphNumber",
        10,   180 + 10);
 g.drawString("Proportion of alien"+ n + "-grams as " +
    " a function of the length of tested material. " +
```

```
      "Vertical scale: from 0 to 1." +
       "Horizontal scale: from 0 to " + xmax, 10, 204);
    }


    //Redraws graphics when either c or repaint() is called
    public void paintComponent( Graphics g )
    {

     deltaX = 0;
    deltaY = 0;
    //go is the condition for halting the for-loop:
    //the studied text must not be larger than
    //the complete training text
    boolean go = true;
    for(int i = 1; go; i++)
     {
      if (undone )
    System.out.println( "\n********GRAPHIC NUMBER " + i
                + "********");
      //Number of graphic
      graphNumber = i;
      //Length of studied testing texts
      nTrainingSample = deltaLearningText*graphNumber;
      //Computations
      work();
      //Adjusting to scale
      rescale(Points);
      //Draw a  graphic
       draw(g);
       //Shifts horizontally
      deltaX = deltaX + (int) (scale * 200);
      //If training test is coped with, halt
      if (nTrainingSample > nTrainingText)  go = false;
      }
    footnote(g);
   undone = false;
    }
}// end of class Canvas
}//end of main class
```

28 ***Exercise. Devise an experimental program to confirm else refute the next generalizations that were made by the Author.***

- For each size of the training text, the difference between the proportion of alien n-grams in German and alien n-grams in the tester test in English tends to stabilize when the size of testing texts grows. We define the discriminating power as the asymptotic value of this difference. It is a growing function of both the length of the training text and that of testing texts.
- The advised length of testing texts to discriminate between English and non-English could be:
    1. One-grams: 150 chars. Discriminating power: 0.06. Discriminating threshold: 0.09.
    2. Two-grams:150 chars. Discriminating power: 0.06. Discriminating threshold: 0.49.
    3. Three-grams:150 chars. Discriminating power: 0.11. Discriminating threshold: 0.85.
    4. Four-grams : 150 chars (lengthy work). Discriminating power: 0.1. Discriminating threshold: 0.93.
    5. The best option is 3-grams: the work is rapidly done and the discriminating power is OK.

29 ***Challenge.*** *Replace the texts in the previous program by others chosen by you. Study the stability of our previous conclusions, both mine and yours.*

30 ***Challenge.*** *The previous program works finely for 1, 2, and 3-grams. Assigned memory needs to be increased 100 times for 4-grams. The program breaks down for 5-grams. Find a way to remedy this.*

English uses a Latin alphabet and German an extension of this. What can we do to study texts in languages with other alphabets?

31 ***Arbitrary alphabets.*** *Java allows you to work with many diverse alphabets. To see this, play with the next code that lists the symbols allowed in Java.*

```
//Program E31 JavaSymbols
//Displays the symbols in Java
public class JavaSymbols {
```

```
 public static void main(String[] args)
  {
int inf = 00;
int sup = 1000;
for(int i= inf; i< sup; i++)
 {
  char ch =   ( char ) ( i  );
  System.out.println( i + " " + ch);
 }
  }
}
```

32 ***Preparing for the study of texts of arbitrary size***

We have been working with short texts but nevertheless we got encouraging results that corroborate our prediction: the strategy used by the immune system is good to discriminate self from not-self in languages of whatever nature. Our problem now is to face the challenge of converting our encouraging results into a robust scientific generalization. To accomplish that aim, we need, as a minimum, to confront our theory with texts of arbitrary size. This will allow us to take diverse samples in order to verify else reject our theory.

To begin with, we need a source of long texts because we need three books, two in English and the other in German. Choose books of your predilection, else, give a look at

`http://www.authorama.com/`
`Cited 20/V/2013`

There you must choose two books in English and the other in German. The Divine comedy is in German (Dante Alighieri: Die Göttliche Komödie). Copy your books from your web browser to a word processor. Next, save them in format `txt` and register the path to your files. In windows, a path looks something like: `C:/ImmuneDistances/EnglishBooks/book1.txt`, in Linux, a path looks like `/home/jose/AJose/Ciencia/DarwinCompleteText.txt`. In Linux, the path can be found if in the file manager, one right clicks on the corresponding file to next choose *properties*.

The books chosen by the Author with the paths to the corresponding files in his hard drive were:

1. Israel Abrahams: The Book of Delight (389KiB). Path:

   ”/home/jose/jose/AAjoser/Ciencia/ImmuneDistances/BookDelight.txt”;

2. Samuel Butler: Essays on Life, Art, and Science (360KiB). Path:
   "/home/jose/jose/AAjoser/Ciencia/ImmuneDistances/SamuelButler.txt";
3. Dante Alighieri: Die Göttliche Komödie (623KiB). Path:
   "/home/jose/jose/AAjoser/Ciencia/ImmuneDistances/DieGKomoedie.txt";

Two books are in English with the intention of studying if our technique allows to discriminate different authors in the same language.

33 ***Code for the study of texts of arbitrary size***

The next code reads from the hard disk files containing complete books and carries an n-gram analysis. To run the next program, you must update in the corresponding reading methods the path to the files with your books.

```
/*Program I33 LargeTexts

 The purpose of this program is to discriminate
 whether a given text is written or not in English.
 The program accepts texts of arbitrary size,
 such as a complete book.
 The books must be copied by the User to the hard disk
 and paths to corresponding files
 must be updated in line 225.

 We mimic the function of the immune system,
 whose work is based in n-grams, that are
 string n-symbols long.
 Output is graphical.

 The program has four steps:

 1. Definition of self:
 Up to some n, all possible n-grams are generated
 and recorded in the boolean vector
 SelfEng. This is done trickily: we generate a
 boolean vector SelfEng and at the beginning,
 every entry of SelfEng is set to TRUE.
 And this is all to it.
```

The idea behind this is that each n-gram
is given an index, a number that
identifies it in the boolean vector SelfEng.
So, recording of Self is done as follows:
the index of each n-gram of the trainingText is
calculated and the corresponding entry in SelfEng is
set to false.
This mimics the training stage of the immune system.

2.Test over an English text: Because of the finiteness
of the learning text,
even another text in English might generate alien
n-grams. So, a second text in English is tested against
the former one. This defines a calibration stage:
the proportion of alien n-grams is reported.

3. Test over a German text:
A third text, in German, is analyzed:
its n-grams are compared against those in the list
that defines the self. The proportion of
alien n-grams is calculated. When text are long,
this proportion
approximately doubles
that of the second text in English.

4.Formulation of the discriminating criterion:
a classifying threshold is defined as the arithmetic
average of the two former proportions. If the proportion
of alien n-grams of a new text is less than
that of the threshold, the text is in English, otherwise
it is in another language.

Various graphics are displayed for diverse values
of the number of chars taken from the training text.
Each graphic represents
the proportion of alien n-grams that one finds in
a testing text as a function
of the length in chars of that testing text.
Two testing texts are contrasted: one in English

```
 the other in German.

*/


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;


import java.lang.IllegalStateException;
import java.util.Scanner;

public class LargeTexts extends JFrame
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static LargeTexts p = new LargeTexts();
 //The painting class is declared and instantiated
 private static Canvas c = p.new Canvas();
 //Number of letters in the n-grams
 private static int n;
 //Number of n-grams
 private static int N;
 //DataBase: if n=4, assigned memory must
 //be increased 100 times
 private static Boolean SelfEnglish[] =
                           new Boolean[450000];
 //Training  text. A StringBuffer is a heavy duty String.
 private static StringBuffer trainingText ;
 //Test  text
```

```
 private static StringBuffer testerEng, testerGer;
 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
 private static  int lengthAlf;
 private static int nTrainingText, nTesterEngText,
                   nTesterGerText, nTrainingSample,
                   nTesterEngSample, nTesterGerSample;
 private static double propSelf, propTest;
 private static Boolean printAll, printTables;
 private static int amplificationY;
 private static double scale;
 private static int graphNumber;
 private static int Points[][] = new int[100][3];
 private static int xmax, ymax;
 private static int deltaX, deltaY, nPoints;
 private static int deltaLearningText, deltaTestingText;
 //Machine to read from the hard disk
 private static  Scanner input;


 //*****************MAIN*****************
 public static void main( String args[] )
 {
  System.out.println("Please, wait for a while");
  //letters per n-gram
  n=3;
  //The size of learning text is incremented by this
  deltaLearningText = 20000;
  //The size of testing text is incremented by this
  deltaTestingText =  20000;
  //Graphical scale = 1 for normal size.
  scale = 0.15;
  //To print all, change to true
  printAll = false;
  //To print numeric results, change to true
  printTables = false;
//length of the Alphabet
  lengthAlf = Alphabet.length();
```

```
  System.out.println( "\nLetters per n-gram = " + n );
  System.out.println( "Lenght of the alphabet = " +
                                    lengthAlf);
  //Length of Self
  N = (int) Math.pow(lengthAlf, n); ;
  System.out.println( "Number of n-grams in vector " +
                               "SelfEng = " + N);
  //The coordinates of points for graphics are set to zero.
  initialize();
  readTexts();
  //The JFrame is constructed with a title
  JFrame f = new JFrame( "Language identifier" );
  //Control is passed to the painting class c (Canvas)
  f.add( c, BorderLayout.CENTER );
  f.add( new JLabel("English (RED) vs. German (BLUE)"),
                            BorderLayout.SOUTH );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  f.setSize( 1050, 250 );
  f.setVisible( true );
 }


 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }

 // File is read
 public  static  StringBuffer readData()
 {
```

```
 StringBuffer text =  new StringBuffer("");
 //try and catch pairs
   try
   {
      while ( input.hasNext() )
      {
       text = text.append(input.next());
       //Insert a space among words
       //text = text.append(" ");
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}
```

```
//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================

//Training text in English
private static void readTrainingText()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
  "ImmuneDistances/BookDelight.txt";
 trainingText =  getText(path);
 String s = trainingText.substring(0, 100);
 System.out.println(s);
}

//Reading calibration text: a second  text in English.
private static void readCalText()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
             "ImmuneDistances/SamuelButler.txt";
 testerEng = getText(path);
}

//Reading test text in German
private static void readTestingText()
{
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
               "ImmuneDistances/DieGKomoedie.txt";
  testerGer = getText(path);
}
//================================================



//Texts are read
private static void readTexts()
{
 readTrainingText();
```

```
  nTrainingText = trainingText.length();
  System.out.println( "Lenght of trainingText = " +
                                nTrainingText);
  readCalText();
  readTestingText();
  nTesterEngText= testerEng.length();
  System.out.println( "Lenght of testerEng = " +
                                nTesterEngText);
  nTesterGerText = testerGer.length();
  System.out.println( "Lenght of testerGer = " +
                                nTesterGerText);
}


 //Mandatory initialization
 private static void initialize()
 {
  for(int i = 0; i < 100; i++)
for(int j = 0; j < 2; j++)
  Points[i][j] = 0;
 }

 //SelfEng contains all possible n-grams.
 private static void initializeSelf(long N)
 {
  for(int i = 0; i< N; i++)
   SelfEnglish[i] = true;
 }

 //Returns the index   of the n-gram 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.
 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;
}


//The boolean vector Self records the identity
//of the training text.
//At the beginning, every entry of Self is set to TRUE.
//Each n-gram is given an index, a number that
//identifies it in the boolean vector Self.
//The index of each n-gram of the trainingText is
//calculated and the corresponding entry in Self is
//set to false.
//The variable nTrainingSample decides
//how many chars of trainingText are taken.
private static void DefinitionOfSelf(int n,
                              int nTrainingSample,
                        StringBuffer trainingText)
{
 initializeSelf(N);
 nTrainingText = trainingText.length();
 int m = Math.min(nTrainingText, nTrainingSample);
 int start = 0;
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
```

```
String ngram =  trainingText.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.
if (ind >= 0)
SelfEnglish[ind] = false;
start = start +1;
  }

}

//
//The n-grams of testText are compared against
//those of the trainingText, whose structure was  kept at
//SelfEnglish. The proportion of non-self n-grams
//is reported. nSample chars are taken from testText.
private static double runTest(StringBuffer testText,
                                int nSample)
{
 if (printAll)
 System.out.println( "Alien n-grams");
 int counter = 0;
 int l1 = testText.length();
 int m = Math.min(l1, nSample);
 int start = 0;
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  testText.substring(start,
                                start+n);
ind = index(n,ngram);
if (ind >0)
{
 if (   SelfEnglish[ind] == true)
{
 counter++ ;
 if (printAll)
 {
```

```
  System.out.print(" alien");
 }
}
}
if (printAll)  System.out.println();
start = start +1;
  }

 double prop = counter;
 prop = prop / (m-n+1);
 return prop;
}

//The training text is studied
private static void learning(int n,
        int nTrainingSample,
        StringBuffer trainingText)
{

 System.out.println( "\nNumb of chars taken from" +
         " trainngText = " +  nTrainingSample);
 DefinitionOfSelf(n,nTrainingSample, trainingText );
}

//A second text in English is studied
private static double calibration(StringBuffer testText,
        int nSample)
{
 propSelf = runTest(testText, nSample);

 return propSelf;
}

//A text in German is classified as self else not-self
private static double testing(StringBuffer testText,
        int nSample)
{
 propTest = runTest(testText, nSample);
 return propTest;
}
```

```
//Main operational work
private static void work()
{
 learning(n,nTrainingSample, trainingText );
 if (printTables)
 {
  System.out.println("LT = Length of testing text");
  System.out.println("PropEng = proportion of alien " +
                          "n-grams in  testerEng");
  System.out.println("PropGer = proportion of alien " +
                           "n-grams in  testerGer");
  System.out.println("LT  \t PropEng \t        PropGer");
 }
 boolean go = true;
 nPoints = 0;
 for(int h = 0; go; h++)
 {
  //How many chars of testerEng must be taken
  nTesterEngSample = deltaTestingText*h;
  propSelf = calibration(testerEng,nTesterEngSample);
  //How many chars of testerGer must be taken
  nTesterGerSample = deltaTestingText*h;
  propTest = testing(testerGer, nTesterGerSample);
  double difference = propTest - propSelf;
  if (printTables)
   System.out.println( nTesterEngSample + "\t"
  + propSelf+ "\t"+ propTest+ "\t" + difference);
  Points[h][0] = nTesterEngSample ;
  amplificationY = 100;
  Points[h][1] =
  (int) Math.round(propSelf*amplificationY);
  Points[h][2] =
  (int) Math.round(propTest*amplificationY);
  if ( (nTesterEngSample > testerEng.length()) ||
  (nTesterGerSample > testerGer.length() ))
  go = false;
  nPoints = nPoints +1;
 }
}
```

```
 //This is the painting class
 public class Canvas extends JPanel
 {

private static final long serialVersionUID = 1L;

    //Constructor
    public Canvas()
    {
     //initializes the frame;
     repaint();
    }



    //Data of a graphic is normalized to be placed
    //in a given rectangle
    private void rescale( int[][] PointsD)
    {
     //scale factor are defined
     double  fx;
     double  fy;
     xmax = Math.min(nTesterEngSample,nTesterGerSample);
     ymax = amplificationY;
     double rxmax = xmax;
     double rymax = ymax;
     fx = 150 * scale / rxmax;
     fy = -170 * scale / rymax;
     int tx = 0;
     int ty = (int) (scale * 170);
     //Scaling formulae
     //System.out.println("Coordinates in the graphic");
     for(int i = 1; i < nPoints; i++)
     {
      Points[i][0] =  (int) (PointsD[i][0]*fx +tx);
      Points[i][1] =  (int)  (PointsD[i][1]*fy +ty);
```

```
  Points[i][2] =  (int)  (PointsD[i][2]*fy +ty);

  /*System.out.println( Points[i][0] + "\t"
   + Points[i][1]+ "\t"+ Points[i][2]);*/
 }
}


//The graphic is drawn
private void draw(Graphics g )
{
 //Axes
 g.setColor(Color.BLACK);
 g.drawLine(deltaX, 0, deltaX, (int) (170*scale));
 g.drawLine(deltaX, (int) (170*scale),
     deltaX + (int ) (150*scale),
                (int) (170*scale));
 g.setColor(Color.RED);
 //Prop of alien n-grams in English Text
 for(int count = 1; count < nPoints-1; count++)
  g.drawLine(Points[count][0]+deltaX,
      Points[count][1]+deltaY,
         Points[count+1][0]+deltaX,
         Points[count+1][1]+deltaY);
 g.setColor(Color.BLUE);
 //Prop of alien n-grams in German Text
 for(int count = 1; count < nPoints-1; count++)
   g.drawLine(Points[count][0]+deltaX,
      Points[count][2]+deltaY,
      Points[count+1][0]+deltaX,
      Points[count+1][2]+deltaY);
 //Graphic number
 g.drawString(""+ graphNumber,   10 + deltaX,
                (int) (scale * 186 + 10));
}

//A footnote is drawn
private void footnote(Graphics g )
{
 g.drawString("Length of Training text =  " +
```

```
        deltaLearningText + "*graphNumber",
         10,   180 + 10);
 g.drawString("Proportion of alien"+ n + "-grams as " +
    " a function of the length of tested material. " +
  "Vertical scale: from 0 to 1." +
   "Horizontal scale: from 0 to " + xmax, 10, 204);
}



//Redraws graphics when either c or repaint() is called
public void paintComponent( Graphics g )
{
deltaX = 0;
deltaY = 0;
//go is the condition for halting the for-loop:
//the studied sub-text must not be larger than
//the complete training text
boolean go = true;
for(int i = 1; go; i++)
 {
  System.out.println( "\n********GRAPHIC NUMBER " + i
            + "********");
  //Number of graphic
  graphNumber = i;
  //Length of studied testing texts
  nTrainingSample = deltaLearningText*graphNumber;
  //Computations
  work();
  //Adjusting to scale
  rescale(Points);
  //Draw a  graphic
   draw(g);
   //Shifts horizontally
  deltaX = deltaX + (int) (scale * 200);
  //If training test is coped with, halt
  if (nTrainingSample > nTrainingText) go = false;
  }
 footnote(g);
 }
```

```
 }// end of class Canvas
}//end of main class
```

34 ***Exercise.*** *Run the program for 1, 2, 3 and 4-grams. Play with the code to try to understand how it functions.*

35 ***Exercise.*** *Make an experimental study to agree else disagree with the next conclusion drawn by the Author: It is elementary to discriminate between English and German if one uses 2, 3, or 4-grams. The discriminating power grows with n and for n=4 it is immense. Nevertheless, the best option seems to be 3-grams: discrimination is clear and not to expensive in time of calculation.*

36 ***Exercise.*** *One intuitively knows that every writer has his or her own style. Make suitable modifications to the previous code to verify else reject this intuitive feeling. Propose a quantifying definition of style.*

37 ***Challenges***

1. The previous program repeats calculations two or three times. And if one resizes the window, calculations are again repeated. All this is useless. Arrange things to make calculations only once. This is achieved if all calculations are done as previous work before the execution of the painting procedure. The price to be payed is that more memory is needed: maybe that is not realizable because the program uses too much memory. Can you imagine a way to diminish the waste of assigned but non used memory?

2. We have been working with the minimum alphabet. Arrange things to accept full fledged treatment of writing expressiveness in whatever language.

38 ***Waiting time***

In the solution of the previous exercise we claimed that good authors tend to use the whole potentiality of the language and so large texts are useless to distinguish with our techniques between two authors. So, it seems reasonable to think that after an initial period of learning, which must be sufficiently long, nothing new there appears. To test this belief, we composed a code to carry out the following task: Let $F(i)$ be the number of chars that have been scanned until finding the $i-$th brand new n-gram. Let $W(i) = F(i) - F(i-1)$ be the waiting time, i.e., the number of additional chars that one must scan to get the next brand new n-gram. We would like to decide whether or not the dependence of W on i is polynomial else exponential. The code follows:

```
/*Program I38 WaitingTime
 Our project   is to discriminate
 whether a given text is written or not in English.
 We expect that after sufficient training no new things
 shall appear. More rigorously,   the waiting time
 until the next brand new n-gram must be an increasing function
 of the size of the read portion of the teaching text.
 Or, equivalently, let i be the i-th brand new n-gram
 that has been recorded, then the waiting time must be an
 increasing function in i.

 This program allows to study the behavior of
 the waiting time function.

 An n-gram is a sub-string n-symbols long.

 Let T(i) be the number of chars that have been
 scanned until finding the i-th brand new n-gram.
 Let  W(i)= T(i) - T(i-1) be the waiting time, i.e.,
 the number of additional chars that one must
 scan to get the next brand new n-gram.
 We study the behavior of W(i).
 Results are presented in graphic form.

 The program has three steps:

 Step one:
 A text   is given for examination.
 All n-grams are captured and recorded in the vector
 Self, which is a  boolean vector:
 At the beginning, every entry of Self is set to TRUE.
 Each n-gram is given an index, a number that
 identifies it in the boolean vector Self.
 The index of each n-gram of the text is
 calculated and the corresponding entry in Self is
 set to false. This mimics the immune system.

 Step two:
 If an n-gram is found for the first time,
```

```
 its appearance time is recorded in T[i],
 where i is the counter of brand-new n-grams.
 The waiting time is W(i)= T(i) - T(i-1).
 W is very rich in information, very complex.
 So,  we calculate a smoothing version of T
 and report in a graphic only a sample of values.
 The smoothing is done as follows:
 for each i, we average  the next nWindow terms of WT
 an we keep the result in AWT[i].

 Step three:
 The data in AWT[i] is presented in a graphic form,
 but only for those i which are multiples of nWindow.

*/


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.lang.IllegalStateException;
import java.util.Scanner;


public class WaitingTime extends JFrame
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static WaitingTime p = new WaitingTime();

//The painting class is declared and instantiated
private static Canvas c = p.new Canvas();
```

```
//Number of letters in the n-grams
private static int n;
//Number of n-grams
private static int N;
//Record of all n-grams: if n>3,
//the dimension must be increased
private static Boolean Self[] = new Boolean[2100000];
//Texts. A StringBuffer is a heavy-duty String.
private static StringBuffer chosenText,trainingText,
testerEng, testerGer;
//English + German chars, only the most informative

private static String Alphabet =
                  "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
 "aäbcdefghijklmnoöpqrstuüvwxyzß ";
//Short alphabet to test the code.
//Diminish nWindow to 3
//private static String Alphabet =  "abc";
private static  int lengthAlf;
//Lengths of texts
private static int nTrainingText, nTesterEngText,
                  nTesterGerText, nText;
//Name of chosen text
private static String nameText;
//Appearance time of the i-th brand new n-gram
private static int T[] = new int[210000];
//Waiting time
private static int WT[] = new int[210000];
//Averaged  waiting time
//For each i, average is taken over nWindow data.
private static int AWT[] = new int[210000];
//Max values of waiting and smoothed waiting times.
private static int maxWT, maxSWT;
//Width of the smoothing window
private static int nWindow;
//Number of new n-grams
private static int nNew;
//vertical scale
private static double scale;
```

```
//Zero y-coordinate
private static int base;
//Coordinate y of points for the graphic
private static int yPoints[] = new int[200000];
//max value of the y-coordinate
private static int  ymax;
//Number of points in the graphic;
private static int  nPoints;
//Print additional information
//If you want to see what you print to the console,
//you must shorten the alphabet down to some few chars.
private static boolean print = false;
private static boolean printAll = false;
//Machine to read from the hard disk
private static  Scanner input;


//******************MAIN******************
public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //letters per n-gram
 n=3;
 //Width of smoothing window
 nWindow = 100;
 //Graphical scale = 1 for normal size.
 scale = 1;
 //To print additional information, change to true
 print = true;
 printAll = false;
 //length of the Alphabet
 lengthAlf = Alphabet.length();
 System.out.println( "\nLetters per n-gram = " + n );
 System.out.println( "Lenght of the alphabet = " +
                                    lengthAlf);
 //Length of Self
 N = (int) Math.pow(lengthAlf, n); ;
 System.out.println( "Number of n-grams in vector " +
                                "Self = " + N);
 readTexts();
```

```
  //Choose the text to be analyzed
  chooseText();
  //
  //The JFrame is constructed with a title
  JFrame f = new JFrame( "Smooted waiting time" );
  //Control is passed to the painting class c (Canvas)
  f.add( c, BorderLayout.CENTER );
  f.add( new JLabel(nameText), BorderLayout.SOUTH );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  //Dimension of the graphical window
  f.setSize( 2100, 500 );
  f.setVisible( true );
 }

 //Text for analysis is chosen.
 private static void chooseText()
 {
  //chosenText =  trainingText; nameText = "English text";
  chosenText =   testerEng; nameText = "Second Eng Text";
  //chosenText =   testerGer; nameText = "German Text";
  /*chosenText = trainingText.append(testerEng);
  nameText = "English + Second English texts"; */
  /*chosenText = trainingText.append(testerGer);
  nameText = "English + German texts";*/
  System.out.println("Chosen text = " + nameText);
 }


 // File to be read is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
```

```
   }
}

// File is read
public  static  StringBuffer readData()
{
 StringBuffer text =  new StringBuffer("");
 //try and catch pairs
   try
   {
      while ( input.hasNext() )
      {
       text = text.append(input.next());
       //if desired, a space among words
       //can be inserted
       //text = text.append(" ");
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

//Text is read from file
```

```
private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}

//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================

//Training text in English
private static void readTrainingText()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
  "ImmuneDistances/BookDelight.txt";
 trainingText =  getText(path);
 String s = trainingText.substring(0, 100);
 System.out.println(s);
}

//Reading calibration text: a second  text in English.
private static void readCalText()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
             "ImmuneDistances/SamuelButler.txt";
 testerEng = getText(path);
}

//Reading test text in German
private static void readTestingText()
{
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
               "ImmuneDistances/DieGKomoedie.txt";
  testerGer = getText(path);
}
//================================================
```

```
 //All texts are read
 private static void readTexts()
 {
  readTrainingText();
  nTrainingText = trainingText.length();
  System.out.println( "Lenght of trainingText = " +
                                nTrainingText);
  readCalText();
  readTestingText();
  nTesterEngText= testerEng.length();
  System.out.println( "Lenght of testerEng = " +
                                nTesterEngText);
  nTesterGerText = testerGer.length();
  System.out.println( "Lenght of testerGer = " +
                                nTesterGerText);
}


 //Mandatory initialization
 private static void initialize()
 {
  for(int i = 0; i < 2000; i++)
  yPoints[i] = 0;
  for(int i = 0; i< N; i++)
   Self[i] = true;
 }



 //Returns the index   of the n-gram 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.
 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;
}



//The boolean vector Self[] records the identity
//of the text.
//At the beginning, every entry of Self[] is set to TRUE.
//Each n-gram is given an index, a number that
//identifies it in the boolean vector Self[].
//The index of each n-gram of the trainingText is
//calculated and the corresponding entry in Self[] is
//set to false.
//n is the number of chars of the n-gram
private static void examination(int n,
                  StringBuffer text)
{
 nText = text.length();
 int start = 0;
 int ind = 0;
 nNew = 0;
 for(int c=0; c < nText-n+1; 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.
if (ind >= 0)
{
 if (Self[ind] == true)
  {
Self[ind] = false;
//c is the appearance time
//of the n-gram with index ind.
T[nNew] = c;
nNew = nNew + 1;
  }
}
start = start +1;
  }
 System.out.println("Number of new n-grams = " + nNew+
 "\nWidth of smoothing window = " + nWindow);
}

//The waiting time is calculated
//and smoothed:
//the average of the next nWindow terms
//is calculated and kept in SW[]
private static void  smoothing()
{
 int sum = 0;
 maxSWT = 0;
 //Waiting time is calculated
 for(int h=1; h < nNew; h++)
          WT[h] = T[h] - T[h-1];
 //Waiting time is averaged over a window of width nWindow
 for(int i = 0; i < nNew - nWindow; i++)
  {
for(int j = 0; j < nWindow; j ++)
  sum = sum + WT[i +j];
double sumd =  sum;
```

```
AWT[i] = (int) sumd / nWindow;
sum = 0;
if (AWT[i] > maxSWT) maxSWT = AWT[i];
  }
}


//Operational part
private static void work()
{
 initialize();
 examination(n, chosenText);
 smoothing();
 if (print)
 {
  //console output is prepared
  System.out.println("i = i-th brand new n-gram " +
                          "n-grams in  testerEng");
  System.out.println("T(i) = number of char read until " +
             "the i-th brand new n-gram was found");
  System.out.println("WT(i) = waiting time = " +
   "T(i)-T(i-1)");
  System.out.println("AWT(i) = averaged waiting time = " +
   "(T(i)-T(i-1))/nWindow");
  System.out.println("i  \t T(i) \t WT(i) \t AWT(i)");
 }
 //Graphical output is prepared
 maxWT = 0;
 for(int h = 0; h < nNew; h++)
 {
  if (WT[h] > maxWT)  maxWT = WT[h];
  if (print)
   System.out.println( h + "\t " + T[h] + "\t " +
                   WT[h] + "\t " + AWT[h] );
  }


 //A sample of values is chosen for a drawing
 System.out.println( "Waiting time at a sample of points");
 double nNewR = nNew;
```

```
  nPoints = (int) nNewR / nWindow;
  for(int i = 0; i < nPoints; i++)
   {
yPoints[i] = (int) Math.round(AWT[nWindow*i]);
System.out.println(i + " " + yPoints[i]);
   }
 }

 //************PAINTING CLASS*****************

 //This is the inner painting class
 public class Canvas extends JPanel
 {

private static final long serialVersionUID = 1L;

    //Constructor
    public Canvas()
    {
     //initializes the frame;
     repaint();
    }

    //Data  is normalized to be placed
    //in a given rectangle
    private void rescale()
    {
     //scale factors are defined
     ymax = 0;
     base = 420;
     for(int i = 0; i < nPoints; i++)
     if ( yPoints[i] > ymax) ymax = yPoints[i];
     double rymax = ymax;
     double fy =  scale * base / rymax;
     for(int i = 1; i < nPoints; i++)
      yPoints[i] =   (int) (base -  yPoints[i]*fy);
    }


    //The graphic is drawn
```

```
    private void draw(Graphics g )
    {
     g.setColor(Color.RED);
     //Prop of alien n-grams in English Text
     for(int i= 1; i < nPoints-1; i++)
     { g.drawLine(i,
          base,
             i,
            yPoints[i]);
     }
    }


    //A footnote is drawn
    private void footnote(Graphics g )
    {
     g.drawString("Length of text =  " + nText +
      ". Number of needles = "+ nPoints +
      ". Number of new n-grams = " + nNew +
      ". Width of smoothing window = " + nWindow+
      ". Max waiting time (not shown) = " + maxWT +
      ". Max smoothed waiting time = " + maxSWT,
            10,   453);
    }


    //Redraws graphics when either c or repaint() is called
    public void paintComponent( Graphics g )
    {
     work();
    //Adjusting to scale
    rescale();
    //Draw a  graphic
     draw(g);
    footnote(g);
   }
 }// end of class Canvas
}//end of main class I38 WaitingTime
```

39 ***Exercise.*** *Run the previous program and play with the code to understand it.*

For instance, change the value of n among 1, 2, or 3. In case of memory overflows, increase the dimensions of arrays, or use a truncated alphabet.

40 ***Exercise.*** *Develop your own research to agree else disagree with the next conclusions drawn by the Author in regard with the output of the previous program:*

1. The graphic of waiting times somehow resembles white noise. This was unexpected by the Author, who thought of a softly increasing function.

2. Renown authors seem to obey the next rule: say nothing if it does not contain something new. But to incarnate new ideas in verbal strings, one seeks economy: the cheapest way to do that is to employ new words with new n-grams. This is already in the making of the language, an inheritance from ancient people.

3. As a consequence, n-grams are not good to define a style of author: authors try to use the potentialities of the language (which is sub-culture dependent), and so all of them try to use the same bank of words an hence of n-grams.

4. Languages have a finite number of words, say, one million. This implies that in spite of a powerful creativeness, one must anyway reuse the very same words. Therefore, waiting times for new n-grams must have an overall increased tendency. This was shown with smoothing techniques: instead of drawing a given function, one reports the average of the function around a certain window. An exponential envelope appears for a width window of 100. To unveil an exponential dependence, one pursues a regression analysis of the logarithm of the dependent variable over the untouched data of the independent one. Using the R-package it was found with an F-test that the null hypothesis of a linear dependence of the natural logarithm of the waiting time on the number of recorded new n-grams can be explained by an average value plus randomness with a probability less than $2e^{-16}$. Thus, we feel free to consider that the dependence of the waiting time on the number of new n-grams is exponential. The correlation coefficient, r was nearly 0.9, so the exponential model is good but not perfect: therefore, apart from a linear form of the exponent, a polynomial perturbation could eventually fit the data much better. To learn how to use the R-package, which is free, professional and expandable with many tools for biology and genetics, see (Rodríguez, 2010).

41 ***Challenges***

1. *Study the fine structure of the waiting time function. You will need to save the output to disk. How to do that is explained in Vol 1 chapter 11, program A158.*

2. *Add to the previous code the possibility to save graphics to disk for further processing. This is explained in Vol 3, program C8.*

42 ***On the origin of the exponential envelope of the waiting time function***.

Our graphical study of the behavior of the smoothed average of waiting time has allowed us to witness an arising of exponential envelope. It seems rather difficult and very intriguing to explain the quantitative origin of this tendency. Nevertheless, a first step in that direction could be to determine the arising envelope for random texts. This study can be pursued by the next code that synthesizes texts by concatenation of chars that are generated at random from a given alphabet (with a uniform distribution). We take as alphabet the same we took in the previous studies. The code follows:

```
/*Program I42 RandomText
 Our final purpose is  to discriminate
 whether a given text is written or not in English.
 We expect that after sufficient training no new things
 shall appear. More rigorously,   the waiting time
 until the next brand new n-gram must be an increasing
 function of the number of recorded new n-grams.

 We have discovered that the waiting time function has
 an exponential envelope.

 Why?

 No idea. nevertheless, our first duty is
 to see how is the waiting time for texts
 that have been concatenated with chars chosen
 at random from a given alphabet.

 To that aim, we compose in this program random texts
 and study the ensuing waiting time function.
```

```
An n-gram is a sub-string n-symbols long.

Let T(i) be the number of chars that have been
scanned until finding the i-th brand new n-gram.
Let  W(i)= T(i) - T(i-1) be the waiting time, i.e.,
the number of additional chars that one must
scan to get the next brand new n-gram.
We study the behavior of W(i).
Results are presented in graphic form.

The program has three steps:

Step one:
A text   is given for examination.
All n-grams are captured and recorded in the vector
Self, which is a  boolean vector:
At the beginning, every entry of Self is set to TRUE.
Each n-gram is given an index, a number that
identifies it in the boolean vector Self.
The index of each n-gram of the text is
calculated and the corresponding entry in Self is
set to false. This mimics the immune system.

Step two:
If an n-gram is found for the first time,
its appearance time is recorded in T[i],
where i is the counter of brand-new n-grams.
The waiting time is W(i)= T(i) - T(i-1).
W is very rich in information, very complex.
So,  we calculate a smoothing version of T
and report in a graphic only a sample of values.
The smoothing is done as follows:
for each i, we average  the next nWindow terms of WT
an we keep the result in AWT[i].

Step three:
The data in AWT[i] is presented in a graphic form,
but only for those i which are multiples of nWindow.
```

```
*/



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;



public class RandomText extends JFrame
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static RandomText p = new RandomText();

//The painting class is declared and instantiated
private static Canvas c = p.new Canvas();

 //Number of letters in the n-grams
 private static int n;
 //Number of n-grams
 private static int N;
 //Record of all n-grams: if n>3,
 //the dimension must be increased
 private static Boolean Self[] = new Boolean[2100000];
 //Texts. A StringBuffer is a heavy-duty String.
 private static StringBuffer text;

 //English + German chars, only the most informative
 private static String Alphabet =
                  "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
```

```
 "aäbcdefghijklmnoöpqrstuüvwxyzß ";
//Short alphabet to test the code.
//Diminish nWindow to 3
//private static String Alphabet =  "abc";
private static  int lengthAlf;
//Lengths of texts
private static int   nText;
//Name of chosen text
private static String nameText;
//Appearance time of the i-th brand new n-gram
private static int T[] = new int[450000];
//Waiting time
private static int WT[] = new int[450000];
//Smoothed, averaged  waiting time
private static int AWT[] = new int[450000];
//Max values of waiting and smoothed waiting times.
private static int maxWT, maxAWT;
//Width of the smoothing window
private static int nWindow;
//Number of new n-grams
private static int nNew;
//vertical scale
private static double scale;
//Zero y-coordinate
private static int base;
//Coordinate y of points for the graphic
private static int yPoints[] = new int[200000];
//max value of the y-coordinate
private static int  ymax;
//Number of points in the graphic;
private static int  nPoints;
//Print additional information.
//If you want to see what you print to the console,
//you must shorten the alphabet down to some few chars
//and define nWindow = 3
private static boolean print;
//Random generator
private static Random r  = new Random();

//******************MAIN******************
```

```
public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //letters per n-gram
 n=3;
 //Width of smoothing window
 nWindow = 100;
 //Graphical scale = 1 for normal size.
 scale = 1;
 //To print additional information, change to true
 print = true;
 //length of the Alphabet
 lengthAlf = Alphabet.length();
 System.out.println( "\nLetters per n-gram = " + n );
 System.out.println( "Lenght of the alphabet = " +
                                    lengthAlf);
 //Length of Self
 N = (int) Math.pow(lengthAlf, n); ;
 System.out.println( "Number of n-grams in vector " +
                               "Self = " + N);
 //A text is composed with chars chosen at random
 nameText = "Random";
 composeText();
 //
 //The JFrame is constructed with a title
 JFrame f = new JFrame( "Smooted waiting time" );
 //Control is passed to the painting class c (Canvas)
 f.add( c, BorderLayout.CENTER );
 f.add( new JLabel(n + "-grams. Text = " + nameText),
                        BorderLayout.SOUTH );
 f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
 //Dimension of the graphical window
 f.setSize( 2100, 500 );
 f.setVisible( true );
}

//A random text is composed
private static void composeText()
{
 text =  new StringBuffer("");
```

```
 for(int i = 0; i < 400000; i++)
 {
  //a random integer number
  int l = r.nextInt(lengthAlf);
  //A random char
  char c = Alphabet.charAt(l);
  //char is appended to text
  text = text.append(c);
 }
 System.out.println("Chosen text = " + nameText);
 String s = text.substring(0,50);
 System.out.println(s);
}
```

```
//Mandatory initialization
private static void initialize()
{
 for(int i = 0; i < 2000; i++)
 yPoints[i] = 0;
 for(int i = 0; i< N; i++)
  Self[i] = true;
}
```

```
//Returns the index   of the n-gram 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.
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;
}
  }
 return ind;
}



/*
 * The boolean vector Self[] records the identity
of the text.
At the beginning, every entry of Self[] is set to TRUE.
Each n-gram is given an index, a number that
identifies it in the boolean vector Self[].
The index of each n-gram of the trainingText is
calculated and the corresponding entry in Self[] is
examined: if it is false, it is set to true
and the number of read chars in recorded in T[].
 *
 */
private static void examination(int n,
                  StringBuffer text)
{
 nText = text.length();
 int start = 0;
 int ind = 0;
 nNew = 0;
 for(int c=0; c < nText-n+1; 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.
if (ind >= 0)
{
 if (Self[ind] == true)
  {
Self[ind] = false;
T[nNew] = c;
nNew = nNew + 1;
  }
}
start = start +1;
  }
 System.out.println("Number of new n-grams = " + nNew+
 "\nWidth of smoothing window = " + nWindow);
}

//The waiting time is calculated
// as W[h] = T[h]-T[h-1]
//and smoothed:
//the average of the next nWindow terms
//is calculated and kept in SW[]
private static void  smoothing()
{
 int sum = 0;
 maxAWT = 0;
 //Waiting time is calculated
 for(int h=1; h < nNew; h++)
          WT[h] = T[h] - T[h-1];
 //Waiting time is averaged over a window of width nWindow
 for(int i = 0; i < nNew - nWindow; i++)
  {
for(int j = 0; j < nWindow; j ++)
  sum = sum + WT[i +j];
double sumd =  sum;
AWT[i] = (int) sumd / nWindow;
sum = 0;
if (AWT[i] > maxAWT) maxAWT = AWT[i];
```

```
   }
 }


 //Operational part
 private static void work()
 {
  initialize();
  examination(n, text);
  smoothing();
  if (print)
  {
   //console output is prepared
   System.out.println("i = i-th brand new n-gram " +
                          " in  testerEng");
   System.out.println("T(i) = number of char read until " +
              "the i-th brand new n-gram was found");
   System.out.println("WT(i) = waiting time = " +
    "T(i)-T(i-1)");
   System.out.println("i  \t T(i) \t        WT(i)");
  }
  //Graphical output is prepared
  maxWT = 0;
  for(int h = 1; h < nNew; h++)
  {
   if (WT[h] > maxWT)  maxWT = WT[h];
   if (print)
    System.out.println( h + "\t " + T[h] + "\t " +
      WT[h] + "\t " + AWT[h] );
   }


  //A sample of values is chosen for a drawing
  System.out.println( "Sampled point \t AWT[]" );
  double nNewR = nNew;
  nPoints = (int) nNewR / nWindow;
  for(int i = 0; i < nPoints; i++)
   {
yPoints[i] = (int) Math.round(AWT[nWindow*i]);
System.out.println(nWindow*i + " \t" + yPoints[i]);
```

```
   }
 }


 //This is the inner painting class
 public class Canvas extends JPanel
 {

private static final long serialVersionUID = 1L;

    //Constructor
    public Canvas()
    {
     //initializes the frame;
     repaint();
    }

    //Data  is normalized to be placed
    //in a given rectangle
    private void rescale()
    {
     //scale factors are defined
     ymax = 0;
     base = 420;
     for(int i = 0; i < nPoints; i++)
     if ( yPoints[i] > ymax) ymax = yPoints[i];
     double rymax = ymax;
     double fy =  scale * base / rymax;
     for(int i = 1; i < nPoints; i++)
      yPoints[i] =   (int) (base -  yPoints[i]*fy);
    }


    //The graphic is drawn
    private void draw(Graphics g )
    {
     g.setColor(Color.RED);
     //Smoothed waited time for the i-th
     //brand new n-gram.
     for(int i= 1; i < nPoints-1; i++)
```

```
     { g.drawLine(i,
          base,
             i,
            yPoints[i]);
     }
    }

    //A footnote is drawn
    private void footnote(Graphics g )
    {
     g.drawString("Length of text =  " + nText +
      ". Number of needles = "+ nPoints +
      ". Number of new n-grams = " + nNew +
      ". Width of smoothing window = " + nWindow+
      ". Max waiting time (not shown) = " + maxWT +
      ". Max smoothed waiting time = " + maxAWT,
            10,   453);
    }

    //Redraws graphics when either c or repaint() is called
    public void paintComponent( Graphics g )
    {
     work();
    //Adjusting to scale
    rescale();
    //Draw a  graphic
     draw(g);
    footnote(g);
   }
 }// end of class Canvas
}//End of main class I42 RandomText
```

43 ***Exercise.*** *Run the previous program and play with the code. Give to n a value among 1, 2, 3 but to see something interesting adjust the width of the smoothing window, nWindow. For instance, for* $n = 3$*, take for nWindow a value among 40 and 500. For* $n = 2$*, nWindow might take values greater than 10 but for* $n = 1$ *less than 10. Memory overflows will appear for values of n higher than 3: you must expand the dimensions of used vectors in the program else truncate the alphabet.*

44 ***Exercise.*** *Verify that the exponential envelope also appears in the study of*

DNA strings that have been synthesized as concatenation of chars taken at random from the DNA alphabet. You must update the alphabet to "ATCG", adjust memory requirements in the dimensions of arrays, and play with parameters, say, n, the number of chars per n-gram, and nWindow the width of the smoothing window, which is the number of data that is averaged to avoid the shock caused by white noise of natural data.

45 ***Exercise.*** *Discuss the next conclusions drawn by the Author in regard with the exponential envelope:*

1. *A text composed with concatenation of chars taken at random also presents an exponential envelope.*
2. *The graphic of the waiting time function of random texts obeys a better fitting than those of human composed texts.*
3. *So, from the stand point of n-grams, creativity and information is a slight perturbation to random assembling.*
4. *This means that automatic methods for creativity might eventually be easy to construct. By the way, evolution is the very first example and possibly the most general one. Nevertheless, let us remember that one can implement evolution as bad as desired, i.e., with an incompetent performance as compared with fully random assembling.*
5. *On the other hand, no writer tries to mimic randomness. So, why do they at last simulate a random process? A possible explanation is that the machinery that is responsible for verbal creativity is both socially and biologically optimized to create at a minimum cost, which is that payed by randomness.*

46 ***Challenge.*** *Make a study of your preferred DNA or protein sequences. Compare results with those gotten for DNA strings synthesized at random.*

1.3 Conclusion

A distance is a mathematical function that is used to measure differences among objects. For text objects, say, DNA, proteins or languages, many diverse distances have been defined. Observing that the immune system can discriminate between self and not self, we deduced that this system defines a discrete distance: the distance between self and self is zero but between self and not-self is one. From this we predicted that a procedure that mimics the operation of the immune system

must be useful to design language identifiers, which are methodologies to establish the identity of a language. At this first stage we just decided whether or not a text is written in English. We tested and verified this prediction with a program in Java for texts in English and German and even to compare diverse authors in the same language. The crucial definition is that of an n-gram, a sub-string n-chars long. We used the holographic technique which works as follows: for a given n, all the n-grams of a training text are recorded. The proportion of alien n-grams in another text in the same language is calculated. The same is done with a text in a foreign language. Thus, we end with two proportions of alien n-grams. We find the semi-sum and this defines a threshold t: if the proportion of alien n-grams of a new testing text is less than t, the text is written in the initial language otherwise in the second. We studied also the behavior of waiting times (the additional number of chars that one must scan to get the next brand-new n-gram) and we found an emergent exponential envelope in the large scale that was made visible by smoothing, i.e. averaging values close to the one under study. This is also observed in texts that have been synthesized as concatenation of random chars. To explain this is a challenge offered to our Community.

Chapter 2

The symmetric distance

`Enhancing common sense`

47 ***Purpose***

With a minor practice a student of German as a second language can correctly decide that a given text is or not written in that language. While the exact mechanism could be person-dependent, a simple theory can be put forward and tested: one pays attention to those distinctive words that are in the language but that do not exist in the others languages one knows. Examples: *ich, und, sie, auf, die, wie, von, hier, ihr, uns*. These shorts German words are frequent and do not appear neither in English, nor in Spanish or French so, in regard with these languages, they are distinctive features of German. Our purpose is to formalize this procedure and to frame it in the general study of n-grams that was sparked by our study of the immune system.

2.1 A distance for sets

Distinctive words are those that are present in one language but not in other(s). Let us make this idea into a **distance**, i.e., a function to measure differences among any pair of objects of a given set.

48 ***The symmetric distance among two sets***

Let us consider two sets $A = \{a,b,c,d\}$ and $B = \{b,c,d,e,f\}$. They are different: a is in A but not in B, besides e and f are in B but not in A. Let us say from now on that the symmetric distance between A and B is 3, because 3 are the elements that make the difference. The definition follows:

The **symmetric distance** between sets A and B is the number of elements that are in A but not in B plus the number of elements that are in B but not in A. Since A and B are valued on equal footing, this distance is really symmetric. Our notation:

Symmetric distance between A and B = $d_\Delta(A,B) = \#(A-B) \ + \ \#(B-A)$

An inefficient form of using the symmetric distance as discriminant tool of languages would be to consider all the words of a given long text. Nevertheless, our developed programs allow us to propose something more interesting:

49 ***The most common n-grams symmetric distance among two texts***

Instead of distinctive words, we can rank the n-grams of a long text according to frequency and take the most common ones up to certain limit to conform the reference set of the language. To calculate the resultant symmetric distance between two texts we proceed as follows:

1. Two texts are given for examination. All n-grams of each text are captured and their absolute frequencies are recorded. This operation is done with the help of an indexation procedure.

2. The most common words, up to certain limit, are picked up and this defines two sets A and B. The elements that make the difference are listed. Example: $A = \{1,2,3,4,5\}$ and $B = \{4,5,6,7,8,9\}$ The set with the elements that make the difference is $\Delta = \{1,2,3,6,7,8,9\}$. This set consists of those elements that are in A but not in B joined to the elements that are in B but not in A.

3. The symmetric distance between the two texts is defined as the number of elements in Δ, 7 in our example.

50 ***The code that uses the most common n-grams of two texts to calculate its symmetric distance follows.***

```
/*Program I50 SymmetricDistance

 We use the symmetric distance between sets
 together with the most common n-grams in a text
 to define a distance among texts. It
 mimics what a human being does when recognizing
```

```
 at once that a text is in English else in German.


 The program has three steps:

 Step one:
 Two texts are given for examination.
 All n-grams of each text are captured
 and their absolute
 frequencies are recorded.
 This operation is done with the help
 of an indexing procedure.

 Step two:
 The most common   words, up to certain limit,
 are picked up
 and this defines two sets A and B.
 The elements that make the difference are listed.
 Example: A = {1,2,3,4,5}  and
 B = {4,5,6,7,8,9}.
 The set with the elements that make the difference
 is Delta = {1,2,3,6,7,8,9}.
 This set consists of  those elements
 that are in A but not in B
 joined to  the  elements that are in
 B but not in A.


 Step three:
 The symmetric distance between
 the two texts is defined as
 the number of elements in Delta, 7 in our example.

*/



import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
```

```
import java.util.Random;
import java.lang.IllegalStateException;
import java.util.Scanner;

public class SymmetricDistance
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static SymmetricDistance p = new SymmetricDistance();

 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n=3;
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);


 //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector
{
int L = (int) Math.pow(lengthAlf+1, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
```

```
  }

//Initialization by cloning from A
intVector(intVector A)
 {
  for(int i = 0; i < L; i++)
   F[i] = A.F[i];
 }
}//end of class intVector

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


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
 private static intVector  freq1, freq2
                                 = p.new intVector();
 private static intVector Common1, Common2
                                 = p.new intVector();

 //Training  text. A StringBuffer is a heavy-duty String.
 private static StringBuffer EngText1, EngText2,
                              GerText,  randText;

 private static  Boolean withBlanks;
 //Number of studied common words
 private static int nCommon;

 private static int nEngText1, nEngText2,
                   nGerText, nText;
 private static Boolean printAll;

 //Machine to read from the hard disk
 private static  Scanner input;

 //Generator of random numbers
 private static Random r  = new Random();


 //**************MAIN**************
```

```
 public static void main( String args[] )
 {
  System.out.println("Please, wait for a while");
  //To print additional information, change to true
  printAll = false;
  //To include   blank spaces among words
  withBlanks = false;
  //Number of reported most common words
  nCommon = 1000;
  System.out.println( "\nLetters per n-gram = " + n );
  System.out.println( "Lenght of the alphabet = " +
                                     lengthAlf);
  System.out.println( "Number of n-grams in vector " +
                                "SelfEng = " + N);
  //Test
  /*
  System.out.println( "Index of AAA = " + index(3,"AAA"));
  System.out.println( "Index of AAÄ = " + index(3,"AAÄ"));
  System.out.println( "Index of ßßß = " + index(3,"ßßß"));
  */
  readTexts();
  work();
 }


 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }
```

```
// File is read
public  static  StringBuffer readData()
{
 StringBuffer text =  new StringBuffer("");
 //try and catch pairs
   try
   {
      while ( input.hasNext() )
      {
       text = text.append(input.next());
       if (withBlanks)
       {
        //A blank space is inserted after each word
        text = text.append(" ");
       }
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

//Text is gotten
```

```
private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================


//Training text in English
private static void readEngText1()
{
 String path = "/home/jose/AAjoser/AJose/Ciencia/" +
           "ImmuneDistances/BookDelight.txt";
 EngText1 =  getText(path);
 String s = EngText1.substring(0, 100);
 System.out.println(s);
}

//Reading  a second  text in English.
private static void readEngText2()
{
 String path = "/home/jose/AAjoser/AJose/Ciencia/" +
          "ImmuneDistances/SamuelButler.txt";
 EngText2 = getText(path);
}

//Reading a text in German
private static void readGerText()
{
  String path = "/home/jose/AAjoser/AJose/Ciencia/" +
           "ImmuneDistances/DieGKomoedie.txt";
  GerText = getText(path);
```

```
 }

//A random text  with l chars is composed
 private static StringBuffer composeText(int L)
 {
  randText =  new StringBuffer("");

  for(int i = 0; i < L; i++)
  {
   //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
   char c = Alphabet.charAt(l);
   //char is appended to text
   randText = randText.append(c);
  }
  return randText;
 }


 private static void readTexts()
 {
  if (withBlanks)
{
 Alphabet = Alphabet + ' ';
 N = Alphabet.length();
}
  readEngText1();
  nEngText1 = EngText1.length();
  System.out.println( "Lenght of trainingText = " +
                                nEngText1);
  readEngText2();
  nEngText2= EngText2.length();
  System.out.println( "Lenght of testerEng = " +
                                nEngText2);
  readGerText();
  nGerText = GerText.length();
  System.out.println( "Lenght of testerGer = " +
                                nGerText);
}
```

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



//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;
}


//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);
 c1 = Alphabet.charAt(p);
 if (withBlanks)
  {
   //Replace blank by a visible symbol
   if ( c1 == ' ') c1 = (int) (765);
  }
 indl = indl - p*k;
}
else c1 = 'A';
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

   }
  /*
  if (printAll)
  System.out.println( "ngram = " + ngram +
                   " index = "   + ind);*/
  return ngram;
 }


 /*
   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 intVector nGramsFrequency(int n,
                        StringBuffer Text)
{
 //Zeroed initialization
 intVector F = p.new intVector();
 nText = Text.length();
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
start = start +1;
  }
 return F;
}



//Returned vector contains the indexes of most frequent n-grams.
private static intVector commons(int n, intVector F)
{
 intVector h = p.new intVector(F);
 intVector C = p.new intVector();
 int Max;
 String s;
 for(int counter = 0; counter < nCommon; counter++)
 {
```

```
   Max = 0;
   //Find the max
   for(int i = 0; i < N; i++)
if (h.F[i] > h.F[Max] )  Max = i;
   //n-gram associated to Max
   s = indexInverse(n, Max);
   //System.out.print(s + " ");
   System.out.println(s + "("+ Max + ") " +
        "Freq = " + h.F[Max] + " ");

   //The max leaves the game
   h.F[Max] = 0;
   //The Max is recorded
   C.F[counter] = Max;
  }
  return C;
 }

 //The symmetric distance among two sets
 // of integers C1 and C2 is calculated.
 //Both sets have the same number of elements = nCommon.

 private static int distance(int nCommon,
               intVector C1, intVector C2)
 {

  for(int i = 0; i < nCommon; i++)
   {
for(int j = 0; j < nCommon; j++)
 {
  //A coincidence  is deleted
  //(an index appears in both vectors)
  if (C1.F[j] == C2.F[i])
   {
 C1.F[j] = -1;
  C2.F[i] = -1;

   }
 }
   }
```

```
  int d = 0;
  System.out.println("Elements in A but not in B " +
   "with their frequencies");
  int counter1 = 0;
  for(int i = 0; i < nCommon; i++)
  if (C1.F[i] > 0)
  {
   d = d+1;
   counter1++;
   System.out.println(counter1 + " " + indexInverse(n,C1.F[i])
   + "  " + freq1.F[C1.F[i]]);
  }
  System.out.println("Elements in B but not in A " +
   "with their frequencies");
  int counter2 = 0;
for(int i = 0; i < nCommon; i++)
  if (C2.F[i] > 0)
      {
   d = d+1;
   counter2++;
   System.out.println(counter2 + " " + indexInverse(n,C2.F[i])
   + "  " + freq2.F[C2.F[i]]);
  }
  return d;
 }

 //The correctness of the program hangs
 //on the index and indexInverse functions.
 //One is the inverse of the other.
 //Test at random
 private static void runTest1()
 {
  //Test for index functions
  System.out.println("\nTest for index functions. index()"
              +   "\nand indexInverse() must" +
              " be inverse one to another");
  System.out.println("Initial ngram  + " +
       "\t Initial index + \tFinal ngram  \n");
  for(int i = 0; i < 1000; i++)
   {
```

```
String si = composeText(3).toString();
System.out.print( si + "\t");
int ind1 = index(3,si);
System.out.print( ind1 + "\t");
String  sf = indexInverse(3, ind1);
System.out.print(  sf+ "\t");
if (si.equals(sf)) System.out.print( "Equal n-grams ");
   else  System.out.print( "Different n-grams " );
int ind2 = index(3,sf);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.println(" Equal indexes ");
   else  System.out.println( "Different indexes " );
   }
 }

 //The correctness of the program hangs
 //on the index and indexInverse functions.
 //One is the inverse of the other.
 //Test on demand.
 private static void runTest2()
 {
  //Test for index functions
  System.out.println("\nTest for index functions. index()"+
  "\n and indexInverse() must be inverse one to another");


//String si = "ßßß";
String si = "ßZn";
System.out.print( si + "\t");
int ind1 = index(3,si);
System.out.print( ind1 + "\t");
String  sf = indexInverse(3, ind1);
System.out.print(  sf+ "\t");
if (si.equals(sf)) System.out.println("Equal n-grams");
   else  System.out.println( "Different n-grams" );
int ind2 = index(3,sf);
System.out.print( ind2 + "\t");
if (ind1 ==ind2) System.out.println( " Equal indexes");
else  System.out.println( "Different indexes" );
```

```
ind1 = 151393;
System.out.print( ind1 + "\t");
 si = indexInverse(3, ind1);
System.out.print(  si+ "\t");
ind2 = index(3,si);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.println(" Equal indexes ");
else  System.out.println( "Different indexes" );
sf = indexInverse(3, ind2);
System.out.print( sf + "\t");
if (si.equals(sf)) System.out.println("Equal n-grams ");
   else  System.out.println( "Different n-grams" );
ind1 = 151392;
System.out.print( ind1 + "\t");
 si = indexInverse(3, ind1);
System.out.print(  si+ "\t");
ind2 = index(3,si);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.print(" Equal indexes ");
else  System.out.print( "Different indexes" );
sf = indexInverse(3, ind2);
System.out.print( sf + "\t");
if (si.equals(sf)) System.out.println("Equal n-grams");
   else  System.out.println( "Different n-grams" );
ind1 = 151394;
System.out.print( ind1 + "\t");
 si = indexInverse(3, ind1);
System.out.print(  si+ "\t");
ind2 = index(3,si);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.print( " Equal indexes ");
else  System.out.println( "Different indexes" );
sf = indexInverse(3, ind2);
System.out.print( sf + "\t");
if (si.equals(sf)) System.out.print( "Equal n-grams ");
   else  System.out.println( "Different n-grams" );
 }

 //List of all n-grams and their indexes
```

```
 private static void runTest3()
 {
  for(int i = 000; i < 1000; i++)
   {
String si = indexInverse(3,i);
int ind = index(n,si);
System.out.println(i + " "  +  si  + " " + ind);
   }
 }

 //
 private static void work()
 {
  //A test for correctness is done.
  Boolean Test = false;
  if (Test) runTest3();
  else
   {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}
    //The frequency of n.grams in EngText1 is calculated
    freq1 = nGramsFrequency(n, EngText1);
    String nGram = "and";
    int ind = index(n,nGram);
    System.out.println("\nTest: Frequency in EngText1 " +
     "of n-gram " +  nGram + " = " + freq1.F[ind]);
    //The nCommon most common n-grams are picked up
    System.out.println("\nA = " + nCommon + " most common "
                       + n +
               "-grams with  their indexes " +
               "\nand frequencies in EngText1");
    //The indexes of most common n-grams in EngText1
    //are recorded in Coomon1
    Common1 = commons(n,freq1);
    /*freq2 = study(n, EngText2);

    System.out.println("\n" + "B = " +  nCommon +
```

```
              " most common " + n
             + "-grams with their indexes  " +
           "and frequencies in EngText2"); */
    //The frequency of n.grams in GerText is calculated
    freq2 = nGramsFrequency(n, GerText);
    System.out.println("\nTest: Frequency in GerText" +
     " of n-gram " +   nGram + " = " + freq2.F[ind]);
    //The nCommon most common n-grams are picked up
    System.out.println("\n" + "B = " +  nCommon +
           " most common " + n
          + "-grams with their indexes  " +
       "\nand frequencies in the German text");
    //The indexes of most common n-grams in EngText1
    //are recorded in Common2
    Common2 = commons(n,freq2);
    System.out.println("\nList of not coincident n-grams");
    int d;
     d = distance(nCommon, Common1, Common2);
     System.out.println("The symmetric distance between " +
      "the two texts is " + d);
     System.out.println(" for the " + nCommon +
      " most common " +
      n + "-grams" );

  }
 }
}//end of main class I50 SymmetricDistance
```

51 ***Exercise.*** *Run the program and play with its code. Try out pairs of various languages. Include texts composed of chars that are chosen at random.*

52 ***Exercise.*** *Design your own experimental program to check out the next generalizations made by the Author:*

1. This method of most common n-grams to define a distance among texts is a good model of what a human does when he or she tries to identify at once in which language a given texts is.

2. For each number nCommon of most common words, we have a distance. The distance for small nCommon is as informative as for a moderate one.

3. The relative distance (the ratio of found distance to maximal possible distance) tends to decrease as the number of considered most common n-grams increases.

4. The 18 most common n-grams of each language are distinctive of it and do not appear in the other.

5. The distance between English and German is for small values of most common n-grams very similar to that between German and random texts.

6. The last two claims seem to contradict the very natural idea about a continuous evolution of languages along human dispersal.

53 ***Exercise.*** *Reuse the previous code to develop a program to decide whether a given text is written in English else in German -assuming that it is written in one of them.*

54 ***Intrigues.*** *Our n-grams distinguish upper case from low one: is this the best option to produce highly effective discriminating rules? Written languages not always coincide with spoken ones: Which is the most important? How shall we measure the distance between English and Russian or Greek if we use written texts?*

55 ***Challenge.*** *Find a reliable procedure to use the most common words to discriminate among English texts written by different authors.*

56 ***Challenge.*** *Reuse our software to verify else reject the Zipf's law in regard with your own language: the frequency of any word in a text is inversely proportional to its rank in the frequency table. This says that the second most frequent word occurs half as often the most frequent item and so on (Wikipedia, 2013 a, b; Gelbukh et al, 2001; Powers, 2013?).*

2.2 Conclusion

It is amazing how easily one can recognize the language in which a text is written even with scare training. This must imply that the underlying mechanism must be very simple. In this chapter we have proposed a methodology that possibly generalizes that procedure and that in any case is simple and effective: one pays attention to those distinctive n-grams that are in the language but that do not exist in the others languages one knows. This was readily formalized with the symmetric

distance between pair of sets. Somehow surprising is the fact that the 18 more frequent 3-grams of both German and English are distinctive of them. This motivates a terrible question: is this finding compatible with the naturally accepted belief that languages evolved continuously along human lineages across their dispersal both in time and space?

Chapter 3

Facilitated distances

`Projecting your set into a distance space`

57 ***Purpose***

A distance or metric space is a set that is furnished with a distance, a function to measure the differences among their elements. Some distance spaces are of immediate concern, say, the plane. A **facilitated distance** takes a set into a distance space and uses the distance of the hosting space to measure distances over the former set. Our purpose is to pay special attention to **Euclidean distances** which use the plane and its generalizations as hosting spaces, in which the distance takes the familiar euclidean form. Next, we plug that distance into our n-gram technology.

3.1 Euclidean distances

Walking induces a sort of natural distance: the distance between two points is the number of steps across the usual path connecting the two points. In a plain terrain, the usual path is along the straight line that connects the two points and the number of steps is precisely a multiple of the **Euclidean distance**, which assigns to a pair of points (x,y) and (z,w) a distance d defined by:

$$d = \sqrt{(x-z)^2 + (y-w)^2}$$

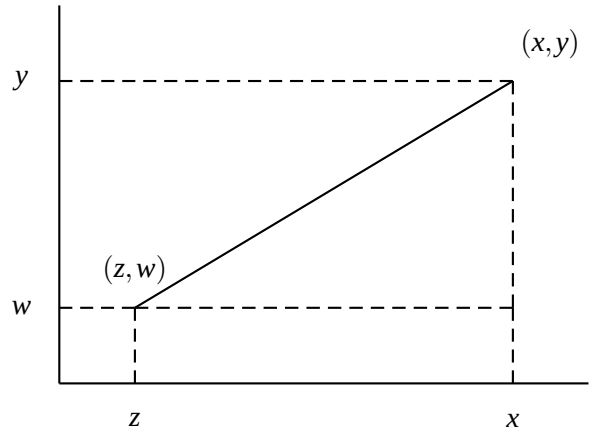


Figure 3.1. The Euclidean distance assigns to two points of the plane a multiple of the distance that one measures by walking along a straight line connecting the two points and counting the number of steps.

The plane with the Euclidean distance is the prototype of distance space. This distance is generalized in the expected form for R^n, the space of n-tuples. If one n-tuple is $(x_1, x_2, ..., x_n)$ and the other is $(y_1, y_2, ..., y_n)$, the distance between them is defined by:

$$d = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2) + ... + (y_n - x_n)^2}$$

For instance, the Euclidean distance between $(0,1,2,3,4)$ and $(5,6,7,8,9)$ is in R^5 given by

$$\begin{aligned} d &= \sqrt{(5-0)^2 + (6-1)^2 + (7-2)^2 + (8-3)^2 + (9-4)^2} \\ &= \sqrt{(5)^2 + (5)^2 + (5)^2 + (5)^2 + (5)^2} = \sqrt{125} = 11.18. \end{aligned}$$

We will refer to R^n with its Euclidean distance as the **Euclidean space**, and the n will be any natural number from 1 to ∞.

It could be appropriate to recall that not all functions classify as distances:

A or function d that assigns to any pair of elements in a set a real number must comply with some conditions to classify as a **distance**:

1. The distance between an object and itself must be always zero: $d(a,a) = 0$.
2. The distance between objects a and b must be equal to that between b and a, $d(a,b) = d(b,a)$.

3. The direct distance must be shorter than an indirect one: $d(a,b) \leq d(a,c) + d(c,b)$. This is the so called triangle inequality.

All these characteristics work together to make of distances a marvelous tool that serves to cluster objects according to their similitude or to discriminate them according to their difference. Thus, distances are at the base of genetic clustering, phylogenies, pedigrees and related affairs.

58 ***Facilitated distances***

To define a method to distinguish self from not-self is equivalent to impose a discrete distance, which assigns 0 to the distance between an object and itself, and 1 to the distance between two different objects. In spite of its tremendous biological importance, discrete distances presents a white-black picture of the world. So, an immediate question arises: How shall we modify a discrete distance to see things in many diverse tones of gray? One answer to this technical question is immediate: use a distance space as intermediate. More concretely:

Let (M,d) be a distance space M with distance d. And let S be a set upon which we want to define a distance. Use a function p to project or represent S into M. The projection p may result from many sides, say, it counts the number of shared attributes from a list. Define d_p the distance between elements A and B of S as

$$d_p(A,B) = d(p(A), p(B)).$$

The distance d works over M while d_p over S. Space M is the hosting space and d_p the induced distance over S by p and d.

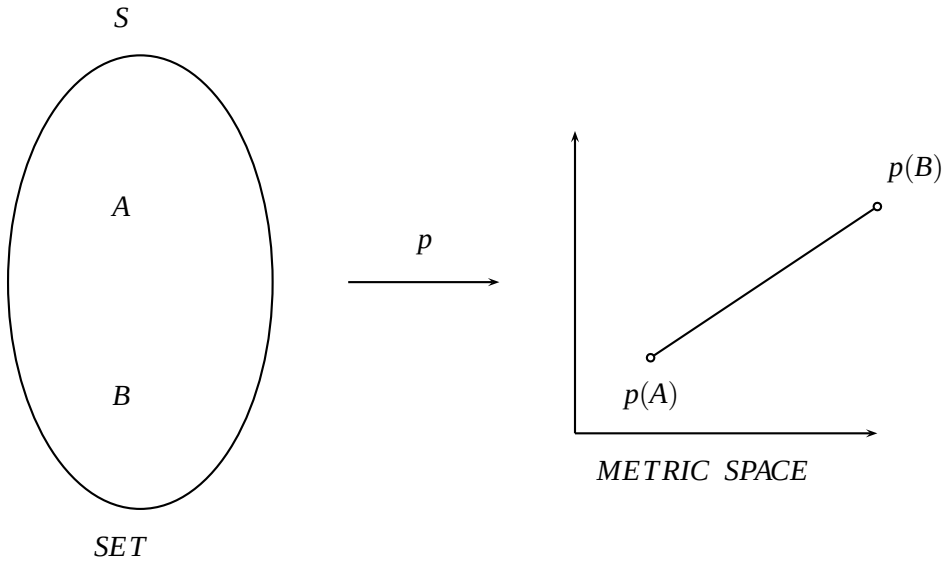


Figure 3.2. A facilitated distance results when a set S is represented by p into a metric or distance space M and its distance is used to measure distances between pairs of elements in S.

The idea of facilitation is very natural and so it can be applied without much ceremonies and might pass unnoticed. Such may be the next case.

3.2 Euclidean n-gram distance

We have produced a form for listing all possible n-grams for a given alphabet together with the absolute frequency of appearing of each n-gram in a given text. Thus, for any text, we can produce the corresponding vector of frequencies of n-grams. This vector is an n-tuple of real numbers, i.e., it is an element of some R^n, and so we have a projection of the sets of texts into a metric space. We can use its Euclidean distance to measure the distance between two frequency vectors and so between two texts.

We have here a case of a facilitated distance that we will call **Euclidean n-gram distance**.

59 ***The code for a Euclidean n-gram distance over texts follows.***

```
/*Program I59 EuclideanDistance
```

```
 This program calculates the
 Euclidean n-gram distance
 between two given texts.

 The program has three steps:

 Step one:
 The frequencies of n-grams captured from a given text
 defines an n-tuple, an element of the form
 (a1,a2,...,an).  The whole operation projects the text
 into R^n, the set of n-tuples.
 This operation is done with the help
 of an indexing procedure.

 Step two:
 Two texts are given for examination.
 All n-grams of each text are captured
 and their absolute
 frequencies are recorded.

 Step three:
 The Euclidean n-gram distance between the two given texts
 is defined as  the  Euclidean distance between
 the n-tuples of absolute frequencies of n-grams.


*/



import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.lang.IllegalStateException;
import java.util.Scanner;

public class EuclideanDistance
{
 //The application is instantiated
```

```
 static EuclideanDistance p = new EuclideanDistance();

 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n=3;
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);


 //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector3
{
int L = (int) Math.pow(lengthAlf+1, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector3( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector

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


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
```

```
private static intVector3  freq1, freq2
                                = p.new intVector3();


//Training  text. A StringBuffer is a heavy-duty String.
private static StringBuffer EngText1, EngText2,
                             GerText;

private static  Boolean withBlanks;


private static int nEngText1, nEngText2,
                  nGerText, nText;
private static Boolean printAll;

//Machine to read from the hard disk
private static  Scanner input;


//**************MAIN**************

public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //To print additional information, change to true
 printAll = false;
 //To include   blank spaces among words
 withBlanks = false;

 System.out.println( "\nLetters per n-gram = " + n );
 System.out.println( "Lenght of the alphabet = " +
                                  lengthAlf);
 System.out.println( "Number of n-grams in vector " +
                             "SelfEng = " + N);
 //Test
 /*
 System.out.println( "Index of AAA = " + index(3,"AAA"));
 System.out.println( "Index of AAÄ = " + index(3,"AAÄ"));
 System.out.println( "Index of ßßß = " + index(3,"ßßß"));
 */
```

```
  readTexts();
  work();

 }


 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }

 // File is read
 public  static  StringBuffer readData()
 {
  StringBuffer text =  new StringBuffer("");
  //try and catch pairs
    try
    {
       while ( input.hasNext() )
       {
        text = text.append(input.next());
        if (withBlanks)
        {
         //A blank space is inserted after each word
         text = text.append(" ");
        }
       }
    }
    catch ( NoSuchElementException elementException )
    {
```

```
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

//Text is gotten
private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================


//Training text in English
private static void readEngText1()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
```

```
            "ImmuneDistances/BookDelight.txt";
  EngText1 =  getText(path);
  String s = EngText1.substring(0, 100);
  System.out.println(s);
 }

 //Reading  a second  text in English.
 private static void readEngText2()
 {
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
           "ImmuneDistances/SamuelButler.txt";
  EngText2 = getText(path);
 }

 //Reading a text in German
 private static void readGerText()
 {
   String path = "/home/jose/jose/AAjoser/Ciencia/" +
            "ImmuneDistances/DieGKomoedie.txt";
   GerText = getText(path);
 }




 private static void readTexts()
 {
  if (withBlanks)
{
 Alphabet = Alphabet + ' ';
 N = Alphabet.length();
}
  readEngText1();
  nEngText1 = EngText1.length();
  System.out.println( "Lenght of trainingText = " +
                                nEngText1);
  readEngText2();
  nEngText2= EngText2.length();
  System.out.println( "Lenght of testerEng = " +
                                nEngText2);
```

```
  readGerText();
  nGerText = GerText.length();
  System.out.println( "Lenght of testerGer = " +
                                nGerText);
}

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



 //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 intVector3 nGramsFrequency(int n,
                        StringBuffer Text)
{
 //Zeroed initialization
 intVector3 F = p.new intVector3();
 nText = Text.length();
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
start = start +1;
  }
 return F;
}


//The Euclidean distance among two vectors of integer type
//(declared as an object intVector3) is calculated
private static double euclideanDistance(int n,
             intVector3 vect1, intVector3 vect2)
{
 double sum = 0;
 double d = 0;
```

```
  for(int i = 0; i < n; i++)
   {
  long a = vect1.F[i]-vect2.F[i];
sum = sum + a*a;
d = Math.sqrt(sum);
   }
  return d;
 }

 //Operative part
 private static void work()
 {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}
    //Frequencies of n-grams in texts are calculated
    freq1 = nGramsFrequency(n, EngText1);
    freq2 = nGramsFrequency(n, GerText);
    //Euclidean distance between freq1 and freq2 is calculated
     double d = euclideanDistance(N, freq1,freq2);
     System.out.println("The Euclidean n-gram distance between " +
      "the two texts is " + d);
 }


}//end of main class I59 EuclideanDistance
```

60 ***Exercise.*** *Run the program and play with its code.*

61 ***Exercise.*** *The previous code is intended to process large texts that must be read from the hard disc. Nevertheless, if it is wrong, everything bellow will be wrong. So, it is mandatory to make some tests for correctness. So, make the necessary amendments to find the distance between very simple texts, whose correct answer can be verified directly.*

62 ***Exercise.*** *Add to the previous code the possibility to compose random texts to carry out comparative studies.*

3.3 Conclusion

We have used the absolute frequencies of diverse n-grams of a text to measure the distance between two texts: those frequencies are n-tuples and we apply to them the Euclidean distance which is an abstraction of the method of measuring distances between two points of a plane by counting the number of steps across the straight line that joins them. The resultant distance has been called Euclidean n-gram distance. It can be applied to pair of texts of the most varied nature and diversity and so our immediate concern is to apply it to DNA or protein sequences.

Chapter 4

Application to proteomes

`Distances for protein sequences`

63 ***Introduction and objective.*** *Working on human languages we have tested with positive results the biological theory about the immune system claiming that the used mechanism to distinguish self from non-self rests on a learning procedure that kills self recognizers and allows others to survive. So, in the previous chapters we have made on our own a research on the potentialities of n-grams to measure differences among texts. In unison with the whole world that uses n-grams in automatic language recognition our verdict is that this enterprise is plainly successful. Our purpose in the present chapter is to try our programs on DNA or protein texts. We face a very intriguing question: can we use or reuse our software to measure differences among genomes and to decide when they belong in different species? Since immune assays has been used in vitro to do this, we expect a positive answer to this question in our silico world. Therefore, if we fail, we must conclude that we have missed some very important biological concept or that our software has a misleading bug. If the answer is positive, we still have the task of explaining what enables such a possibility.*

4.1 Reuse of previous software

We have worked with human languages and with Latin and German alphabets. Now we turn to DNA and protein texts.

64 ***Exercise: Reuse for proteome analysis.*** *A proteome is a set of sequences with all the protein sequences of a given living being. Explain what must be done to adapt our software to the analysis of proteomes. Compare your answer with ours in a program below.*

65 ***Reuse for DNA analysis***

To reuse our program for the study of DNA all we need is to change the employed alphabet by {A,T,C,G} or another suitable extension, say {A,T,C,G, D} that includes the symbol D for the dimer TT, and so on.

66 ***Devising fictitious DNA species***

With the only purpose of checking out our software, let us learn to devise diverse fictitious DNA species to test next whether or not our software can detect that diversity.

The idea is as follows: we use the random generator to produce DNA texts with a given distribution of frequencies of bases A,T,C and G. If we change the distribution, we produce a different species of DNA: can that difference be detected by our software? The only allowed answer is positive. Let us check for that.

```
/*Program I66 FictitiousDNA

 We use the random generator to produce DNA texts with
 a given distribution of  frequencies of bases A,T,C and G.
 If we change the distribution,
 we produce a different species of DNA:
 can that difference be detected by our software
 that calculates the Euclidean n-gram distance
 between two given texts?


 The Euclidean n-gram distance
 is calculated in three steps:

 Step one:
 The frequencies of n-grams captured from a given text
 defines an n-tuple, an element of the form
 (a1,a2,...,an).  The whole operation projects the text
 into R^n, the set of n-tuples.
 This operation is done with the help
 of an indexing procedure.

 Step two:
```

```
 Two texts are given for examination.
 All n-grams of each text are captured
 and their absolute
 frequencies are recorded.

 Step three:
 The Euclidean n-gram distance between the two given texts
 is defined as the  Euclidean distance between
 the n-tuples of absolute frequencies of n-grams.

*/



import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.util.Random;
import java.lang.IllegalStateException;
import java.util.Scanner;

public class FictitiousDNA
{
 //The application is instantiated
 static FictitiousDNA p = new FictitiousDNA();


 //English + German chars, only the most informative
 private static String Alphabet =    "ATCG";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n = 10;
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);


 //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
```

```
private  class intVector6
{
int L = (int) Math.pow(lengthAlf+1, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector6( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector

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


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
 private static intVector6  freq1, freq2
                                 = p.new intVector6();


 //Training  text. A StringBuffer is a heavy-duty String.
 private static StringBuffer EngText1, EngText2, GerText,
                       randomText,randomText1, randomText2;

 private static  Boolean withBlanks;


 private static int nEngText1, nEngText2,
                    nGerText, nText;
 private static Boolean printAll;

 //Machine to read from the hard disk
 private static  Scanner input;
```

```
//Test with simple strings
private static boolean test = false;

private static StringBuffer s1,s2;

private static int lengthRandomText = 10000;

private static  double DistriBases1[] = new double[4];

private static  double  DistriBases2[] = new double[4];


//Generator of random numbers
private static Random r  = new Random();

//**************MAIN*************

public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //To print additional information, change to true
 printAll = true;
 //To include   blank spaces among words
 withBlanks = false;

 System.out.println( "\nLetters per n-gram = " + n );
 System.out.println( "Lenght of the complete alphabet = "
                            +     lengthAlf);
 System.out.println( "Number of n-grams in vector " +
                             "SelfEng = " + N);
 //The test is included
 test = false;
 if (test)  readTextsTest();
 /*
     else
      {
       readTexts();
          randomText = composeText(lengthRandomText);
        }
```

```
     */



  System.out.println( "Length of randomText = "
                                + lengthRandomText);

  //Fictitious DNA species 1:

  //Probabilities of A,T,C and G:
  DistriBases1[0] = 0.25;
  DistriBases1[1] = 0.25;
  DistriBases1[2] = 0.25;
  DistriBases1[3] = 0.25;
  randomText1 = composeTexts(lengthRandomText, DistriBases1);

  //Fictitious DNA species 2:

  //Probabilities of A,T,C and G:
  DistriBases2[0] = 0.10;
  DistriBases2[1] = 0.20;
  DistriBases2[2] = 0.30;
  DistriBases2[3] = 0.40;
  randomText2 = composeTexts(lengthRandomText, DistriBases2);
  analizeRandomTexts();
 }

//A test over simple pairs of strings is carried out
private  static  void readTest()
{
 //Very short alphabet
 Alphabet = "ab";
 lengthAlf = Alphabet.length();
 //Number of n-grams
 N = (int) Math.pow(lengthAlf, n);
 System.out.println( "Number of n-grams short alphabet = "
                                                    + N);
 s1 = new StringBuffer ( "aaaaaaaaaaaaaaaaa" ) ;
 s2 = new StringBuffer ( "aaaaababaabbbaababbbabbb" ) ;
```

```
}



//A random text  with l chars is composed
private static StringBuffer composeText(int L)
{
StringBuffer randText =  new StringBuffer("");

for(int i = 0; i < L; i++)
{
 //a random integer number
 int l = r.nextInt(lengthAlf);
 //A random char
 char c = Alphabet.charAt(l);
 //char is appended to text
 randText = randText.append(c);
}
return randText;
}


//A random text  with L chars is composed.
//Frequencies of A,T,C,G follow the distribution
//given by Distri
private static StringBuffer composeTexts(int L,
                     double[] Distri)
{
 StringBuffer randText =  new StringBuffer("");

 for(int i = 0; i < L; i++)
 {
  char c = 'G';
  //a random double number
  double a = r.nextDouble();
  //Filtering process
  if ( a < Distri[0]+ Distri[1]+ Distri[2])  c = 'C';
  if ( a < Distri[0]+ Distri[1])            c = 'T';
  if ( a < Distri[0] )                      c = 'A';
  //char is appended to text
```

```
  randText = randText.append(c);
}
return randText;
}



private static void readTextsTest()
{
 readTest();
 EngText1 = s1;
 GerText = s2;
 nEngText1 = EngText1.length();
 System.out.println( "EngText1 = " + EngText1 +
 ", Lenght  = " +
                                nEngText1);
 nGerText = GerText.length();
 System.out.println( "GerText = " + GerText +
 ", Lenght = " +
                                nGerText);
}



 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }
```

```
// File is read
public  static  StringBuffer readData()
{
 StringBuffer text =  new StringBuffer("");
 //try and catch pairs
   try
   {
      while ( input.hasNext() )
      {
       text = text.append(input.next());
       if (withBlanks)
       {
        //A blank space is inserted after each word
        text = text.append(" ");
       }
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

//Text is gotten
private static StringBuffer getText(String path)
```

```
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================


//Training text in English
private static void readEngText1()
{
 String path = "/home/jose/AAjoser/AJose/Ciencia/" +
           "ImmuneDistances/BookDelight.txt";
 EngText1 =  getText(path);
 String s = EngText1.substring(0, 100);
 System.out.println(s);
}

//Reading  a second  text in English.
private static void readEngText2()
{
 String path = "/home/jose/AAjoser/AJose/Ciencia/" +
          "ImmuneDistances/SamuelButler.txt";
 EngText2 = getText(path);
}

//Reading a text in German
private static void readGerText()
{
  String path = "/home/jose/AAjoser/AJose/Ciencia/" +
           "ImmuneDistances/DieGKomoedie.txt";
  GerText = getText(path);
}
```

```
 private static void readTexts()
 {
  if (withBlanks)
{
 Alphabet = Alphabet + ' ';
 N = Alphabet.length();
}
  readEngText1();
  nEngText1 = EngText1.length();
  System.out.println( "Lenght of trainingText = " +
                               nEngText1);
  readEngText2();
  nEngText2= EngText2.length();
  System.out.println( "Lenght of testerEng = " +
                               nEngText2);
  readGerText();
  nGerText = GerText.length();
  System.out.println( "Lenght of testerGer = " +
                               nGerText);
}

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



 //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 intVector6 nGramsFrequency(int n,
                        StringBuffer Text)
{
 //Zeroed initialization
 intVector6 F = p.new intVector6();
 nText = Text.length();
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
 start = start +1;
   }
  return F;
 }


 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
               intVector6 vect1, intVector6 vect2)
 {
  double sum = 0;
  double d = 0;
  if (printAll)
System.out.println("Frequency vectors to compare");
  for(int i = 0; i < N; i++)
   {
  long a = vect1.F[i]-vect2.F[i];
sum = sum + a*a;
d = Math.sqrt(sum);
if (printAll)
System.out.println(vect1.F[i] + "  " + vect2.F[i]);
   }
  return d;
 }

 //Operative part
 private static void work()
 {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}
```

```
//EngText1 vs GerText
    //Frequencies of n-grams in EngText1 are calculated
if (printAll)
 System.out.println("\nn-gram analysis of EngText1");
    freq1 = nGramsFrequency(n, EngText1);
    if (printAll)
     {
     System.out.println("Frequencies of n-grams");
      for(int i = 0; i < N; i++)
    System.out.println(freq1.F[i]);
      System.out.println();
     }
    if (printAll)
    System.out.println("\nn-gram analysis of GerText");
    freq2 = nGramsFrequency(n, GerText);
    if (printAll)
    {
    System.out.println("Frequencies of n-grams");
     for(int i = 0; i < N; i++)
  System.out.println(freq2.F[i]);
     System.out.println();
    }
    //Euclidean distance between freq1 and freq2
     double d = euclideanDistance(N, freq1,freq2);
     System.out.println("\nThe Euclidean n-gram distance" +
      " between the two texts, \nEngText1 and " +
      "GerText, is " + d);
  //EngText1 vs RandomText
     //Frequencies of n-grams in EngText1 are calculated
  if (printAll)
   System.out.println("\nn-gram analysis of EngText1");
     freq1 = nGramsFrequency(n, EngText1);
     if (printAll)
      {
      System.out.println("Frequencies of n-grams");
       for(int i = 0; i < N; i++)
     System.out.println(freq1.F[i]);
       System.out.println();
      }
     if (printAll)
```

```
     System.out.println("\nn-gram analysis of randomText");
     freq2 = nGramsFrequency(n, randomText);
     if (printAll)
     {
     System.out.println("Frequencies of n-grams");
      for(int i = 0; i < N; i++)
    System.out.println(freq2.F[i]);
      System.out.println();
     }
     //Euclidean distance between freq1 and freq2
      d = euclideanDistance(N, freq1,freq2);
      System.out.println("\nThe Euclidean vector " +
       "distance  between the two texts, " +
       "\nEngText1 and randomText,  is " + d);
 }

 //Operative part
 private static void analizeRandomTexts()
 {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}

  //randomText1 vs randomText2
     //Frequencies of n-grams in randomText1 are calculated
  if (printAll)
   System.out.println("\nn-gram analysis of randomText1");
     freq1 = nGramsFrequency(n, randomText1);
     if (printAll)
      {
      System.out.println("Frequencies of n-grams");
       for(int i = 0; i < N; i++)
     System.out.println(freq1.F[i]);
       System.out.println();
      }
     //Frequencies of n-grams in randomText2 are calculated
     if (printAll)
     System.out.println("\nn-gram analysis of randomText2");
```

```
     freq2 = nGramsFrequency(n, randomText2);
     if (printAll)
     {
      System.out.println("Frequencies of n-grams");
      for(int i = 0; i < N; i++)
    System.out.println(freq2.F[i]);
      System.out.println();
     }
     //Euclidean distance between freq1 and freq2
      double d = euclideanDistance(N, freq1,freq2);
      System.out.println("\nThe Euclidean n-gram distance" +
       " between  the two texts, " +
       "randomText1 and randomText2 is " + d);
 }

}//end of main class I66 FictitiousDNA
```

67 ***Exercise.*** *Run the program and play with the code. You can turn off some printing procedures by setting in line 127 printAll = false. Play enough to support else reject the appreciation of the Author that our software is really good to discriminate different fictitious DNA species.*

68 ***Exercise.*** *Verify, as expected, that when one restricts the study to 1-grams, one obtains as frequencies of 1-grams the injected probabilities of different bases.*

69 ***Exercise and challenge.*** *Verify, as unexpected, that when one varies n in the study of n-grams, one obtains a distance that varies as a decreasing function that resembles $y = k + a/x^j$ and $y = Ce^{-}kx$ or something else. Explain why your selected functional dependence is observed.*

70 ***Challenge.*** *Verify and prove else reject the intuitive proposal of the Author: to maximize the information given by n-grams, one needs to take just a super-distance that is the sum of the Euclidean n-gram distance for n-grams for just two different values of n.*

71 ***Challenge.*** *Theoretically, our software should be able to distinguish apart two fictitious DNA species if just sufficiently large texts are provided. But in practical terms, texts must be finite and preferentially short. The problem is that for short texts, the distance is not a number but a random variable with some variance. To*

find a good compromise between length of texts and resolution, one must include somehow that variance. Realize a study and a solution to this problem. Hint: look at our programs that proposed a solution to this optimization problem, say, the hologramic methodology in program I33 LargeTexts.

72 ***Our directive and challenge.*** *We dismiss sampling effects that affect the discrimination of short texts when considering complete genomes. This means that the size of complete genomes is declared to be infinite. But if that is accepted, where must we place genetic polymorphism?*

4.2 Protein sequences from Internet

We have made the mechanism of the immune system into a set of tools to discriminate among texts. The fundamental abstraction is the concept of n-gram, a sequence of n letters taken from a given alphabet. It happens that this concept is an ordinary tool in our modern word dominated by information retrieval as one can guess from, say, the invitation of Microsoft (2013) to get engaged in the research to use n-grams to improve web search efficiency.

73 ***The work of Bailin Hao***

The academic application of n-grams to the study of genome diversity has been also seriously undertaken. In particular, Bailin Hao has made of this theme a project for his life and career (Hao, 2013a).

We consider that it is impossible to get expertise outside from his works, which are grouped under the title of CVTree (Composition Vector Tree). Besides, he offers a service of a permanently updated bank of complete genomes together with a web service of computational tools. Let us see.

74 ***Retrieving sequences of complete genomes***

We might follow the next steps to retrieve the full library of complete genomes:

1. Open the site

   `http://tlife.fudan.edu.cn/cvtree/` (revised 20/V/2013).

   It is desirable to give a look at the *Online User's Manual*. If you want to use the server to run the local application, you must *create a new project* and select the desired genomes to be compared. A new project is not necessary if you just want to download some sequences in protein format.

2. Left-click on the box *Example project*.

3. Left-click on *See details*. A list of genomes will appear. Go to the end of the list and there *uncheck* all species. Next, select those species you want, 7 will suffice to begin with. To be precise, one species is all we need for most of our wok: the Author chose `Arcobacter_L_uid158135.faa`. In any case, the system of the tlife site does not allow the selection of more than 900 species per download.

4. Return to the main page and there click *download*. Wait for the result which is a link to a great file in the server with all your sequences. Click over the link and save your great file to an appropriate folder.

5. Decompress your great file into a bundle of files each one with the complete proteome of a living being, most probably a bacteria. Check one of them with a text editor to see its contain full of protein sequences, many of which are of unknown function. Notice that the name of files end with the suffix .faa (see below).

6. You can edit your files by hand to get away with commentaries and use our previous software to solve a given task, say, to find distances between pair of genomes. Else, use a program below which deals not with specific archives but with all files of a given folder. Maybe you will prefer to work with a sample of, say, 7 genomes in a separated folder.

75 ***The FASTA format for protein sequences***

Our files have the extension .faa, which means that they have been originally retrieved from the NCBI (2013) and that come in **FASTA** format for Amino Acids (NCBI, 2013):

Example:

```
>gi|129295|sp|P01013|OVAX_CHICK GENE X PROTEIN (OVALBUMIN-RELATED)
QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAFNAEDTREMPFHVTKQESKPVQMMCMNNSFNVATLPAE
KMKILELPFASGDLSMLVLLPDEVSDLERIEKTINFEKLTEWTNPNTMEKRRVKVYLPQMKIEEKYNLTS
VLMALGMTDLFIPSANLTGISSAESLKISQAVHGAFMELSEDGIEMAGSTGVIEDIKHSPESEQFRADHP
FLFLIKHNPTNTIVYFGRYWSP
```

The first line is the description of the sequence, while all other lines contain sequence data. The description line (defline) is distinguished from the sequence data by a greater-than (>) symbol at the beginning. This line consists of an identifier and function, if it is known.

The FASTA code for amino acids are:

```
A  alanine               P  proline
B  aspartate/asparagine  Q  glutamine
C  cystine               R  arginine
D  aspartate             S  serine
E  glutamate             T  threonine
F  phenylalanine         U  selenocysteine
G  glycine               V  valine
H  histidine             W  tryptophan
I  isoleucine            Y  tyrosine
K  lysine                Z  glutamate/glutamine
L  leucine               X  any
M  methionine            *  translation stop
N  asparagine            -  gap of indeterminate length
```

This implies that we work with the alphabet given by
ABCDEFGHIKLMNPQRSTUVWYZX*-

Let us notice that we have more than the ordinary 20 amino acids encoded by the genetic code. This is due to the fact that after synthesis in the ribosomes some amino acids of some proteins or enzymes undergo biochemical transformation.

76 ***Filtering commentaries.*** *Let us learn how to filter, ignore, the first line of a sequence in .faa format. The executing code follows:*

```
//Program I76 GenomeCleaner
//This program reads a  genome from a file,
//ignores commentaries
//and displays its content.

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;
```

```
public class GenomeCleaner
{
  private static  Scanner input;

  private static String nameOfFolder =
  "/home/jose/jose/AAjoser/Ciencia/" +
            "GenomesComplete/GenomesSample/";
  private static String nameOfFile =
  "Arcobacter_L_uid158135.faa";

  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer genome = new StringBuffer("");




   // We read  record from file
   public  static  void readData() throws IOException
   {
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
       {
       //Filtering of first line of each subsequence
       if (s.charAt(0) != '>' )
     {
       System.out.println(s);
           genome = genome.append(s);
     }
       }

       fr.close();
      }
```

```
      catch ( NoSuchElementException elementException )
      {
         System.err.println( "File is corrupted." );
         input.close();
         System.exit( 1 );
      }
      catch ( IllegalStateException stateException )
      {
         System.err.println( "Reading aborted." );
         System.exit( 1 );
      }
   }


   public static void main(String[] args) throws IOException
   {
 readData();
   }
}//End of main class I76 GenomeCleaner
```

77 ***Exercise.*** *Adapt the code to your files and run it. Play with the code, say, reuse it to read and print a file with .tex extension.*

78 ***Exercise.*** *Add to the previous code the function of n-gram analysis, i.e., the program must receive a genome as input and must output a vector of n-gram frequencies.*

79 ***Challenge.*** *The previous program produces a very large output that overburdens the console of Eclipse. Made the necessary amendments to use the hard disc as receiver of output information. This will be a recurrent need in the future.*

Following an idea of Bailin Hao (cited above), we proceed to design a graphic colored display of the vector of absolute frequencies of n-grams for a given genome.

80 ***A visor for n-gram analysis.*** *The next code reports an n-gram in a graphic according to a coloring code.*

```
//Program I80 nGramVisor
//This applet reads a  proteome from a file,
//ignores commentaries,
//displays its content
```

```
//and carries an n-gram analysis.
//Report has a graphic form
//according to a coloring code.

//Fasta .faa format is required.


import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;


public class nGramVisor extends Applet
{

 private static final long serialVersionUID = 1L;

  //The application is instantiated
  private  static nGramVisor p =
                     new nGramVisor();
  //Machine for reading from hard disc
  private static  Scanner input;

  private static String nameOfFolder =
  "/home/jose/jose/AAjoser/Ciencia/" +
            "GenomesComplete/GenomesSample/";
  private static String nameOfFile =
  "Arcobacter_L_uid158135.faa";

  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer proteome =
                       new StringBuffer("");
```

```
  //The alphabet conforms to  fasta format
  //for amino acids (.faa extension).

  private static String Alphabet =
                    "ABCDEFGHIKLMNPQRSTUVWYZX*-";

  //length of the Alphabet
  private static  int lengthAlf = Alphabet.length();
  //letters per n-gram
  private static  int n = 3;
  //Number of n-grams
  private static  int N = (int) Math.pow(lengthAlf, n);




  private static boolean print1 = false;
  private static boolean print2 = false;


  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector7
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector7( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }
```

```
}//end of class intVector


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


private static intVector7  freq1
                                = p.new intVector7();



//This is the first method to be executed
public void init()
{
 //Size of applet
 this.setSize(2000,2000);
 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
        + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);
 print1 = false;
 print2 = false;
}

//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.

public void paint (Graphics g)
 {
  try
  {
   //n-gram frequencies are found and kept in freq1
findFrequencies();
  }
 catch (IOException e)
  {
e.printStackTrace();
```

```
  }
  //Width of pixel
  int width = 5;
  int max = findMax();
  System.out.println("Scale goes from 0 to " + max);
  System.out.println("Code for intensity from less to more" +
   ": \n black (absence), gray, blue, cyan, yelow," +
   " orange, pink, red");
  int nPoints = freq1.F.length;
  boolean go = true;
  int pixelsPerRow = 100;
  int nRows = nPoints / pixelsPerRow +1;
  int counter = 0;
  for (int i = 0; (i < nRows) & go; i++)
   {
for (int j = 0; (j < pixelsPerRow) & go; j++)
 {
  int index = i*pixelsPerRow + j;
  //Everything begins in black = absence
  g.setColor(Color.BLACK);
  //Colors are acquired as frequency increases
  if ( freq1.F[index] >  0)
        g.setColor(Color.GRAY);
  if ( freq1.F[index] >  max/7)
        g.setColor(Color.BLUE);
  if ( freq1.F[index] >  2* max/7)
        g.setColor(Color.CYAN);
  if ( freq1.F[index] >  3* max/7)
        g.setColor(Color.YELLOW);
  if ( freq1.F[index] >  4* max/7)
        g.setColor(Color.ORANGE);
  if ( freq1.F[index] >  5* max/7)
        g.setColor(Color.PINK);
  if ( freq1.F[index] >  6* max/7)
        g.setColor(Color.RED);
  //Pixel is drawn
  g.fillOval( j*width,  i*width, width, width );
  if (counter >= nPoints-1) go = false;
  counter = counter+1;
  //System.out.println("index = " + index );
```

```
 }
}
 }


public static void findFrequencies() throws IOException
{
 readData(path);
 proteome = readData(path);
 System.out.println("length of proteome = "
         + proteome.length());
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  {
 System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
               + "\t" + freq1.F[i]);
  }
 findMax();
}


   // We read  record from file
   public  static  StringBuffer readData(String path)
                        throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
       {
       //Filtering of first line of each subsequence
       if (s.charAt(0) != '>' )
```

```
    {
      //System.out.println(s);
      text = text.append(s);
    }
      }

      fr.close();
     }
     catch ( NoSuchElementException elementException )
     {
        System.err.println( "File is corrupted." );
        input.close();
        System.exit( 1 );
     }
     catch ( IllegalStateException stateException )
     {
        System.err.println( "Reading aborted." );
        System.exit( 1 );
     }
     return text;
  }


  //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 (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

//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 ;
```

```
  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                   " index = "   + ind);
 return ngram;
}


/*
  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 intVector7 nGramsFrequency(int n,
                          StringBuffer Text)
{
 //Zeroed initialization
 intVector7 F = p.new intVector7();
 int nText = Text.length();
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
 start = start +1;
  }
 return F;
```

```
   }


   private static int findMax()
   {
 int max = 0;
 for(int i = 0; i< freq1.F.length; i++)
 if (freq1.F[i] > max ) max = freq1.F[i];
 return max;
   }


}//End of main class I80 nGramVisor
```

81 ***Exercise.*** *Adapt the code to your local circumstances and run it for values of n = 3,2,1,4,5. Adjust consequently the size of pixels and the number of pixels per row. Change to different species. Try out other color encodings according to your knowledge of the visual force of color.*

82 ***Exercise.*** *Our previous Applet breaks down for n = 5 because produced graphic is larger than a physical display. This trouble can be remedied if one uses a JFrame with a JScrollPane instead of an Applet. Make a research on Internet and try that out.*

83 ***Challenge.*** *Verify that our approach to memory management breaks down for n = 6. Redesign our program nGramVisor2 to circumvent this problem. Expect larger execution times.*

84 ***Exercise.*** *Notice that there is large connected holes in our n-grams images. Propose an explanation for this. Test your theories.*

85 ***Exercise.*** *Add to the developed code the possibility to print into the console those n-grams that have frequency 0: this is another form to trace holes. The very same code must be useful to print those n-grams that appear once, twice, ..., and to report a frequency table of frequency ranges.*

We have produced a software to execute a visual n-gram analysis. This is an important task. To make sure that the program does what it must, it is a good idea to add some feedback information - that maybe important by itself.

86 ***Visor with additional information.*** *The next code offers to the User the possibility to see some relevant information about n-grams in our visual analysis.*

```
//Program I86 nGramVisor4
//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content,
//carries an n-gram analysis and
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//might also  be reported
//together with a frequency table of frequency ranges.
//Feedback information is  displayed.
//Scrolling is possible but because of memory demands,
//it works very poorly.



//Fasta .faa format is required.


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;


import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
```

```
public class nGramVisor4 extends JFrame
                          implements MouseListener
{

 private static final long serialVersionUID = 1L;


 //========Fundamental parameters========



//Modify them as it pleases you:

//letters per n-gram
private static  int n = 3;
//Width of pixel
private static int widthOfPixel = 10;
//Number of pixels per row
int pixelsPerRow = 50;
//Width of frame
private static int widthOfFrame = 700;
//Height of frame
private static int heightOfFrame = 700;
//Directory where your files are kept into
private static String nameOfFolder =
"/home/jose/jose/AAjoser/Ciencia/GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */
```

```
//Short .faa alphabet
private static String Alphabet =
                   "ACDEFGHIKLMNPQRSTVWY";

private static int rawFreqTable[] = new int[100];
private static int freqTable[] = new int[100];
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;

private JLabel barMessage;


private  static nGramVisor4 x =  new nGramVisor4();

//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public nGramVisor4()
{
 //Title
 //super("n-gram visor");
 JFrame frame = new JFrame("n-gram visor");
 //Enabling exit by clicking the terminator icon
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setSize(widthOfFrame, heightOfFrame);

 frame.setVisible(true);
 barMessage=new JLabel( "Right-click over a given pixel" );
 frame.add( barMessage, BorderLayout.SOUTH );
 JScrollPane js = new JScrollPane(new Painter());
 //Inner painting class is added

 js.addMouseListener(this);
 frame.getContentPane().add(js);
 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
```

```
         + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);

}



public static void main(String[] args)
 {
//Nothing as yet
 }




//========== INNER DRAWING CLASS ==============


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

 //Constructor = initialization
 public Painter ()
  {

   //setSize(getPreferredSize());
   Painter.this.setBackground(Color.white);
  }


 //Size of the drawing area
 public Dimension getPreferredSize()
 {
  return new Dimension(widthOfFrame, heightOfFrame);
 }
```

```
 //Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.


 public void paintComponent(Graphics g)
 {

  Graphics2D g2d = (Graphics2D) g;
  try
   {
    //n-gram frequencies are found and kept in freq1
findFrequencies();
   }
  catch (IOException e)
   {
e.printStackTrace();
   }

 int max = findMax();
 System.out.println("Frequency scale goes from 0 to " + max);
 System.out.println("Code for intensity from less to more frequency" +
": \n black (absence), white (freq = 1), gray, blue, cyan, yelow," +
" orange, pink, red");
 int nPoints = freq1.F.length;
 boolean go = true;
 int nRows = nPoints / pixelsPerRow +1;
 int counter = 0;
 for(int i = 0; i < 100; i++)
 rawFreqTable[i] = 0;

 for (int j = 0; (j < nRows) & go; j++)
  {
   for (int i = 0; (i < pixelsPerRow) & go; i++)
    {
     int index = j*pixelsPerRow + i;
     if ( freq1.F[index] ==  0)
      {
```

```
 g2d.setColor(Color.BLACK);
 rawFreqTable[0]++;
 /*System.out.println(index + "   "
           + indexInverse(n, index));
           */
 titlesFreqTable[0] = "freq = 0";
 }

if ( freq1.F[index] ==  1)
{
 g2d.setColor(Color.WHITE);
 rawFreqTable[1]++;
 /*System.out.println(index + "   "
          + indexInverse(n, index));
          */
titlesFreqTable[1] = "freq = 1";
}

//Colors are acquired as frequency increases
if ( freq1.F[index] >  1)
 {
   g2d.setColor(Color.GRAY);
  rawFreqTable[2]++;
 titlesFreqTable[2] = " 1 < freq <= " +  max/7;
 }
if ( freq1.F[index] >  max/7)
 {
   g2d.setColor(Color.BLUE);
   rawFreqTable[3]++;
titlesFreqTable[3] = max/7 + " < freq <= " +  2*max/7;
 }
if ( freq1.F[index] >  2* max/7)
 {
g2d.setColor(Color.CYAN);
rawFreqTable[4]++;
titlesFreqTable[4] =  2*max/7 + " < freq <= " +  3*max/7;
 }
if ( freq1.F[index] >  3* max/7)
 {
g2d.setColor(Color.YELLOW);
```

```
     rawFreqTable[5]++;
    titlesFreqTable[5] =   3*max/7 + "  < freq <= " + 4* max/7;
      }
     if ( freq1.F[index] >  4* max/7)
      {
     g2d.setColor(Color.ORANGE);
     rawFreqTable[6]++;
    titlesFreqTable[6] =  4*max/7 + "  < freq <= " +  5*max/7;
      }
     if ( freq1.F[index] >  5* max/7)
      {
     g2d.setColor(Color.PINK);
    rawFreqTable[7]++;
    titlesFreqTable[7] =  5*max/7 + " < freq <= " +  6*max/7;

      }
     if ( freq1.F[index] >  6* max/7)
      {
     g2d.setColor(Color.RED);
     rawFreqTable[8]++;
    titlesFreqTable[8] = 6*max/7 + "  < freq <= " +  max;
      }
     //Pixel is drawn
     g2d.fillOval( i*widthOfPixel,  j*widthOfPixel,
      widthOfPixel, widthOfPixel );
     if (counter >= nPoints-1) go = false;
     counter = counter+1;
     //System.out.println("index = " + index );
    }
   }

 /*
  System.out.println("rawFrequency table ");
  for(int j = 0; j < 9; j++)
   System.out.println(j + "   " + rawFreqTable[j]);
  */
  freqTable[0] = rawFreqTable[0];
  freqTable[1] = rawFreqTable[1];

  for(int j = 2; j < 8; j++)
```

```
  freqTable[j] =
       rawFreqTable[j] - rawFreqTable[j+1];
  freqTable[8] = rawFreqTable[8];

  System.out.println("Frequency table of " + " " + n +  "-grams");
  for(int j = 0; j < 9; j++)
   System.out.println(j +  " " + freqTable[j]
    + " for range  " +  titlesFreqTable[j]);

  //Test
  int sumNGrams = 0;
  for(int j = 0; j < 9; j++)
    sumNGrams = sumNGrams + freqTable[j];
  System.out.println("Sum =  " + sumNGrams + " " + n +  "-grams");


}//End of paintComponent




}//End of Painter

//======== End of inner class Painter =======




 //======== Operative part ============


  //Machine for reading from hard disc
  private static  Scanner input;

  //Path to file
  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer proteome =
                       new StringBuffer("");
```

```
  //length of the Alphabet
  private static  int lengthAlf = Alphabet.length();

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



  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector10
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector10( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector


//========End of inner class intVector======
```

```
private static intVector10  freq1
                                = x.new intVector10();



public static void findFrequencies() throws IOException
{
 readData(path);
 proteome = readData(path);
 System.out.println("length of proteome = "
        + proteome.length());
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  {
 System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
               + "\t" + freq1.F[i]);
  }
 findMax();
}


   // We read  record from file
   public  static  StringBuffer readData(String path)
                        throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
```

```
      {
      //Filtering of first line of each subsequence
      if (s.charAt(0) != '>' )
    {
      //System.out.println(s);
      text = text.append(s);
    }
      }

      fr.close();
     }
     catch ( NoSuchElementException elementException )
     {
        System.err.println( "File is corrupted." );
        input.close();
        System.exit( 1 );
     }
     catch ( IllegalStateException stateException )
     {
        System.err.println( "Reading aborted." );
        System.exit( 1 );
     }
     return text;
}


//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;
  System.out.println("Alien : " + p);
 }
  }
 if (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

//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 ;

  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                   " index = "   + ind);
 return ngram;
}


/*
  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 intVector10 nGramsFrequency(int n,
                         StringBuffer Text)
{
 //Zeroed initialization
 intVector10 F = x.new intVector10();
 int nText = Text.length();
 System.out.println("Length  of text = " + nText);
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
   start = start +1;
    }
   return F;
  }


  private static int findMax()
  {
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
  }


//==========Mouse methods==================


 public void mouseClicked( MouseEvent event )
 {
  int i = (int) Math.floor(event.getX()/widthOfPixel);
  int j = (int) Math.floor(event.getY()/widthOfPixel);
  int index = j*pixelsPerRow + i;
  int freq = freq1.F[index];
  String nGram = indexInverse(n,index);
  /*
  freq1.F[index] = 0;
  g2d.fillOval( j*widthOfPixel,  i*widthOfPixel,
   widthOfPixel, widthOfPixel );*/
  barMessage.setText( String.format(
              "Clicked  at [%d, %d]," +
              " index = %d, freq = %d, n-gram = %s",
               i, j, index,freq,nGram) );
 }

 public void mousePressed( MouseEvent event ){  }
 public void mouseReleased( MouseEvent event ){  }
 public void mouseEntered( MouseEvent event ){  }
```

```
  public void mouseMoved( MouseEvent event ){  }
  public void mouseDragged( MouseEvent event ){  }
  public void mouseExited( MouseEvent event ){  }


}//End of main class I86  nGramVisor4
```

87 ***Exercise.*** *Modify the code to enable the written presentation of the color of the clicked pixel. This is important when n is large (greater than 3).*

88 ***Shooting memory problems.*** *An n-gram analysis produces memory pressing and overflows. Graphic scrolling gets unsatisfactory even for very low values of n. There are two exits from this problem. First: changing the order of the listed amino acids that form the alphabet. In that way the first and upper part of the graphic presents different slides of the overall picture. One always can do this by hand. Second: cutting the graphic into pieces that fit the display without scrolling and that are presented one after the other as requested by the User. We will implement the second idea in a moment.*

89 ***Interlude.*** *We usually hide the many trials that we do before getting an acceptable code. So, we proceed just as nature: living beings are exceedingly complex but at the same time they look perfect and no trace of the infinitely many imperfect trials expected under the Darwinian Evolutionary Theory are found. That is why the Evolutionary Theory is obviously false. To illustrate this, let us present the main steps that were taken to solve the posed problem.*

90 ***Challenge.*** *It is certain that the Darwinian Evolutionary Theory is obviously false. Anyway, some questions might be interesting, say: all our programs have been composed with an aim in mind. But the Darwinian Evolutionary Theory presents evolution as a mindless software developer that lacks an aim (this is correct). So, can we apply our claim to a mindless world and if the answer is affirmative, how? Invitation: compare your answer with ours that is somewhere in Vol II.*

91 ***No go.*** *Our aim is to cut a great graphic into pieces that could be shown without scrolling while offering the possibility to navigate along slides. So, we need two buttons, one for the next page and the other for the previous page. It happened that we imagine the solution to be simple and tried out the next proposal:*

```
//Program I91 nGramVisor6

//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content in various slides,
//carries an n-gram analysis and
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//might also  be reported
//together with a frequency table of frequency ranges.
//Feedback information is  displayed.
//Scrolling is possible but because of memory demands,
//it works very poorly.
//The color of the clicked pixel can be read
//at the bottom bar.

//Fasta .faa format is required.

//This version is a proposal to implement the next task:
//to cut a great graphic into pieces that could be
//shown without scrolling while offering the
//possibility to navigate along slides.

//So, we add two buttons, one for the next page and
//the other for the previous page.

//The program does not work:
//Activate instructions in line 212 and 228
//and verify that they cause a bug.



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Graphics;
```

```
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;


import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;


public class nGramVisor6 extends JFrame
                          implements MouseListener
{

 private static final long serialVersionUID = 1L;


 //========Fundamental parameters========



//Modify them as it pleases you:

//letters per n-gram
private static  int n = 3;
//Width of pixel
private static int widthOfPixel = 10;
```

```
//Number of pixels per row
private static int pixelsPerRow = 50;
private static int pixelsPerPage;

private static int rowsPerPage;
private static int pageCounter;

private static int counter;
private static int nPoints;


//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 1000;

//Number of pixels  per page in the vertical direction
private static int usefulPageHeight = 800;

//Directory where your files are kept into
private static String nameOfFolder =
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                   "ACDEFGHIKLMNPQRSTVWY";
```

```
private static int freqTable[] = new int[100];
private static int max = 0;
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;

private JLabel barMessage;

//Initialization of  color variables

private static Color OurColors[ ] = {Color.BLACK,
Color.WHITE, Color.GRAY, Color.BLUE, Color.CYAN,
Color.YELLOW, Color.ORANGE, Color.PINK, Color.RED};
private static String NamesColors[ ] = {"BLACK",
"WHITE", "GRAY", "BLUE", "CYAN",
"YELLOW", "ORANGE", "PINK", "RED"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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

private static boolean counting = true;
private static  int numberOfpages =0;
private static boolean go = true;

private static JButton nextPage =
                   new JButton("Next page");
private static JButton previousPage =
new JButton("Previous page");

//Instantiation of main class
private  static nGramVisor6 x =  new nGramVisor6();



//======= End of fundamental parameters
```

```
//Constructor of main class:
//Primary instructions for initialization
public nGramVisor6()
{
 //Title
 //super("n-gram visor");
 JFrame frame = new JFrame("n-gram visor");
 //Enabling exit by clicking the terminator icon
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setSize(widthOfFrame, heightOfFrame);

 frame.setVisible(true);
 barMessage=new JLabel( "Right-click over a given pixel" );
 frame.add( barMessage, BorderLayout.SOUTH );
 JScrollPane js = new JScrollPane(new Painter());
 js.addMouseListener(this);
 //Inner painting class is added


 //Buttons go here
 JPanel rightPanel = new JPanel();
 rightPanel.setLayout(new BoxLayout(rightPanel,
         BoxLayout.PAGE_AXIS));

 nextPage.setVisible(true);
 previousPage.setVisible(true);
 rightPanel.add(previousPage);
 rightPanel.add(nextPage);
 rightPanel.add(js);

// frame.getContentPane().add(js);

 frame.getContentPane().add(rightPanel);


 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
         + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);
```

```
 //Button previousPage
 //
 previousPage.addActionListener(new ActionListener()
 {


public void actionPerformed(ActionEvent e)
{
pageCounter--;
System.out.println("PPPP");

//Activate the next line to see an error
//Painter.repaint();

}
 });


 //Button nextPage
 //
 nextPage.addActionListener(new ActionListener()
 {
 public void actionPerformed(ActionEvent e)
 {
  pageCounter++;
  System.out.println("NNNN");

  //Activate the next line to see an error
  //Painter.repaint();

 }
 });

}//End of constructor of nGramVisor6()



//Specifications for the frequency table
public static void definitions()
```

```
{
titlesFreqTable[0] = "freq = 0";
titlesFreqTable[1] = "freq = 1";
titlesFreqTable[2] = " 1 < freq < " +  max/7;
titlesFreqTable[3] = max/7 + " <= freq < " +  2*max/7;
titlesFreqTable[4] = 2*max/7 + " <= freq < " +  3*max/7;
titlesFreqTable[5] = 3*max/7 + "  <= freq < " + 4* max/7;
titlesFreqTable[6] = 4*max/7 + "  <= freq < " +  5*max/7;
titlesFreqTable[7] = 5 *max/7 + " <= freq < " +  6*max/7;
titlesFreqTable[8] = 6*max/7 + "  <= freq <= " +  max;
}


//Assigns a frequency range to the n-gram
//given by index.
//Builds the frequency table by ranges.
//counting is a boolean flag for counting
public static int classification(int index)
{
int rangeNumber = -1;
//Colors are assigned according to frequency
if ( freq1.F[index] ==  0)
{

 if (counting) freqTable[0]++;
      /*System.out.println(index + "   "
                  + indexInverse(n, index));
                  */
 rangeNumber = 0;

}

if ( freq1.F[index] ==  1)
{
  if (counting) freqTable[1]++;
      /*System.out.println(index + "   "
                 + indexInverse(n, index));
                 */
  rangeNumber = 1;
}
```

```
if (( freq1.F[index] >  1)  &
               ( freq1.F[index] <  max/7))
 {
  if (counting) freqTable[2]++;
  rangeNumber = 2;
  }
if (( freq1.F[index] >=  max/7)  &
               ( freq1.F[index] <  2* max/7))
  {
  if (counting) freqTable[3]++;
   rangeNumber = 3;
  }
if (( freq1.F[index] >= 2* max/7)  &
               ( freq1.F[index]  < 3 * max/7))
 {
  if (counting) freqTable[4]++;
  rangeNumber = 4;
 }
if (( freq1.F[index] >=  3*max/7)  &
              ( freq1.F[index] <  4* max/7))
 {
  if (counting) freqTable[5]++;
  rangeNumber = 5;
 }
if (( freq1.F[index] >=  4*max/7)  &
              ( freq1.F[index] <  5* max/7))
 {
  if (counting) freqTable[6]++;
  rangeNumber = 6;
 }
if (( freq1.F[index] >= 5* max/7)  &
  ( freq1.F[index] <  6* max/7))
  {
  if (counting) freqTable[7]++;
  rangeNumber = 7;
  }
if (( freq1.F[index] >=  6*max/7)  &
          ( freq1.F[index] <=    max))
```

```
  {
  if (counting) freqTable[8]++;
   rangeNumber = 8;
  }

return rangeNumber;
}



public static void report()
{
System.out.println("Frequency table of " + " " + n
                                        +  "-grams");
for(int j = 0; j < 9; j++)
   System.out.println(j +  " " + freqTable[j]
    + " for range  " +  titlesFreqTable[j]
     + "  " + NamesColors[j]);
//Test
int sumNGrams = 0;
for(int j = 0; j < 9; j++)
    sumNGrams = sumNGrams + freqTable[j];
System.out.println("Sum =  " + sumNGrams + " " + n
                                      +  "-grams");

}


//Frequencies are grouped in ranges
public static void rangeTable()
{
 try
   {
    //n-gram frequencies are found and kept in freq1
findFrequencies();
   }
  catch (IOException e)
   {
e.printStackTrace();
   }
```

```
 max = findMax();
 definitions();
 System.out.println("Frequency scale goes from 0 to "
     + max + "\nCode for intensity from less to" +
  " more frequency: \n black (absence), " +
  "white (freq = 1), gray, blue, cyan, yelow," +
" orange, pink, red");

 nPoints = freq1.F.length;

 for (int index = 0; (index < nPoints) & go; index++)
  {
   int range = classification(index);
   if (print1)
   System.out.println("index = " + index  + " range = "
     + range  + "Frequency " + freq1.F[index]);

  }
 report();

}


//The main method is necessary
public static void main(String[] args)
 {
//Nothing as yet, methods are invoked by
//paintComponent(Graphics g)


 }




//========== INNER DRAWING CLASS ==============


static class Painter extends JPanel
{
```

```
 private static final long serialVersionUID = 1L;

 //Constructor = initialization
 public Painter ()
  {

   //setSize(getPreferredSize());
   Painter.this.setBackground(Color.white);
   revalidate();
   //A cursor is chosen
   setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
  }


 //Size of the drawing area
 public Dimension getPreferredSize()
 {
  return new Dimension(widthOfFrame, heightOfFrame);
 }


 public static void drawPage(Graphics g2d, int pageCounter)
 {
counter = pageCounter*pixelsPerPage;
System.out.println("counter at drawpage= " + counter);
 for (int j = 0; (j < rowsPerPage) & go; j++)
 {
  for (int i = 0; (i < pixelsPerRow) & go; i++)
   {
    int index = pageCounter *pixelsPerPage +
           j*pixelsPerRow + i;
    int range = classification(index);
    if (print1)
    System.out.println("index = " + index  + " range = "
    + range  + "Frequency " + freq1.F[index]);

    g2d.setColor(OurColors[range]);
    //Pixel is drawn
    g2d.fillOval( i*widthOfPixel, j*widthOfPixel,
```

```
     widthOfPixel, widthOfPixel );
    if (counter >= nPoints-1) go = false;
    counter = counter+1;
   }
  }
 }

//Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.
 @Override
 protected void paintComponent(Graphics g)
 {
     super.paintComponent(g);

  Graphics2D g2d = (Graphics2D) g;
  rangeTable();
//Counting must be done just once
counting = false;
 //painting
 rowsPerPage = usefulPageHeight/widthOfPixel;
 pixelsPerPage = rowsPerPage * pixelsPerRow;
 nPoints = freq1.F.length;
 numberOfpages = nPoints/(pixelsPerPage);
 int counter = 0;
 int pageCounter = 1;

 drawPage(g2d,  pageCounter);

 System.out.println("counter = " + counter);
 counter = 0;
 //pageCounter = 0;
}//End of paintComponent



}//End of Painter
```

```
//======== End of inner class Painter =======



 //======== Operative part ============


  //Machine for reading from hard disc
  private static  Scanner input;

  //Path to file
  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer proteome =
                        new StringBuffer("");









  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector10
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector10( )
  {
```

```
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector


//========End of inner class intVector======




private static intVector10  freq1
                                = x.new intVector10();




public static void findFrequencies() throws IOException
{
 readData(path);
 proteome = readData(path);
 System.out.println("length of proteome = "
        + proteome.length());
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  {
 System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
               + "\t" + freq1.F[i]);
  }
 findMax();
}


   // We read  record from file
```

```
   public  static  StringBuffer readData(String path)
                        throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
       {
       //Filtering of first line of each subsequence
       if (s.charAt(0) != '>' )
     {
       //System.out.println(s);
       text = text.append(s);
     }
       }

       fr.close();
      }
      catch ( NoSuchElementException elementException )
      {
         System.err.println( "File is corrupted." );
         input.close();
         System.exit( 1 );
      }
      catch ( IllegalStateException stateException )
      {
         System.err.println( "Reading aborted." );
         System.exit( 1 );
      }
      return text;
   }


   //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;
  System.out.println("Alien : " + p);
 }
  }
 if (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

//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 ;

  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                   " index = "   + ind);
 return ngram;
}


/*
  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 intVector10 nGramsFrequency(int n,
                         StringBuffer Text)
{
 //Zeroed initialization
 intVector10 F = x.new intVector10();
```

```
    int nText = Text.length();
    System.out.println("Length  of text = " + nText);
    int start = 0;
    int ind = 0;
    for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
    start = start +1;
     }
    return F;
   }


   private static int findMax()
   {
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
   }


//==========Mouse methods==================


 public void mouseClicked( MouseEvent event )
 {
  int i = (int) Math.floor(event.getX()/widthOfPixel);
  int j = (int) Math.floor(event.getY()/widthOfPixel);
  int index = j*pixelsPerRow + i;
  int freq = freq1.F[index];
  //counting = false;
  int range = classification(index);
```

```
   String nameOfColor = NamesColors[range];

   String nGram = indexInverse(n,index);
   barMessage.setText( String.format(
           "Clicked  at [%d, %d]," +
               "Page number " + "page" +
           " index = %d, freq = %d, n-gram = %s, %s",
           i, j, index, freq, nGram, nameOfColor) );
  }

  public void mousePressed( MouseEvent event ){  }
  public void mouseReleased( MouseEvent event ){  }
  public void mouseEntered( MouseEvent event ){  }
  public void mouseMoved( MouseEvent event ){  }
  public void mouseDragged( MouseEvent event ){  }
  public void mouseExited( MouseEvent event ){  }


}//End of main class I91 nGramVisor6
```

92 ***Exercise.*** *Verify that the previous program runs in spite of two errors, that the two buttons are visible, that the listeners hear normally, but that the execution is not done because of a call to the method repaint() that cannot be processed and that cause fatal errors.*

93 ***Help from Internet.*** *Since we found ourselves in a blind alley, we decided to look at Internet for a solution to our problem: the method repaint() cannot be invoked because a trouble with inner classes. No direct solution was found but the next code offered a hope:*

```
//Program I93 GraphPanel
//This program shows how to control repaint()
//with a button.
/*By trashgod
http://stackoverflow.com/questions/4282159
Retrieved 1/V/2013
*/
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;


public class GraphPanel extends JPanel {


private static final long serialVersionUID = 1L;
private boolean colorFlag;

    public GraphPanel() {
        this.setPreferredSize(new Dimension(640, 480));
    }

    public void toggle() {
        colorFlag = !colorFlag;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        if (colorFlag) {
            g2.setColor(Color.red);
        } else {
            g2.setColor(Color.blue);
        }
```

```
        g2.drawLine(0, 0, getWidth(), getHeight());
    }

    private void display() {
        JFrame f = new JFrame("GraphPanel");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(this, BorderLayout.CENTER);
        f.add(new ControlPanel(this), BorderLayout.SOUTH);
        f.pack();
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                new GraphPanel().display();
            }
        });
    }
}

class ControlPanel extends JPanel {

    public ControlPanel(final GraphPanel gp) {
        this.add(new JButton(new AbstractAction("Click") {

            @Override
            public void actionPerformed(ActionEvent e) {
                gp.toggle();
                gp.repaint();
            }
        }));
    }
}//End of main class I93 GraphPanel
```

94 ***Exercise.*** *Run the previous code. Use Eclipse to correct the warnings.*

95 ***Two buttons.*** *The code from Internet was pretty good to show how to solve*

our problem allowing to control repaint() from a button. But we need two buttons, one for previous page and the other for the next page. Can we modify this code to include two buttons? Our proposal follows:

```
//Program I95 TwoButtons
//This program shows how to control repaint()
//with two buttons.

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;


public class TwoButtons extends JPanel
 {
  private static final long serialVersionUID = 1L;
private boolean colorFlag;

    public TwoButtons()
     {
        this.setPreferredSize(new Dimension(640, 480));
     }

    public void toggle()
     {
       colorFlag = !colorFlag;
     }

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

```
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        if (colorFlag) {
            g2.setColor(Color.red);
        } else {
            g2.setColor(Color.blue);
        }
        g2.drawLine(0, 0, getWidth(), getHeight());
    }

    private void display()
    {
      JFrame f = new JFrame("twoButtons");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(this, BorderLayout.CENTER);
        f.add(new ControlPanel1(this), BorderLayout.SOUTH);
        //f.add(new ControlPanel2(this), BorderLayout.SOUTH);
        f.pack();
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                new TwoButtons().display();
            }
        });
    }
}


class ControlPanel1 extends JPanel {

    /**
 *
 */
```

```
private static final long serialVersionUID = 1L;

public ControlPanel1(final TwoButtons gp)
{
        this.add
        (new JButton
        (new AbstractAction("Click")
         {

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

@Override
            public void actionPerformed(ActionEvent e)
{
                gp.toggle();
                gp.repaint();
            }
          }
        ));

        this.add
        (new JButton
        (new AbstractAction("Clpppck")
         {

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

@Override
            public void actionPerformed(ActionEvent e)
{
                gp.toggle();
                gp.repaint();
            }
          }
```

```
        ));
    }

}//End of main class I95 TwoButtons
```

96 ***Exercise.*** *Run the previous program. Play with the code, say, add a third button or rename some subunits.*

97 ***Exercise.*** *Merge the idea of Internet with our code in nGramVisor6 to see whether or not we have a solution although imperfect. You are expected to suffer a lot, i.e., you will produce many unviable programs and many others that are too crude to offer a hope. At a given moment decide to stop suffering and ask yourself: What shall I say to the mania of modern science to claim that Darwinian Evolution is the unquestionable explanation to every event of the so called parallel gene transfer?*

98 ***Merging.*** *Let us try to achieve a full fledged program that merges the idea of Internet with our code. One of the problems we shall solve is the following: an abstract button is one that needs no name because it receives no call. But when one needs to control the behavior of a button, say, to make it invisible, one needs to call on its name, an so it needs one. Another problem arises when we decide to conserve, as suggested by Internet assistance, the thread technology: in that case, everything that can be done in parallel, will be done in parallel. The result is that the output breaks the expected order of executing tasks and so it appears damaged. To remedy this trouble, one can try re-positioning of instructions or to turn from global variables to the function paradigm: things that must be concatenated serially, must be tied by explicit parameter dependence.*

```
//Program I98 nGramVisor9


//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content in various slides,
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//or with just one event,
//might also  be reported
//together with a frequency table of frequency ranges.
```

```
//Feedback information is  displayed.

//The color of the clicked pixel can be read
//at the SOUTH.

//To overcome troubles with too heavy graphics,
//we cut the  great graphic into pieces that could be
//shown without scrolling while offering the
//possibility to navigate along slides.

//So, we add two buttons, one for the next page and
//the other for the previous page.


//Fasta .faa format is required.



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
```

```
import javax.swing.JPanel;



public class nGramVisor9 extends JPanel
 {
  private static final long serialVersionUID = 1L;

//========Fundamental parameters========



//Modify them as it pleases you:

//letters per n-gram
private static  int n = 3;
//Width of pixel
private static int widthOfPixel = 10;
//Number of pixels per row
private static int pixelsPerRow = 50;
private static int pixelsPerPage;

private static int rowsPerPage;
private static int page;

private static int shownPixels;
private static int nPoints;


//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 800;

//Number of pixels  per page in the vertical direction
private static int usefulPageHeight = 450;

//Directory where your files are kept into
private static String nameOfFolder =
```

```
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                  "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                  "ACDEFGHIKLMNPQRSTVWY";

private static int freqTable[] = new int[100];
private static int max = 0;
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;



//Initialization of  color variables

private static Color OurColors[ ] = {Color.BLACK,
Color.WHITE, Color.GRAY, Color.BLUE, Color.CYAN,
Color.YELLOW, Color.ORANGE, Color.PINK, Color.RED};
private static String NamesColors[ ] = {"BLACK",
"WHITE", "GRAY", "BLUE", "CYAN",
"YELLOW", "ORANGE", "PINK", "RED"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();
```

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

private static boolean counting = true;
private static  int numberOfPages =0;
private static boolean go = true;

private static String message = "";

private JLabel label = new JLabel("Click over a pixel");


//Instantiation of main class
private  static nGramVisor9 x =  new nGramVisor9();

JFrame f = new JFrame("n-gram visor");


//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public nGramVisor9()
 {
this.setPreferredSize(
new Dimension(widthOfFrame, heightOfFrame));
this.setBackground(Color.white);
//A cursor is chosen
setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
 }



private void starter()
{
 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
          + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
```

```
 System.out.println("Number of n-grams = " + N);
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.add(this, BorderLayout.BEFORE_FIRST_LINE);
 f.add(new buttonPanel3(this), BorderLayout.SOUTH);
 f.add(label);
 f.pack();
 f.setLocationRelativeTo(null);
 f.setVisible(true);
 MouseWork2 l = new MouseWork2();
 f.addMouseListener(l);
 rangeTable();
}


public static void main(String[] args)
{

  //Thread inauguration: assigns second priority to this
  EventQueue.invokeLater
(new Runnable()
     {
        @Override
        public void run()
        {
            new nGramVisor9().starter();
        }
     }
);
}//End of main




//Draws a page of the great graphic
public static void drawPage(Graphics g2d, int page)
{
 shownPixels = page*pixelsPerPage;
 //Control of buttons
 if (page <= 0)
 {
```

```
  nGramVisor9.buttonPanel3.previousPage.setEnabled(false);
  nGramVisor9.buttonPanel3.nextPage.setEnabled(true);
 }
 if (page >= numberOfPages)
 {
  nGramVisor9.buttonPanel3.previousPage.setEnabled(true);
  nGramVisor9.buttonPanel3.nextPage.setEnabled(false);
 }
 if ((page >= 0) & (page <=numberOfPages)) go = true;
else go = false;

 if (numberOfPages == 0)
  {
   nGramVisor9.buttonPanel3.previousPage.setEnabled(false);
   nGramVisor9.buttonPanel3.nextPage.setEnabled(false);
  }

 //Page is displayed
 for (int j = 0; (j < rowsPerPage) & go; j++)
  {
   for (int i = 0; (i < pixelsPerRow) & go; i++)
{
 int index = page *pixelsPerPage +
           j*pixelsPerRow + i;
 int range = classification(index);
 if (print1)
 System.out.println("index = " + index  + " range = "
    + range  + "Frequency " + freq1.F[index]);

 g2d.setColor(OurColors[range]);
    //Pixel is drawn
 g2d.fillOval( i*widthOfPixel, j*widthOfPixel,
     widthOfPixel, widthOfPixel );
 if (shownPixels >= nPoints-1) go = false;
 shownPixels = shownPixels+1;
}
  }
}
```

```
//Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.
@Override //Repainting is allowed
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
  //Embellishment
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

  drawPage(g2d,  page);

 }


 //Specifications for the frequency table
 public static void definitions()
  {
  titlesFreqTable[0] = "freq = 0";
  titlesFreqTable[1] = "freq = 1";
  titlesFreqTable[2] = " 1 < freq < " +  max/7;
  titlesFreqTable[3] = max/7 + " <= freq < " +  2*max/7;
  titlesFreqTable[4] = 2*max/7 + " <= freq < " +  3*max/7;
  titlesFreqTable[5] = 3*max/7 + "  <= freq < " + 4* max/7;
  titlesFreqTable[6] = 4*max/7 + "  <= freq < " +  5*max/7;
  titlesFreqTable[7] = 5 *max/7 + " <= freq < " +  6*max/7;
  titlesFreqTable[8] = 6*max/7 + "  <= freq <= " +  max;
  }


  //Assigns a frequency range to the n-gram
  //given by index.
  //Builds the frequency table by ranges.
  //counting is a boolean flag for counting
  public static int classification(int index)
  {
```

```
int rangeNumber = -1;
//Colors are assigned according to frequency
if ( freq1.F[index] ==  0)
 {

  if (counting) freqTable[0]++;
       /*System.out.println(index + "   "
                  + indexInverse(n, index));
                  */
  rangeNumber = 0;

 }

if ( freq1.F[index] ==  1)
 {
   if (counting) freqTable[1]++;
       /*System.out.println(index + "   "
                 + indexInverse(n, index));
                 */
   rangeNumber = 1;
 }


if (( freq1.F[index] >  1)  &
                ( freq1.F[index] <  max/7))
  {
   if (counting) freqTable[2]++;
   rangeNumber = 2;
   }
if (( freq1.F[index] >=  max/7)  &
                ( freq1.F[index] <  2* max/7))
   {
   if (counting) freqTable[3]++;
    rangeNumber = 3;
   }
if (( freq1.F[index] >= 2* max/7)  &
                ( freq1.F[index]  < 3 * max/7))
  {
   if (counting) freqTable[4]++;
   rangeNumber = 4;
```

```
    }
  if (( freq1.F[index] >=  3*max/7)  &
                ( freq1.F[index] <  4* max/7))
    {
     if (counting) freqTable[5]++;
     rangeNumber = 5;
    }
  if (( freq1.F[index] >=  4*max/7)  &
                ( freq1.F[index] <  5* max/7))
    {
     if (counting) freqTable[6]++;
     rangeNumber = 6;
    }
  if (( freq1.F[index] >= 5* max/7)  &
     ( freq1.F[index] <  6* max/7))
     {
     if (counting) freqTable[7]++;
     rangeNumber = 7;
     }
  if (( freq1.F[index] >=  6*max/7)  &
              ( freq1.F[index] <=    max))
     {
     if (counting) freqTable[8]++;
      rangeNumber = 8;
     }


  return rangeNumber;
  }


//Frequency table is reported
 public static void report(int max)
  {

   System.out.println("Frequency table of " + " " + n
                                          +  "-grams");
   for(int j = 0; j < 9; j++)
      System.out.println(j +  " " + freqTable[j]
       + " for range  " +  titlesFreqTable[j]
```

```
      + "  " + NamesColors[j]);
//Test
int sumNGrams = 0;
for(int j = 0; j < 9; j++)
     sumNGrams = sumNGrams + freqTable[j];
System.out.println("Sum =  " + sumNGrams + " " + n
                                      +  "-grams");

}


//Frequencies are grouped in ranges
public static void rangeTable()
{
  try
    {
     //n-gram frequencies are found and kept in freq1
 findFrequencies();
    }
   catch (IOException e)
    {
 e.printStackTrace();
    }
  max = findMax(freq1);
  definitions();


  System.out.println("Frequency scale goes from 0 to "
      + max + "\nCode for intensity from less to" +
   " more frequency: \n black (absence), " +
   "white (freq = 1), gray, blue, cyan, yelow," +
 " orange, pink, red");

  nPoints = freq1.F.length;

  for (int index = 0; (index < nPoints) & go; index++)
   {
    int range = classification(index);
    if (print1)
    System.out.println("index = " + index  + " range = "
```

```
        + range  + "Frequency " + freq1.F[index]);

     }
     //Counting of frequencies must be done just once
     counting = false;
   report(max);
  }


    //======== Operative part ============


     //Machine for reading from hard disc
     private static  Scanner input;

     //Path to file
     private static  String path = nameOfFolder + nameOfFile;

     private static StringBuffer proteome =
                           new StringBuffer("");


     //======= Inner class intVector ========
   //This inner class defines a vector with
   //integer numbers as a new type
   private  class intVector11
   {
   int L = (int) Math.pow(lengthAlf, n);
   int F[] = new int[L];

   //An instance of intVector can be
   //initialized in various ways:

   //Automatic zeroed initialization
   intVector11( )
     {
      for(int i = 0; i < L; i++)
      F[i] = 0;
     }
```

```
   }//end of class intVector


   //========End of inner class intVector======



    private static intVector11  freq1
    = x.new intVector11();


//Frequencies of n-grams are calculated
public static void findFrequencies() throws IOException
{
proteome = readData(path);
rowsPerPage = usefulPageHeight/widthOfPixel;
pixelsPerPage = rowsPerPage * pixelsPerRow;
nPoints = freq1.F.length;

numberOfPages =  nPoints/pixelsPerPage;


if (numberOfPages * pixelsPerPage == nPoints)
        numberOfPages--;
System.out.println("Number of pages = " + numberOfPages);
System.out.println("length of proteome = "
+ proteome.length());
freq1 = nGramsFrequency(n, proteome);
if (print2)
{
System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
System.out.println(i + "\t" +  indexInverse(n, i)
    + "\t" + freq1.F[i]);
}


}
```

```
// We read  record from file
public  static  StringBuffer readData(String path)
    throws IOException
{
StringBuffer text = new StringBuffer("");
//try and catch gates
try
{
FileReader fr = new FileReader(path);
BufferedReader br = new BufferedReader(fr);
String s = "";

while((s = br.readLine()) != null)
{
//Filtering of first line of each subsequence
if (s.charAt(0) != '>' )
{
//System.out.println(s);
text = text.append(s);
}
}

fr.close();
}
catch ( NoSuchElementException elementException )
{
System.err.println( "File is corrupted." );
input.close();
System.exit( 1 );
}
catch ( IllegalStateException stateException )
{
System.err.println( "Reading aborted." );
System.exit( 1 );
}
return text;
}
```

```
//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;
System.out.println("Alien : " + p);
}
}
if (print1)
System.out.println( "ngram = " + ngram + " index = "
                        + ind);
return ind;
}

//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 ;

}
if (print1)
System.out.println( "ngram = " + ngram +
 " index = "   + ind);
return ngram;
}


/*
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 intVector11 nGramsFrequency(int n,
       StringBuffer Text)
```

```
{
//Zeroed initialization
intVector11 F = x.new intVector11();
int nText = Text.length();
System.out.println("Length  of working text = " + nText);
int start = 0;
int ind = 0;
for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
start = start +1;
}
findMax(freq1);
return F;
}


//Maximal frequency is found
private static int findMax(intVector11 freq1)
{
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
}


//==========Mouse methods=================

public class MouseWork2  implements MouseListener
{
 public void mouseClicked( MouseEvent event )
  {
```

```
   //System.out.println(event.getX()+ " " + event.getY());
   int i = (int) Math.floor(event.getX()/widthOfPixel);
   //28 = Interference  from label
   int j = (int) Math.floor((event.getY()-28)/widthOfPixel);
   int index = page *pixelsPerPage +
      j*pixelsPerRow + i;
   int freq = freq1.F[index];
   //counting = false;
   int range = classification(index);
   String nameOfColor = NamesColors[range];
   String nGram = indexInverse(n,index);
    message = String.format(
          "Clicked  at [%d, %d]," +
              " Page = %d," +
      " index = %d, freq = %d, n-gram = %s, %s",
      i, j, page, index, freq, nGram, nameOfColor);
   label.setText(message);
}

public void mousePressed( MouseEvent event ){  }
public void mouseReleased( MouseEvent event ){  }
public void mouseEntered( MouseEvent event ){  }
public void mouseMoved( MouseEvent event ){  }
public void mouseDragged( MouseEvent event ){  }
public void mouseExited( MouseEvent event ){  }

 }//End of inner class MouseWork


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


//Buttons are implemented
static class buttonPanel3 extends JPanel
{


private static final long serialVersionUID = 1L;
final static JButton previousPage =
                              new JButton("Previous page");
```

```
final static JButton nextPage = new JButton("Next page");
//Constructor
public buttonPanel3(final nGramVisor9 v)
{
        //Button = previous page

this.add(previousPage);
        previousPage.addActionListener(new ActionListener()
        {
          public void actionPerformed(ActionEvent e)
          {
           nGramVisor9.page--;
              v.repaint();
              if (page <= 0)
                {
               previousPage.setEnabled(false);
               nextPage.setEnabled(true);
               go = false;
                }
              else
                {
               previousPage.setEnabled(true);
               go = true;
                }
          }
        });

         //Button = next page

        this.add(nextPage);
       nextPage.addActionListener(new ActionListener()
       {
         public void actionPerformed(ActionEvent e)
         {
          nGramVisor9.page++;
             v.repaint();
             if (page >= numberOfPages)
             {
            nextPage.setEnabled(false);
           previousPage.setEnabled(true);
```

```
            go = false;
             }
           else
             {
            nextPage.setEnabled(true);
            previousPage.setEnabled(true);
            go = true;
             }
        }
      });



    }//End of constructor of buttonPanel3

}//End of inner class buttonPanel3

 }//End of class I 98 nGramVisor9
```

99 ***Exercise.*** *Run the previous program and play with the code. Verify the claim of the Author that it is correctly designed and makes what it must: otherwise, write to the Author.*

100 ***Looking for perfection.*** *Let us notice that the previous program, nGramVisor9, is for 4-grams 5 times slower than the former version nGramVisor5. Let us devise a correct and efficient application. The code follows:*

```
//Program I100 nGramVisora10

//This program reads a  proteome from a file,
//ignores commentaries,
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//or with just one event,
//might also  be reported
//together with a frequency table of frequency ranges.
```

```
//Feedback information is  displayed.

//The color of the clicked pixel can be read
//at the SOUTH.

//To overcome troubles with too heavy graphics,
//we cut the  great graphic into pieces that could be
//shown without scrolling while offering the
//possibility to navigate along slides.

//So, we add two buttons, one for the next page and
//the other for the previous page.


//Fasta .faa format is required.



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
```

```
import javax.swing.JPanel;



public class nGramVisora10 extends JPanel
 {
  private static final long serialVersionUID = 1L;

//========Fundamental parameters========
//Modify them as it pleases you:

//letters per n-gram
private static  int n = 5;
//Width of pixel
private static int widthOfPixel = 1;
//Number of pixels per row
private static int pixelsPerRow = 500;
private static int pixelsPerPage;

private static int rowsPerPage;
private static int page;

private static int shownPixels;
private static int nPoints;


//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 800;

//Number of pixels  per page in the vertical direction
private static int usefulPageHeight = 500;

//Directory where your files are kept into
private static String nameOfFolder =
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
```

```
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                  "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                  "ACDEFGHIKLMNPQRSTVWY";

private static int freqTable[] = new int[100];
private static int max = 0;
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;



//Initialization of  color variables

private static Color OurColors[ ] = {Color.BLACK,
Color.WHITE, Color.GRAY, Color.BLUE, Color.CYAN,
Color.YELLOW, Color.ORANGE, Color.PINK, Color.RED};
private static String NamesColors[ ] = {"BLACK",
"WHITE", "GRAY", "BLUE", "CYAN",
"YELLOW", "ORANGE", "PINK", "RED"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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

```
private static boolean counting = true;
private static  int numberOfPages =0;
private static boolean go = true;

private static String message = "";

private JLabel label = new JLabel("Click over a pixel");


//Instantiation of main class
private  static nGramVisora10 x =  new nGramVisora10();

JFrame f = new JFrame("n-gram visor");


//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public nGramVisora10()
 {
this.setPreferredSize(
new Dimension(widthOfFrame, heightOfFrame));
this.setBackground(Color.white);
//A cursor is chosen
setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
 }



private void starter()
{
 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
          + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.add(this, BorderLayout.BEFORE_FIRST_LINE);
```

```
 f.add(new buttonPanel4(this), BorderLayout.SOUTH);
 f.add(label);
 f.pack();
 f.setLocationRelativeTo(null);
 f.setVisible(true);
 MouseWork2 l = new MouseWork2();
 f.addMouseListener(l);
 rangeTable();
}


public static void main(String[] args)
{

  //Thread inauguration: assigns second priority to this
  EventQueue.invokeLater
(new Runnable()
     {
        @Override
        public void run()
        {
            new nGramVisora10().starter();
        }
     }
);
}//End of main




//Draws a page of the great graphic
public static void drawPage(Graphics g2d, int page)
{
 shownPixels = page*pixelsPerPage;
 //buttons' housekeeping
 if (page <= 0)
 {
  nGramVisora10.buttonPanel4.previousPage.setEnabled(false);
  nGramVisora10.buttonPanel4.nextPage.setEnabled(true);
 }
```

```
 if (page >= numberOfPages)
 {
  nGramVisora10.buttonPanel4.previousPage.setEnabled(true);
  nGramVisora10.buttonPanel4.nextPage.setEnabled(false);
 }
 if ((page >= 0) & (page <=numberOfPages)) go = true;
else go = false;

 if (numberOfPages == 0)
  {
   nGramVisora10.buttonPanel4.previousPage.setEnabled(false);
   nGramVisora10.buttonPanel4.nextPage.setEnabled(false);
  }

 //Page is displayed
 for (int j = 0; (j < rowsPerPage) & go; j++)
  {
   for (int i = 0; (i < pixelsPerRow) & go; i++)
{
 int index = page *pixelsPerPage +
           j*pixelsPerRow + i;
 int range = classification(index);
 if (print1)
 System.out.println("index = " + index  + " range = "
    + range  + "Frequency " + freq1.F[index]);

 g2d.setColor(OurColors[range]);
    //Pixel is drawn
 g2d.fillOval( i*widthOfPixel, j*widthOfPixel,
     widthOfPixel, widthOfPixel );
 if (shownPixels >= nPoints-1) go = false;
 shownPixels = shownPixels+1;
}
  }
}


//Drawing tool
//Second method to be executed and
//then on always ready to get into action.
```

```
//The vector of frequencies is displayed in
//graphic form according to a coloring code.
@Override //Repainting is allowed
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
  //Embellishment =  clogging code
  if (n==3)
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

  drawPage(g2d,  page);

 }


 //Specifications for the frequency table
 public static void definitions()
  {
  titlesFreqTable[0] = "freq = 0";
  titlesFreqTable[1] = "freq = 1";
  titlesFreqTable[2] = " 1 < freq < " +  max/7;
  titlesFreqTable[3] = max/7 + " <= freq < " +  2*max/7;
  titlesFreqTable[4] = 2*max/7 + " <= freq < " +  3*max/7;
  titlesFreqTable[5] = 3*max/7 + "  <= freq < " + 4* max/7;
  titlesFreqTable[6] = 4*max/7 + "  <= freq < " +  5*max/7;
  titlesFreqTable[7] = 5 *max/7 + " <= freq < " +  6*max/7;
  titlesFreqTable[8] = 6*max/7 + "  <= freq <= " +  max;
  }


  //Assigns a frequency range to the n-gram
  //given by index.
  //Builds the frequency table by ranges.
  //counting is a boolean flag for counting
  public static int classification(int index)
  {
  int rangeNumber = -1;
  //Colors are assigned according to frequency
```

```
if ( freq1.F[index] ==  0)
 {

  if (counting) freqTable[0]++;
       /*System.out.println(index + "   "
                  + indexInverse(n, index));
                  */
  rangeNumber = 0;

 }

if ( freq1.F[index] ==  1)
 {
   if (counting) freqTable[1]++;
       /*System.out.println(index + "   "
                 + indexInverse(n, index));
                 */
   rangeNumber = 1;
 }


if (( freq1.F[index] >  1)  &
               ( freq1.F[index] <  max/7))
  {
   if (counting) freqTable[2]++;
   rangeNumber = 2;
   }
if (( freq1.F[index] >=  max/7)  &
               ( freq1.F[index] <  2* max/7))
   {
   if (counting) freqTable[3]++;
    rangeNumber = 3;
   }
if (( freq1.F[index] >= 2* max/7)  &
               ( freq1.F[index]  < 3 * max/7))
  {
   if (counting) freqTable[4]++;
   rangeNumber = 4;
  }
if (( freq1.F[index] >=  3*max/7)  &
```

```
                ( freq1.F[index] <  4* max/7))
    {
     if (counting) freqTable[5]++;
     rangeNumber = 5;
    }
  if (( freq1.F[index] >=  4*max/7)  &
                ( freq1.F[index] <  5* max/7))
    {
     if (counting) freqTable[6]++;
     rangeNumber = 6;
    }
  if (( freq1.F[index] >= 5* max/7)  &
     ( freq1.F[index] <  6* max/7))
     {
     if (counting) freqTable[7]++;
     rangeNumber = 7;
     }
  if (( freq1.F[index] >=  6*max/7)  &
              ( freq1.F[index] <=    max))
     {
     if (counting) freqTable[8]++;
      rangeNumber = 8;
     }


  return rangeNumber;
  }


//Frequency table is reported
 public static void report(int max)
  {

   System.out.println("Frequency table of " + " " + n
                                          +  "-grams");
   for(int j = 0; j < 9; j++)
      System.out.println(j +  " " + freqTable[j]
       + " for range  " +  titlesFreqTable[j]
        + "  " + NamesColors[j]);
  //Test
```

```
int sumNGrams = 0;
for(int j = 0; j < 9; j++)
     sumNGrams = sumNGrams + freqTable[j];
System.out.println("Sum =  " + sumNGrams + " " + n
                                      +  "-grams");

}


//Frequencies are grouped in ranges
public static void rangeTable()
{
  try
    {
     //n-gram frequencies are found and kept in freq1
 findFrequencies();
    }
   catch (IOException e)
    {
 e.printStackTrace();
    }
  max = findMax(freq1);
  definitions();


  System.out.println("Frequency scale goes from 0 to "
      + max + "\nCode for intensity from less to" +
   " more frequency: \n black (absence), " +
   "white (freq = 1), gray, blue, cyan, yelow," +
 " orange, pink, red");

  nPoints = freq1.F.length;

  for (int index = 0; (index < nPoints) & go; index++)
   {
    int range = classification(index);
    if (print1)
    System.out.println("index = " + index  + " range = "
      + range  + "Frequency " + freq1.F[index]);
```

```
    }
    //Counting of frequencies must be done just once
    counting = false;
  report(max);
 }


  //======== Operative part ============


    //Machine for reading from hard disc
    private static  Scanner input;

    //Path to file
    private static  String path = nameOfFolder + nameOfFile;

    private static StringBuffer proteome =
                           new StringBuffer("");


    //======= Inner class intVector ========
  //This inner class defines a vector with
  //integer numbers as a new type
  private  class intVector11
  {
  int L = (int) Math.pow(lengthAlf, n);
  int F[] = new int[L];

  //An instance of intVector can be
  //initialized in various ways:

  //Automatic zeroed initialization
  intVector11( )
    {
     for(int i = 0; i < L; i++)
     F[i] = 0;
    }


  }//end of class intVector
```

```
   //========End of inner class intVector======


    private static intVector11  freq1
    = x.new intVector11();


//Frequencies of n-grams are calculated
public static void findFrequencies() throws IOException
{
proteome = readData(path);
rowsPerPage = usefulPageHeight/widthOfPixel;
pixelsPerPage = rowsPerPage * pixelsPerRow;
nPoints = freq1.F.length;

numberOfPages =  nPoints/pixelsPerPage;


if (numberOfPages * pixelsPerPage == nPoints)
        numberOfPages--;
System.out.println("Number of pages = " + numberOfPages);
System.out.println("length of proteome = "
+ proteome.length());
freq1 = nGramsFrequency(n, proteome);
if (print2)
{
System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
System.out.println(i + "\t" +  indexInverse(n, i)
    + "\t" + freq1.F[i]);
}


}
```

```
// We read  record from file
public  static  StringBuffer readData(String path)
    throws IOException
{
StringBuffer text = new StringBuffer("");
//try and catch gates
try
{
FileReader fr = new FileReader(path);
BufferedReader br = new BufferedReader(fr);
String s = "";

while((s = br.readLine()) != null)
{
//Filtering of first line of each subsequence
if (s.charAt(0) != '>' )
{
//System.out.println(s);
text = text.append(s);
}
}

fr.close();
}
catch ( NoSuchElementException elementException )
{
System.err.println( "File is corrupted." );
input.close();
System.exit( 1 );
}
catch ( IllegalStateException stateException )
{
System.err.println( "Reading aborted." );
System.exit( 1 );
}
return text;
}


//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;
System.out.println("Alien : " + p);
}
}
if (print1)
System.out.println( "ngram = " + ngram + " index = "
                        + ind);
return ind;
}

//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 ;

}
if (print1)
System.out.println( "ngram = " + ngram +
 " index = "   + ind);
return ngram;
}


/*
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 intVector11 nGramsFrequency(int n,
       StringBuffer Text)
{
//Zeroed initialization
```

```
intVector11 F = x.new intVector11();
int nText = Text.length();
System.out.println("Length  of working text = " + nText);
int start = 0;
int ind = 0;
for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
start = start +1;
}
findMax(freq1);
return F;
}


//Maximal frequency is found
private static int findMax(intVector11 freq1)
{
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
}


//==========Mouse methods==================

public class MouseWork2  implements MouseListener
{
 public void mouseClicked( MouseEvent event )
  {
   //System.out.println(event.getX()+ " " + event.getY());
   int i = (int) Math.floor(event.getX()/widthOfPixel);
```

```
   //28 = Interference  from label
   int j = (int) Math.floor((event.getY()-28)/widthOfPixel);
   int index = page *pixelsPerPage +
      j*pixelsPerRow + i;
   int freq = freq1.F[index];
   //counting = false;
   int range = classification(index);
   String nameOfColor = NamesColors[range];
   String nGram = indexInverse(n,index);
    message = String.format(
          "Clicked  at [%d, %d]," +
              " Page = %d," +
      " index = %d, freq = %d, n-gram = %s, %s",
      i, j, page, index, freq, nGram, nameOfColor);
   label.setText(message);
}

public void mousePressed( MouseEvent event ){  }
public void mouseReleased( MouseEvent event ){  }
public void mouseEntered( MouseEvent event ){  }
public void mouseMoved( MouseEvent event ){  }
public void mouseDragged( MouseEvent event ){  }
public void mouseExited( MouseEvent event ){  }

 }//End of inner class MouseWork


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


//Buttons are implemented
static class buttonPanel4 extends JPanel
{


private static final long serialVersionUID = 1L;
final static JButton previousPage =
                              new JButton("Previous page");
final static JButton nextPage = new JButton("Next page");
//Constructor
```

```
public buttonPanel4(final nGramVisora10 v)
{
        //Button = previous page

this.add(previousPage);
        previousPage.addActionListener(new ActionListener()
        {
          public void actionPerformed(ActionEvent e)
          {
           nGramVisora10.page--;
              v.repaint();
              if (page <= 0)
                {
               previousPage.setEnabled(false);
               nextPage.setEnabled(true);
               go = false;
                }
              else
                {
               previousPage.setEnabled(true);
               go = true;
                }
          }
        });

         //Button = next page

        this.add(nextPage);
       nextPage.addActionListener(new ActionListener()
       {
         public void actionPerformed(ActionEvent e)
         {
          nGramVisora10.page++;
             v.repaint();
             if (page >= numberOfPages)
             {
            nextPage.setEnabled(false);
           previousPage.setEnabled(true);
            go = false;
             }
```

```
          else
            {
           nextPage.setEnabled(true);
           previousPage.setEnabled(true);
           go = true;
            }
        }
      });
```

```
    }//End of constructor of buttonPanel3

}//End of inner class buttonPanel3

 }//End of class I100 nGramVisora10
```

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

4.3 Proteomes synthesized at random

Why proteomes are as they are? The first theory that we must consider is that they exist because they are the result of random concatenation of letters from a given alphabet.

102 ***Exercise.*** *Add to the previous program nGramVisora10 the possibility to synthesize a proteome with letters concatenated at random.*

103 ***The advantage of relative frequencies.***

We have witnessed that working with absolute frequencies of n-grams causes table frequencies and n-gram charts to depend on the length of synthesized text. This flaw must be remedied with the introduction of relative frequencies. But we have a problem: must we divide the absolute frequency of a given index by the possible number of n-grams or by the length of the analyzed proteome or by what.

To get independence of the length of analyzed text or proteome we must divide by the number of n-grams contained in the text. We immediately get that relative frequencies add to one.

Next, n-grams charts are a kind of art that is difficult to interpret because chart appearance depends directly on the order of considered amino acids and on the width of displaying frame. So, the importance of the more fundamental frequency table is of first rank. Let us explore its use as a tool to compare two proteomes.

104 ***Turning to relative frequencies.*** *In view of the flaws due to absolute frequencies, let us turn to relative ones to check whether or not we can safely compare two proteomes, be they natural or synthetic.*

```
/*
Program I104 twoFreqtables

This program calculates the   frequency tables
of n-grams of two true proteomes and
other two that are synthetic
and displays the corresponding graphics.

True proteomes are read from a file,
Fasta .faa format is required.
False proteomes are  synthesized by random concatenation
of the letters of a given alphabet,
the length of the first one is equal to
the length of the fist real proteome
and likewise with the second.


Relative frequencies result from dividing
absolutes frequencies by the number of n-grams
that are contained in the studied proteome.
In that way, relative frequencies add up to one
and are independent of the length of the text.

Relative frequencies are classified by ranges
and in this way we get the frequency tables
whose bar charts are displayed.

*/
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Random;
import java.util.Scanner;


import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;



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

//========Fundamental parameters========
//=====Modify them as it pleases you====

//letters per n-gram
private static  int n = 2;

//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 800;

//Width of bars in bar chart
private static int widthOfBar = 50;
```

```
//Directory where your files are kept into
private static String nameOfFolder =
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Names of two species whose proteomes are to be compared
private static String nameOfFile1 =
             "Arcobacter_L_uid158135.faa";
private static String nameOfFile2 =
"Zymomonas_mobilis_ATCC_10988_uid55403.faa";
private static String nameOfFile3 =
"Synthetic-1";
private static String nameOfFile4 =
"Synthetic-2";


/*The alphabet conforms to  fasta format
for amino acids (.faa extension).

 //Full .faa alphabet:
private static String Alphabet =
                  "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                  "ACDEFGHIKLMNPQRSTVWY";



private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;



//Initialization of  color variables
```

```
private static Color OurColors[ ] =
   {Color.BLACK, Color.BLUE, Color.CYAN, Color.GRAY,
Color.GREEN, Color.MAGENTA, Color.ORANGE,
Color.PINK, Color.RED,    Color.YELLOW,
Color.black, Color.blue, Color.cyan, Color.gray,
Color.green, Color.magenta, Color.orange,
Color.PINK, Color.RED, Color.WHITE,  Color.YELLOW};
private static String NamesColors[ ] =
   {"BLACK","BLUE", "CYAN","GRAY",
"GREEN", "MAGENTA", "ORANGE",
"PINK", "RED",    "YELLOW",
"black", "blue", "cyan", "gray",
"green", "magenta", "orange",
"pink", "red", "white", "yellow"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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


private static JLabel label = new JLabel("TTTT");


//Machine for reading from hard disc
private static  Scanner input;

//Path to file
private static  String path1 = nameOfFolder + nameOfFile1;
private static  String path2 = nameOfFolder + nameOfFile2;
//Proteomes: two real, two synthetic
private static StringBuffer proteome1 =
                        new StringBuffer("");
private static StringBuffer proteome2 =
            new StringBuffer("");
private static StringBuffer proteome3 =
            new StringBuffer("");
private static StringBuffer proteome4 =
         new StringBuffer("");
```

```
//Vectors to keep relative frequencies
private static double[]  freq1
                      = new double[N] ;
private static double[]   freq2
                     = new double[N];
private static double[]  freq3
                     = new double[N];
private static double[]   freq4
                    = new double[N];
//Maximal values of freq-ith vectors
private static double max1, max2, max3, max4;
//Global maximal value = the greatest max
private static double max;
private static int nMarks;
private static double delta;

//Frequencies are classified by ranges
private static int freqTable1[] = new int[100];
private static int freqTable2[] = new int[100];
private static int freqTable3[] = new int[100];
private static int freqTable4[] = new int[100];

//Generator of random numbers
private static Random r  = new Random();


JFrame f = new JFrame(
"Relative frequency tables of two proteomes");


//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public twoFreqTables()
 {
  this.setPreferredSize(
new Dimension(widthOfFrame, heightOfFrame));
  this.setBackground(Color.white);
```

```
  //A cursor is chosen
  setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
 }


//This i how the application begins
private void starter() throws IOException
{
 System.out.println("Comparative n-gram analysis of \n"
                    + nameOfFile1 + "\n and \n"
            + nameOfFile2 + "\n");
 System.out.println("Length of alphabet = "
          + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of possible n-grams = " + N);
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.add(this, BorderLayout.BEFORE_FIRST_LINE);
 f.add(label);
 f.pack();
 f.setLocationRelativeTo(null);
 f.setVisible(true);
 //Proteomes are read from files or are synthesized
 readProteomes();
 //All the work is done
 study();

}


public static void main(String[] args)
{

  //Thread inauguration: assigns second priority to this
  EventQueue.invokeLater
(new Runnable()
     {
        @Override
        public void run()
        {
```

```
            try {
new twoFreqTables().starter();
} catch (IOException e) {
e.printStackTrace();
}
        }
     }
);
}//End of main



//Draws bar charts
private static void drawCharts(Graphics g2d)
{
 int xo = 10;
 int yo = 120;
 int maxHeight = 100;
 int height;
 //Proteome1
 for (int i = 0; (i < nMarks) ; i++)
  {
   double r = freqTable1[i];
   height = (int) ((r/max )* maxHeight);
   g2d.setColor(OurColors[i]);
   g2d.drawRect(xo + i*widthOfBar, yo-height,
   widthOfBar, height);
  }
 //Proteome2
   for (int i = 0; (i < nMarks) ; i++)
{
   double r = freqTable2[i];
  height = (int) ((r/max )* maxHeight);
 g2d.setColor(OurColors[i]);
     g2d.drawRect(xo + i*widthOfBar, 2*yo-height,
           widthOfBar, height);
}
 //Proteome3
   for (int i = 0; (i < nMarks) ; i++)
```

```
{
   double r = freqTable3[i];
  height = (int) ((r/max )* maxHeight);
 g2d.setColor(OurColors[i]);
     g2d.drawRect(xo + i*widthOfBar, 3*yo-height,
      widthOfBar, height);
}
 //Proteome4
   for (int i = 0; (i < nMarks) ; i++)
{
   double r = freqTable4[i];
  height = (int) ((r/max )* maxHeight);
 g2d.setColor(OurColors[i]);
     g2d.drawRect(xo + i*widthOfBar, 4*yo-height,
      widthOfBar, height);
}
label.setText( nameOfFile1 + ";    " + nameOfFile2 + ";    "
+ nameOfFile3 + ";    " + nameOfFile4 + ". ");
label.setVisible(true);
}


//Drawing tool
//Second method to be executed and
//then on always ready to get into action.

protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
  //Embellishment
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

  drawCharts(g2d);

 }
```

```
//Title specifications for the frequency table
private static void definitions(double max)
{
System.out.println(  "maxh = "+ max);
 //Scale is found
 double k = max;
 double j = 0;
 while (k < 1)
 {
  k =  k * 10;
  j++;
 }

 delta = 1;
 delta = delta / Math.pow(10, j+1);
 System.out.println("j = " + j + " Delta = "+ delta);

 nMarks =   (int) (max/delta)  + 2;
 if (nMarks > 20)
  {
delta = delta * 10;
nMarks = nMarks/10+2;
  }
 for(int i = 0; i < nMarks; i++)
titlesFreqTable[i] = " freq = " + i + " * "  + delta;

}


//Builds the frequency table by ranges.
//counting is a boolean flag for counting
private static int[] classification(double[] freq,
                            double max)
{
int freqTable[] = new int[100];
boolean go = true;
for(int index = 0; index < freq.length; index++)
{
 for(int i = 0; (i < (nMarks+4) & (go)); i++)
```

```
  {
if (( freq[index] >=  i*delta)  &
             ( freq[index] <  (i+1)*delta))
     {
      freqTable[i]++;
      go = false;
     }
  }
 go = true;
}
return freqTable;
}


//Frequency table is reported
private static void reportTable(int[] freqTable)
 {
   System.out.println("Frequency table of " + " " + n
                  +  "-grams  ");
   for(int j = 0; j < nMarks; j++)
      System.out.println(j +  " " + freqTable[j]
       + " for range  " +  titlesFreqTable[j]
        + "  " + NamesColors[j]);
  //Test
  int sumNGrams = 0;
  for(int j = 0; j < nMarks; j++)
       sumNGrams = sumNGrams + freqTable[j];
  System.out.println("Sum =  " + sumNGrams + " " + n
                                        +  "-grams");
  }


//Frequency tables are  reported
private static void reportTables()
{
 System.out.println("\n  " + nameOfFile1);
 reportTable(freqTable1);
 System.out.println("\n  " + nameOfFile2);
 reportTable(freqTable2);
 System.out.println("\n  " + nameOfFile3);
```

```
 reportTable(freqTable3);
 System.out.println("\n  " + nameOfFile4);
 reportTable(freqTable4);
 }


//Frequencies are grouped in ranges
private static void rangeTables() throws IOException
 {
  max1 = findMax(freq1);
  max2 = findMax(freq2);
  max3 = findMax(freq3);
  max4 = findMax(freq4);
  //A global max is found to define a universal scale
  max = max1;
  if (max < max2) max = max2;
  if (max < max3) max = max3;
  if (max < max4) max = max4;
  System.out.println("\nGlobal max of relative freq =" +
                       "  "+ max + "\n");
  definitions(max);
  freqTable1 = classification(freq1, max);
  freqTable2 = classification(freq2, max);
  freqTable3 = classification(freq3, max);
  freqTable4 = classification(freq4, max);
  max1 = findMax(freqTable1);
  max2 = findMax(freqTable2);
  max3 = findMax(freqTable3);
  max4 = findMax(freqTable4);
//A global max is found to define a universal scale
  max = max1;
  if (max < max2) max = max2;
  if (max < max3) max = max3;
  if (max < max4) max = max4;
  System.out.println("\nGlobal max of absolute freq =" +
                       "  "+ max + "\n");
 }


//All the work is done
```

```
private static void study() throws IOException
{
 //n-gram frequencies are found and kept in freq1
 findFrequencies();
 rangeTables();
 reportTables();
}


    //======== Operative part ============

 //Frequencies of n-grams are calculated
 private static StringBuffer randomProteome(int length)
  {
   StringBuffer randText =  new StringBuffer("");
   char c = ' ';
   //N = number of possible n-grams
   for(int i = 0; i < length; i++)
{
 int k = r.nextInt(lengthAlf);
 c = Alphabet.charAt(k);
 //char is appended to text
 randText = randText.append(c);
 }
  return randText;
 }

 //True proteomes are read from files,
 //false ones are synthesized.
 private static void readProteomes() throws IOException
 {
  proteome1 = readData(path1);
  proteome2 = readData(path2);
  int length = proteome1.length();
  proteome3 = randomProteome(length);
  length = proteome2.length();
  proteome4 = randomProteome(length);
 }

//Frequencies of n-grams are calculated
```

```
private static void findFrequencies() throws IOException
{
System.out.println("\nProteome 1: " + nameOfFile1);
 //n is the number of letters in n-grams
 freq1 = nGramsFrequency(n, proteome1);
 System.out.println("\nProteome 2: " + nameOfFile2);
 freq2 = nGramsFrequency(n, proteome2);
 System.out.println("\nProteome 3: synthetic");
 freq3 = nGramsFrequency(n, proteome3);
 System.out.println("\nProteome 4: synthetic");
 freq4 = nGramsFrequency(n, proteome4);


 if (print2)
  {
   System.out.println(proteome1);
   System.out.println("Index, n-gram, and frequency");
   for(int i = 0; i < freq1.length; i++)
   System.out.println(i + "\t" +  indexInverse(n, i)
       + "\t" + freq1[i]);
  }
}


// We read  record from file
private  static  StringBuffer readData(String path)
    throws IOException
{
StringBuffer text = new StringBuffer("");
//try and catch gates
try
{
FileReader fr = new FileReader(path);
BufferedReader br = new BufferedReader(fr);
String s = "";

while((s = br.readLine()) != null)
{
//Filtering of first line of each subsequence
if (s.charAt(0) != '>' )
```

```
{
//System.out.println(s);
text = text.append(s);
}
}

fr.close();
}
catch ( NoSuchElementException elementException )
{
System.err.println( "File is corrupted." );
input.close();
System.exit( 1 );
}
catch ( IllegalStateException stateException )
{
System.err.println( "Reading aborted." );
System.exit( 1 );
}
return text;
}


//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;
System.out.println("Alien : " + p);
}
}
if (print1)
System.out.println( "ngram = " + ngram + " index = "
                        + ind);
return ind;
}

//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 ;

}
if (print1)
System.out.println( "ngram = " + ngram +
 " index = "   + ind);
return ngram;
}


/*
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;
  }
 findMax(F);
 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);
 double max = findMax(F);
 System.out.println("Max of  frequencies "+ max);
 return F;
}


//Maximal frequency is found
private static double findMax(double[] F)
{
 double max = 0;
 for(int i = 0; i< F.length; i++)
 if (F[i] > max ) max = F[i];
 //System.out.println("Max of  frequencies "+ max);
 return max;
}

//Maximal frequency is found
private static double findMax(int[] F)
{
int max = 0;
for(int i = 0; i< F.length; i++)
if (F[i] > max ) max = F[i];
//System.out.println("Max of  frequencies "+ max);
return max;
}

 }//End of class I104  twoFreqTables
```

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

106 ***Important exercise.*** *Play enough with the previous code to reliably agree else disagree with the next claim of the Author:*

The bar chars of n-grams frequencies of natural proteomes and synthetic ones are so dissimilar that it is obvious that the explanation of both type of proteomes cannot be the same. In this case, randomness under a uniform distribution is behind the mechanism of synthesis of synthetic proteomes so we immediately conclude that a uniform distribution is not the explanation of the present form of life on Earth.

If you agree, pass to the next exercise else make fun of both science and creationism because they both resort to systematic effects to explain our existence.

107 ***Exercise.*** *Add to the previous program the code to calculate the necessary values to make a Kolmogorov–Smirnov test to support your conclusion in regard with the previous exercise. The null hypothesis of this test is that data in the two samples come from the same distribution and so, this is exactly what we dealt with.*

108 ***Challenge.*** *To get conscious that randomness comes in infinitely many distributions, and not only in a uniform one, compose the code to synthesize proteomes at random but with different distributions and add the code for a Smirnov-Kolmogorov test, to prove or disprove that you have effectively two different distributions. Hint: you can learn how to generate random texts with diverse distribution by studying program I66 FictitiousDNA. The S-K test can be copied from the previous program I107.*

109 ***Important exercise.*** *Consciously analyze the next claim of the Author to reliably agree else disagree with him:*

We have ruled out a uniform distribution of random events as the explanation of life on Earth. Can we pass from here to reject randomness in general? The ensuing question is: what about all the others distributions? The Author considers that, in regard with proteomes that come from genetic information, randomness means randomness under a uniform distribution because of a lack of a reason to think of another one under natural circumstances.

Therefore, to rule out randomness under a uniform distribution means to rule out randomness in general.

So, in the name of the eJ Community we authorize everybody to belief that direct and isolated randomness is not the explanation of present life on Earth: other proposals are welcome.

110 ***Important exercise.*** *Consciously analyze the next claim of the Author to reliably agree else disagree with him:*

Once one has accepted that randomness is not the explanation of life on Earth, we have two important and interesting options: science (with its Darwinian Evolutionary Theory) and creationism (creation by God or by aliens). Now, which option must we pay attention to with mandatory priority?

In our eJ community we choose science as our priority for suggesting a plausible explanation for our existence. The reason is that our bar charts that were output by the last program are precisely indicators of reuse of basic building blocks. Reuse fits well natural evolution as well as intelligent designers (which also use evolution as a tool for design). But the virtue of natural evolution is that it is by far simpler to study by Java simulations than intelligent designers.

The great conclusion is that we at last have the official right to get involved in the study of science: the difficult art of proposing explanations of folds of nature that could be simple and nice enough to mean a challenge to researchers to test them.

111 ***Tremendous exercise.*** *To be authorized to make science is one of the best news ever heard. So, take your time and some pains to revise the code of the last programs. Work enough to confidently agree else disagree with the claim of the Author: the program twoFreqTables2 is correct and it does what it must. Besides, it is very efficient.*

4.4 Distance matrices

Our tools have allowed us to execute the n-gram analysis of proteomes including the production of n-gram charts and n-gram frequency tables. As a result, we have decided that direct and isolated randomness is not a reliable explanation of proteomes. Let us pass now to the systematic study of the differences among proteomes. We compare them by pairs. Our procedures will associate to each pair a number, an n-gram distance among its members.

Our purpose is to attack this work systematically because we want to build a matrix with the distances between all possible pairs of genomes that are contained in a given folder. Our first task is therefore to learn how to retrieve the names of files in a folder.

112 ***The next code lists the names of files of a folder.*** *A specific extension is allowed.*

```
//Program I112 ListingFiles

//This program lists all files of a given folder
//with a given termination or suffix.

//Chosen termination is .faa

import java.io.File;

public class ListingFiles
{

 public static void main(String[] args)
  {
//Change this path to that to the folder
//with your proteomes.
String path = "/home/jose/jose/AAjoser/Ciencia/" +
            "GenomesComplete/GenomesSample";
File folder = new File(path);
File[] listOfFiles = folder.listFiles();
String file;
System.out.println("List of species: \n");
for (int i = 0; i < listOfFiles.length; i++)
 {
   if (listOfFiles[i].isFile())
   {
    file = listOfFiles[i].getName();
    //Filter for .faa format. Change this on need.
    if (file.endsWith(".faa") )
       System.out.println(i + " " + file);
     }
  }
   }

}//End of main class I112 ListingFiles
```

113 ***Exercise.*** *Adapt the code to your folders and run it. Play with the code, say, reuse it to find files with .tex extension.*

114 ***Dealing with a complete folder of proteomes.*** *The next code finds the matrix of distances among all pairs of genomes that are found in the folder GenomesSample.*

```
//Program I114 EuclideanDistMatrix

/*
 The program 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,
 next it calculates the matrix of Euclidean n-gram distances
 between all proteomes in the chosen folder.

 Sequences must  have extension  .faa (fasta amino acids).

 */


import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;



import java.lang.IllegalStateException;


public class EuclideanDistMatrix
{

 //Machine for reading from hard disc
 private static  Scanner input;

 //Folder with proteomes with .faa extension.
```

```
 //Update this to your circumstances
 private static String nameOfFolder =
  "/home/jose/jose/AAjoser/Ciencia/" +
            "GenomesComplete/GenomesSample";

 //Full path to a specific file
 private static  String path;

 private static File[] listOfFiles;


 //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=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];

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

 //We make run for 7 proteomes
 private static double[][] nGramFreqMatrix = new double[L][7];

 //We make run for 7 proteomes
 private static double[][] nDistanceMatrix = new double[L][7];


 private static Boolean printAll;
```

```
 //All files in the folder that is addressed
 //by path are found.
 //Extension .faa is required.
 public static void getNamesOfFiles(String path)
  {
File folder = new File(path);
listOfFiles = folder.listFiles();

String file;
System.out.println("List of species: \n");
for (int i = 0; i < listOfFiles.length; i++)
 {
  if (listOfFiles[i].isFile())
{
 file = listOfFiles[i].getName();
 //Filter for .faa format. Change this on need.
 if (file.endsWith(".faa") )
       System.out.println(i + " " + file);
 }
  }
}

  // We read  record from file
  public  static  StringBuffer readData(String path)
                  throws IOException
  {
   //A proteome is kept in a stringBUffer (heavy duty string)
   StringBuffer genome = new StringBuffer("");
   //try and catch gates
     try
     {
      FileReader fr = new FileReader(path);
      BufferedReader br = new BufferedReader(fr);
      String s = "";

      while((s = br.readLine()) != null)
      {
      //Filtering of first line of each subsequence
```

```
      if (s.charAt(0) != '>' )
    {
      //System.out.println(s);
          genome = genome.append(s);
    }
      }

      fr.close();
     }
     catch ( NoSuchElementException elementException )
     {
        System.err.println( "File is corrupted." );
        input.close();
        System.exit( 1 );
     }
     catch ( IllegalStateException stateException )
     {
        System.err.println( "Reading aborted." );
        System.exit( 1 );
     }
     return genome;
  }


  //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;
   }
  findMax(F);
  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);
  double max = findMax(F);
  System.out.println("Max of  frequencies "+ max);
  return F;
}


//Maximal frequency is found
private static double findMax(double[] F)
{
 double max = 0;
 for(int i = 0; i< F.length; i++)
 if (F[i] > max ) max = F[i];
 //System.out.println("Max of  frequencies "+ max);
 return max;
}


//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;
 if (printAll)
System.out.println("Relative 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)
  System.out.println(F1[i] + "  " + F2[i]);
    }
   return d;
  }



 //Reports a matrix with the Euclidean n-gram distances between
 //all possible pairs of proteomes in listOfFiles
 public static void calculateMatrix() throws IOException
 {
  StringBuffer proteome = new StringBuffer("");
  //n-gram analysis of all proteomes
  for (int i = 0; i < listOfFiles.length; i++)
   {
String file = listOfFiles[i].getName();
path = nameOfFolder + "/" + file;
proteome  = readData(path);
freq1 = nGramsFrequency(n, proteome);
//Results are kept in a matrix
for (int j = 0; j < freq1.length; j++)
 nGramFreqMatrix[j][i] = freq1[j];
   }
  //Euclidean n-gram distances are found
  int L = (int) Math.pow(lengthAlf, n);
  for (int i = 0; i < listOfFiles.length; i++)
   {
 for (int k = 0; k < L; k++)
 freq1[k] = nGramFreqMatrix[k][i];

 for (int j = 0; j < listOfFiles.length; j++)
  {
 //Euclidean distance between freq1 and freq2
 for (int k = 0; k < L; k++)
 freq2[k] = nGramFreqMatrix[k][j];
 nDistanceMatrix[i][j] =
```

```
 euclideanDistance(N, freq1,freq2);
  }
   }
 }



 public static void printMatrix()
 {
  System.out.println( "\nMatrix of Euclidean n-gram distances" );
  for (int i = 0; i < listOfFiles.length; i++)
   {
for (int j = 0; j < listOfFiles.length; j++)
 System.out.print(nDistanceMatrix[i][j] + "\t");
 System.out.println();
   }
 }


 public static void main(String[] args) throws IOException
  {
getNamesOfFiles(nameOfFolder);
//To print additional information, change to true
printAll = false;
System.out.println( "\nLetters per n-gram = " + n );
System.out.println( "Lenght of the complete alphabet = " +
                                      lengthAlf);
System.out.println( "Number of n-grams in vector " +
                                  "SelfEng = " + N);

    calculateMatrix();
    printMatrix();
   }

}//End of main class I114 EuclideanDistMatrix
```

115 ***Exercise.*** *Adapt the code to your circumstances, run it and play with it.*

116 ***Exercise.*** *The output of the previous program is difficult to read because so many zeros in the distances. Invent something to easy that task.*

4.5 Evolutionary theories

We have ruled out randomness as a direct and pure explanation of proteomes. Now, what else could be the right explanation? It is a designer. But, is the designer rational or irrational? To decide this question, we must in first place burn irrational designers. To that aim, let us formulate a very simple model in within the evolutionary framework.

117 ***The nearest neighbor phylogeny.***

By the very nature of life and once randomness has been ruled out, evolution (reproduction with mutation, recombination and selection) is the most immediate candidate as a designer of life. This directrix must be made into specific evolutionary theories with clear cut predictions. This can be done in many, many forms, so it would be nice if we can single out a natural, simple and robust form to do that. Our invitation is to begin with a very natural question and its simplest answer:

Who is my father?

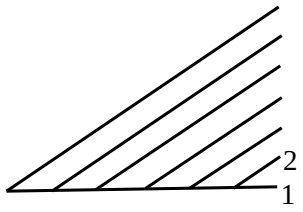
If no additional information is at hand, my father is the man that most resembles me. This gives rise to the **nearest neighbor phylogeny**.

118 ***Example.*** *We use real data and our previous software to calculate a distance matrix and the nearest neighbor phylogeny:*

```
List of species and the length of their proteomes:

0 Edwardsiella_tarda_EIB202_uid41819.faa: 1067782
1 Acaryochloris_marina_MBIC11017_uid58167.faa: 1816176
2 Zymomonas_mobilis_ATCC_10988_uid55403.faa: 571778
3 Neurospora_crassa_FGI.faa: 4784823
4 Arcobacter_L_uid158135.faa: 912047
5 Syntrophomonas_wolfei_Goettingen_uid58179.faa: 807837
6 Cenarchaeum_symbiosum_A_uid61411.faa: 620959



Amplified and rounded  Matrix of Euclidean n-gram distances
  0
 59   0
```

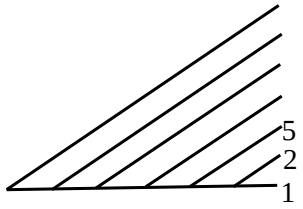
```
 56  50   0
 86  69  69   0
139 114 111 126   0
 79  58  54  80  85  0
 82  80  66  73 129 77 0
```

In the first step we find the shortest distance given by the matrix: it is 50 that measures the differences between species 2 and 1. Next, we draw a tree as follows:

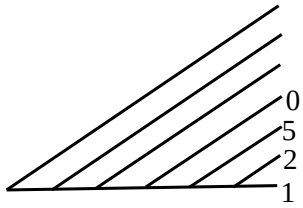


This depiction saids that species 1 and 2 have a common ancestor which is not shared as a common ancestor with all other species.

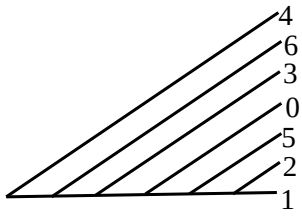
Now we can choose the closest species to the cluster formed by species 1 and 2: we take species 5 because its distance to species 2 is 54 which is the minimum one from all other species. We consequently update our proto-tree:



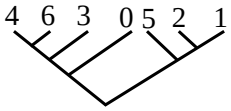
Which is the next species to be added? The closest one to the cluster 1,2,5. It is species 0 because its distance to species 2 is 56. The tree is now:



And so on: we end with the following tree:



The tree we have constructed is an **unrooted tree**, because in spite of its appearance it shows no direction of evolution. To see this, it is enough to deform the tree in one of the many equivalent forms, as the following:



Observe that all species appear at the same horizontal level: this means that they are contemporaries. In fact, they all exist in our days.

119 ***Challenge.*** *Test the just built nearest neighbor phylogeny. To begin with, use the length of the proteomes as indicators of complexity: the more far we are one from another according to the distance, the more complex must be the changes along any path connecting us. If you find a contradiction, try to use the bivariate criterion given by* ***nearest distance + minimum complexity change*** *to suggest a plausible* ***rooted tree****, i.e., a tree that shows a direction of evolution across time. Most possible no satisfactory tree can be found so one must get content with a tree with the less number of contradictions. Species ordered by length of proteomes from simplest to most complex: 2,6,5,4,0,1,3.*

120 ***Graduation.*** *Use the technology that we have learned in this section about folder management to devise an application that reads the proteomes of a folder, shows for each species the 2 and 3 n-gram charts and the corresponding bar diagrams. Include navigation buttons of next and previous species. Observe that the nearest neighbor phylogeny is completely determined by a sequence of species, in our case: 4,6,3,0,5,2,1. So, no actual graphic is necessary. Develop and include the corresponding Java algorithm.*

121 ***Challenge.*** *Dress the software that we have developed in this volume in a GUI.*

4.6 Conclusion

In previous chapters we developed some software for the study of language divergence. Here we have reused it for the study of the proteome. We can see now an n-gram chart showing the n-gram frequencies of a given proteome and the corresponding bar diagram. The synthesis of artificial proteomes by concatenation of chars taken at random allowed us to test the hypothesis that n-gram frequencies fit a uniform distribution. Because we consider that this is the only natural distribution to care of, we conclude that this test really decides whether or not we are a direct effect of randomness. Our answer is negative and so we invite everyone to propose alternate explanations. The Darwinian Evolutionary Theory is, of course, the natural candidate.

We used n-gram relative frequencies to quantify the difference between two species and to form a distance matrix for the set of species whose proteomes are located in a given folder. Next, we used that matrix to build an evolutionary tree that is singled out by the nearest neighbor phylogeny that says: my father is the man that most resembles me. Thus, for the first time in the history of the eJ community, the Evolutionary Theory is not a theory sustained by other people, but a theory that we made ourselves and that we will test with pleasure.

Chapter 5

Review

```
This is a very important work for us
```

THE IMMUNE SYSTEM AND HOMEMADE EVOLUTIONARY THEORIES

We refer here how we began with the immune system, rediscovered the n-gram technology and witnessed its proficiency in language discrimination. Next, we used it to calculate a distance matrix for proteomes of a set of species, and implemented the Evolutionary Theory into a concrete proposal: the nearest neighbor phylogeny that is defined by the dictum that my father is the man that most resembles me. In that way, the evolutionary enterprise is anymore a borrowed dealt: we have thoroughly illustrated how to convert simple, natural ideas into algorithms that take real data and end with an evolutionary theory with clear cut retro-dictions that are packed into a phylogenetic tree.

5.1 Introduction

Nature has a natural system that discriminates self from non-self and whose operation is routinely employed to recognize languages.

5.2 The immune system

The body routinely launches attacks against invading microbes, infected cells and tumors while helping the recovery of damaged tissues. Were not for its work, one

would rapidly die. How is that achieved? The central point is to distinguish self from non self, a task that is fulfilled by the **immune system**.

The accepted explanation of how the immune systems discriminates self from non-self is given by the clonal deletion theory that proposes a learning procedure (NIH, 2013):

1. The original setting of the immune system of the embryo is that everything, self and non-self, must be destroyed.

2. Immune cells give rise to clones over the life of the organism. Each immune cell has a target chosen at random but the population of immune cells is large enough that targeting is almost exhaustive. If a clone is exposed to something early in its life, the representative cells activate an internal self-destructive pathway, and the immune cell dies. If an immune cell matures, it and its clone will get ready to attack its target if it only appears.

3. So, self is what kills young immune clones and non self is what comes next after their maturation.

4. This system works if the embryo is protected by the immune system of the mother from the attacks of every invader.

This theory immediately explains rheumatism as an auto-immune illness: the inner tissues of bones are not exposed to the immune system early in life, when the immune system is in its process of learning. But when late in life a lesion to the bone happens, preferentially at the joins by the accumulation of residues, some proteins that never have been seen by the immune system make their appearance and thus they are taken as alien and so the tissues that contain them are prosecuted to disintegration.

The foreign molecular entities that trigger the operation of the immune system are preferentially proteins, which are encoded by DNA. Nevertheless, the immune system does not deal with entire proteins or enzymes but with small portions of it:

Every kind of proteins are continuously recycled and one of the first steps is to cut them to pieces up to 6 or 7 amino acids long. These small strokes are precisely the objects that are revised by the scanning cells of the immune system (Darnell et al, 1986).

We have now a simple theory:

Immunological recognizing theory*: reading of small DNA or protein segments suffices to discriminate self from non-self.*

This theory is simple enough to be tested with Java immediately: the theory must also function if we use written texts of human languages as testbed for our theory. Our election seems quite reasonable since after some initial training, one can discriminate at once if a text is written in English or in German. How is that possible? Because one looks for substrings with typical combinations of letters. Say, the substring *freu* does not happen in English but it appears in German. As a consequence, one inmediately knows that the word *freud* is not in English and that it belongs in German.

Officially, a substring n-letters long is called an **n-gram**. Thus, our brain as language recognizer and the immune system both have n-grams at the basis of its functioning. A test in Java is mandatory. So, with a Java program we verified that n-gram frequencies are useful to discriminate between English and German and even to compare diverse authors in the same language. We used the **holographic technique** which works as follows:

For a given n, all the n-grams of a training text are recorded. The proportion of alien n-grams in another text in the same language is calculated. The same is done with a text in a foreign language. Thus, we end with two proportions of contrasting n-grams. We find the semi-sum and this defines a threshold t: if the proportion of alien n-grams of a given testing text is less than t, the text is written in the initial language otherwise in the second one.

5.3 Curious facts

We studied the behavior of waiting times (the additional number of chars that one must scan to get the next brand-new n-gram) and we found an emergent exponential envelope in the large scale that was made visible by smoothing, i.e. averaging values close to the one under study.

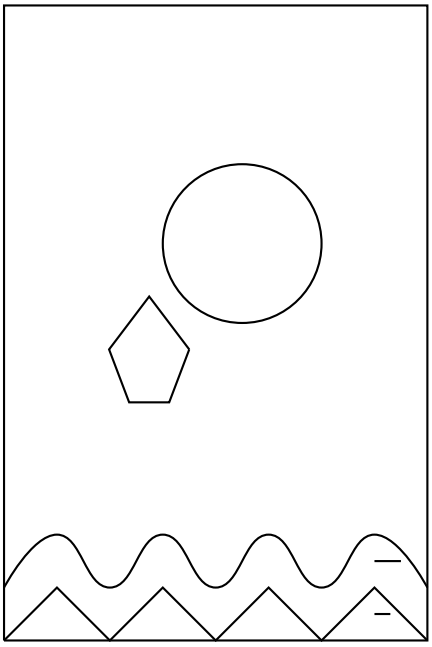
An exponential envelope of the waiting time function is also observed in texts that have been synthesized as concatenation of random chars. On the other hand, the 18 most common n-grams of English and German are distinctive of each one and do not appear in the other. Besides, the distance between English and German is for small values of most common n-grams very similar to that between German and random texts.

These curious facts awakes the imagination in regard with the origin of languages: we all know that languages evolve along human lineages across their dispersal in time, space and culture. And since a word is just a term whose meaning is conventional, it might be natural to consider that accumulation of small change

could be all to the evolution of languages. But our curious results call for a challenge:

Since randomness is not that far from human languages, has it been used to synthesized de novo some human languages? This is a curious idea that combines with our curious facts. The marvelous thing is that ancient cultures gifted us a testimony in favor of the thesis that languages or parts of them have been invented many times de novo along the human history. The witness reads as follows:

5.4 The Bachué myth



The goddess Bachué is a primordial Mother and Creator Goddess of the Chibche people of Columbia. Her name means The One With the Naked Breast. She is the Mother of mankind. According to legend, She emerged of the waters in the Lake Iguaque with a baby in her arms; this baby grew to become her husband and together they became the progenitors of the human race. And once the humans had grown up and learned to live on their own, She turned Herself and Her mate into

dragons and returned to the depths of the lake...leaving the natives with a world of knowledge.

Bachue is also a Goddess of the harvest, of agriculture, of water and farming...as well as the Goddess of famine. She occasionally returns to guide Her creation and watches over the crops that sustains mankind.

Every year the Andean Indians assembled at Lake Guatavita to venerate this Goddess.

5.5 Interpretation

The myth is ordinarily seen as a beautiful piece of religious literature of an ancient culture. That is not indeed the whole story: one might notice that the aforementioned version include dragons, an European sign of the power of darkness, but the original myth refers to snakes, which are nice and not poisonous at the region where the myth was originated. The reason of this gyro is that the myth is also a treasure for modern fans of dark religions, a fact that its author, *mxtodis123* (2013), clearly unveils in the name of the link to his blog: *reclaimingthedarkgoddess*. And, what is the knowledge that the Author so much appreciates? He says nothing about that because there is only one option: the cult of the dead is a generous source of dark power that enables desires to be fulfilled: just pray to your grandmother, or to a saint or to a goddess and you will see that your dreams come true.

The religious force of this myth is so great that it fades away another aspect that has an immense relevance for our work: the myth has a natural anthropological interpretation in terms of a report of a true and recurrent story about the colonization of the Earth by ancient scouting couples:

A young couple, boy and girl teenagers, decide to abandon their homeland to go to live their own life far away from any perturbing society. The boy dies in the enterprise while the girl remains alive with a baby in her arms. As time flies, the baby rises up, becomes the husband of her mother and together populate the new land.

Now, young couples have an immanent tendency for escaping. This means that the myth refers a particular unfolding of a very usual event in the ordinary life of our early ancestors with immediate linguistic implications:

1. If both parents survive, the language of the new land is expected to be very similar to that of the mother land.

2. If the father dies, the new language would have new words invented by the progeny to name events related with male activities, say, chase. If the mother dies, the progeny would need to invent words for the intimate feminine life in relation with family, cooking, dressing, agriculture.

3. Many variants exist that may afford with linguistic variability, say, the progeny may get maturity but has defects such as anatomical problems with lips, palate or larynx that deprive them from a perfect verbal communication with healthy grand sons.

All these events point to the the possibility that languages or part of them have been invented de novo along the population of the Earth. Observe that proposed mechanism, officially known as a bottleneck or founder effect (Nettle, 1999), also provides us with testable predictions.

5.6 Distinguishing styles

With some difficulty, the n-gram technology even allows us to distinguish two authors because they have different styles and we have found a way to quantify that difference. So, what is a style? A **style** is the mode of using and reusing n-grams, words and expressions.

5.7 Application to proteomics

Having tested our machinery over languages, we pass to apply it over **proteomes**, the complete sets of proteins of a given species.

For the very purpose of n-gram analysis of the proteome, we added the possibility to see n-gram charts showing the n-gram frequencies of a given proteome and the corresponding bar diagram. With these tools it becomes apparent that randomness is not the explanation of the proteome. In fact, the synthesis of artificial proteomes by concatenation of chars taken at random allowed us to test the hypothesis that n-gram frequencies fit a uniform distribution. Because we consider that this is the only natural distribution to care of, we conclude that this test really decides whether or not we are a direct effect of randomness. Our answer is negative: we find no reason to consider that the proteome is explained by a direct effect of randomness.

Let us be a bit more expressive in regard with our decision and its relation with the scientific method: it is possible to consider that science cannot reject randomness anyhow because if something exists then it also can be generated by randomness directly. While this consideration is plainly valid, there is no a single person that accepts that: who will believe that free bodies fall not because the force of gravity pulls them but just because of a coincidence at random? So, our directive is this: the reason that compels us to believe that a gravity force exists must be the very same reason that inspire us to believe that the proteome is not the direct result of randomness. So, what is that reason? It is the following:

One releases a body to see its trajectory: it falls down. This may happen by randomness. One repeats the experiment: in this occasion the body also falls down. This also may happen by randomness but the joint probability diminishes. And so on: one can pick up as many events as desired and no exception is found if the body is massive enough. This also can happen by randomness but the probability is exceedingly small. To get a probability as small as desired, one needs to increase the number of repetitions but only to a finite limit (our Volume V might help you to get expertise on this topic).

The same happens with our study of the proteome: one sees that the n-gram charts of a species is quite different from that of an artificial proteome that is the fruit of synthesis by concatenation of chars taken at random from a given alphabet. This may happen by randomness. So, one can go to test another species that also can be explained by randomness but with a joint lower probability. Then one can go to test a third species. And so on: thanks to the work and kindness of Bailin Hao (2013b), we enjoy a data bank with the proteomes of more than 2000 microbial species.

At last, we do not rule out randomness, but direct experience gives us the courage to decide that direct randomness is not the explanation of our existence. As we see, science is not a matter of science but of courage.

Since we reject randomness as the direct explanation of our existence, everyone is invited to propose alternate explanations to our existence.

The Darwinian Evolutionary Theory is, of course, the immediate candidate: given that reproduction involves mutations and recombination, their accumulation under favorable environments might produce great changes, whose real span must be investigated. In any case, evolution is a crude reality that ruthlessly threatens us through microbial attacks. They permanently defy us to understand evolution to the full.

To make a concrete evolutionary proposal that could be tested and falsified, we

need an operational methodology. The simplest one is built around the following concept: *my father is the man that most resembles me.* As a consequence, we need a method to quantify differences: my father is that man that differs the least from me. Mathematically, this is done by means of a distance. The simplest versions suffice for our purpose.

5.8 The Euclidean n-gram distance

A distance is a mathematical function that is used to measure differences among objects. For text objects, say, DNA, proteins or languages, many diverse distances have been defined. For instance, noticing that the immune system can discriminate between self and not self, we deduce that this system defines a discrete distance: the distance between self and self is zero but between self and not-self is one. Other distances also are very useful:

Motivated by geometry, the Euclidean distance is an abstraction of the method of measuring distances between two points of a plane by counting the number of steps across the straight line that joins them. One can eventually pass from the plane, a space of two dimensions, to spaces of arbitrary dimension. In that way, we can apply the Euclidean distance to n-tuples or vectors composed of the absolute frequencies of diverse n-grams. The resultant distance is called **Euclidean n-gram distance**. It can be applied to pairs of texts of the most varied nature and diversity and so our immediate concern is to apply it to proteome sequences. The problem is that this distance depends on the length of texts, so if a text is composed of three copies of a short one, the original and enlarged texts will appear as highly distant one form another. Such effects are not welcome everywhere so, we have provided the possibility to resort to relative n-gram frequencies, which are obtained by dividing absolute frequencies by the total number of n-grams in the analyzed text. On this new setting, the term **Euclidean n-gram distance** refers, by default, to the Euclidean distance between two n-tuples or vectors of relative n-gram frequencies.

So, we used n-gram relative frequencies to quantify the difference between two species and to form a distance matrix for the set of species whose proteomes are located in a given folder. Next, we used that matrix to build an evolutionary tree that is singled out by the **nearest neighbor phylogeny** that is determined by the dictum: my father is the man that is closest to me in the relative n-gram frequency space endowed with the Euclidean distance.

Challenge: We also have used the n-gram technology to discriminate between two authors. Why is that possible? Because expressing oneself is a very complicated problem and so it can be solved in many ways. So, every Author has his or her

personal form, i.e., the used algorithms to pack personal ideas differ from author to author. Is this difference that is measured by the n-gram analysis. This explains why it is possible to discriminate between two authors. Now, we are saying that we can also discriminate between two species. Why is that possible?

5.9 On the importance of this work

In hindsight, the work that has been developed in this volume is of maximal importance: the Evolutionary Theory always has been taken by us as a theory invented by some else and sustained by other community. So, we have been working with borrowed objects. With this volume such shameful situation has been healed: we do know how to build a concrete evolutionary theory beginning from scratch and using tools and ideas that have been developed and fabricated by observing nature directly. It is certain that our one and only theory is rudimentary, but nothing else is needed to spark an evolutionary trend that looks for as much perfection as desired.

Answers to exercises

Problems of Chapter 1

18, page 6. To discover that *calcultae* is a misspelling of *calculate*, a mismatch distance is enough. But to discover that *calclae* is also a misspelling of the same word, one needs a distance that include best alignment procedures.

26, page 22. Expectations are approximately correct, but as an enveloping tendency. Errors might be explained by the fact that texts are not truly random: some redundancy is used to allow for self-correction.

36, page 54. The *hologramic technique* to discriminate between two authors is as follows: for one author, take some initial text as training text for self and the rest of the text as contrasting one. Next take another author and contrast the proportion of aliens n-grams in the contrasting text with that of the other author. Conclusion: When the length of texts grows, the whole power of the language is employed and so different authors tend to be indistinguishable. Short samples are preferred to distinguish two authors. We are saying that we can distinguish two authors because they have different styles and we have found a way to quantify that difference. So, what is a style? A **style** is the mode of using and reusing n-grams, words and expressions. The used code follows:

```
/*Program I36 TwoAuthors

 The purpose of this program is to discriminate
 whether two authors that writer in the same language.
 The program accepts texts of arbitrary size,
 such as a complete book.
```

```
The books must be copied by the User to the hard disk
and paths to corresponding files
must be updated in line 225.

We mimic the function of the immune system,
whose work is based in n-grams, that are
string n-symbols long.
Output is graphical.

The program has four steps:

1. Definition of self:
Up to some n, all possible n-grams are generated
and recorded in the boolean vector
SelfEng. This is done trickily: we generate a
boolean vector SelfEng and at the beginning,
every entry of SelfEng is set to TRUE.
And this is all to it.
The idea behind this is that each n-gram
is given an index, a number that
identifies it in the boolean vector SelfEng.
So, recording of Self is done as follows:
the index of each n-gram of the trainingText is
calculated and the corresponding entry in SelfEng is
set to false.
This mimics the training stage of the immune system.
The first part of a book is the training text.

2.Test over an English text: Because of the finiteness
of the learning text,
even another text in English might generate alien
n-grams. So, a second text in English is tested against
the former one. This defines a calibration stage:
the proportion of alien n-grams is reported.
The second part of the same book is used here.


3. Test over a second author:
```

```
 A second book,  in English but by another author,
 is analyzed:
 its n-grams are compared against those in the list
 that defines the self.  The proportion of
 alien n-grams is calculated. When text are long,
 this proportion
 approximately doubles
 that of the second text in English.

 4.Formulation of the discriminating criterion:
 a classifying threshold is defined as the arithmetic
 average of the two former proportions. If the proportion
 of alien n-grams of a new text is less than
 that of the threshold, the text is in English, otherwise
 it is in another language.

 Various graphics are displayed for diverse values
 of the number of chars taken from the training text.
 Each  graphic represents
 the proportion  of alien n-grams that one finds in
 a testing text as a function
 of the length in chars of that testing text.
 Two testing texts are contrasted: one in English
 the other in German.

*/



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;


import java.lang.IllegalStateException;
import java.util.Scanner;

public class TwoAuthors extends JFrame
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static TwoAuthors p = new TwoAuthors();
 //The painting class is declared and instantiated
 private static Canvas2 c = p.new Canvas2();
 //Number of letters in the n-grams
 private static int n;
 //Number of n-grams
 private static int N;
 //DataBase: if n=4, assigned memory must
 //be increased 100 times
 private static Boolean SelfEnglish[] =
                            new Boolean[450000];
 //Training  text. A StringBuffer is a heavy duty String.
 private static StringBuffer trainingText ;
 //Test  text
 private static StringBuffer testerEng, testerGer;
 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
 private static  int lengthAlf;
 private static int nTrainingText, nTesterEngText,
                   nTesterGerText, nTrainingSample,
                   nTesterEngSample, nTesterGerSample;
 private static double propSelf, propTest;
```

```
 private static Boolean printAll, printTables;
 private static int amplificationY;
 private static double scale;
 private static int graphNumber;
 private static int Points[][] = new int[100][3];
 private static int xmax, ymax;
 private static int deltaX, deltaY, nPoints;
 private static int deltaLearningText, deltaTestingText;
 //Machine to read from the hard disk
 private static  Scanner input;


 //*****************MAIN*****************
 public static void main( String args[] )
 {
  System.out.println("Please, wait for a while");
  //letters per n-gram
  n=3;
  //The size of learning text is incremented by this
  deltaLearningText = 20000;
  //The size of testing text is incremented by this
  deltaTestingText =  20000;
  //Graphical scale = 1 for normal size.
  scale = 0.15;
  //To print all, change to true
  printAll = false;
  //To print numeric results, change to true
  printTables = false;
//length of the Alphabet
  lengthAlf = Alphabet.length();
  System.out.println( "\nLetters per n-gram = " + n );
  System.out.println( "Lenght of the alphabet = " +
                                    lengthAlf);
  //Length of Self
  N = (int) Math.pow(lengthAlf, n); ;
  System.out.println( "Number of n-grams in vector " +
                                  "SelfEng = " + N);
  //The coordinates of points for graphics are set to zero.
```

```
  initialize();
  readTexts();
  //The JFrame is constructed with a title
  JFrame f = new JFrame( "Language identifier" );
  //Control is passed to the painting class c (Canvas)
  f.add( c, BorderLayout.CENTER );
  f.add( new JLabel("English (RED) vs. German (BLUE)"),
                              BorderLayout.SOUTH );
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  f.setSize( 1050, 250 );
  f.setVisible( true );
 }


 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }

 // File is read
 public  static  StringBuffer readData()
 {
  StringBuffer text =  new StringBuffer("");
  //try and catch pairs
    try
    {
       while ( input.hasNext() )
       {
```

```
         text = text.append(input.next());
         //Insert a space among words
         //text = text.append(" ");
        }
    }
    catch ( NoSuchElementException elementException )
    {
       System.err.println( "File is corrupted." );
       input.close();
       System.exit( 1 );
    }
    catch ( IllegalStateException stateException )
    {
       System.err.println( "Reading aborted." );
       System.exit( 1 );
    }
    return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
```

```
//====Make the changes through your own paths=====
//================================================



//Reading second author in English
private static void readSecondAuthor()
{
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
          "ImmuneDistances/SamuelButler.txt";
  testerGer = getText(path);
}

//A first book is read.
//The first part is the training text.
//The second half is the contrasting text
//SecondBook is read.
private static void readTexts()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
  "ImmuneDistances/BookDelight.txt";
 StringBuffer firstBook =  getText(path);

 int nFirstBook = firstBook.length();
 System.out.println( "Lenght of firstBook = " +
  nFirstBook);

 //TrainingText is the first part of firstBook;
 int nFirstBookHalf = nFirstBook/2;
 //TrainngText is the first part of first book,
 //delete second part.
 trainingText =
 firstBook.delete(nFirstBookHalf,nFirstBook);
 nTrainingText = trainingText.length();
 System.out.println( "Lenght of trainingText  = " +
                              nTrainingText);
 String s = trainingText.substring(0, 100);
 System.out.println("First chars of trainingText " +s);
```

```
  //TesterText is the second part of firstBook.
  //Reread firstBook (no other solution was found)
  firstBook =  getText(path);
  //testerBook is the second part (first is deleted).
  //delete first part
  testerEng = firstBook.delete(0, nTrainingText);
   System.out.println( "Lenght of secondPart = " +
           firstBook.length());
   s = testerEng.substring(0, 100);
   System.out.println("First chars of the second part" +
                         " of first author " + s);
  readSecondAuthor();
  nTesterGerText = testerGer.length();
  System.out.println( "Lenght of testerGer = " +
                               nTesterGerText);
  s = testerGer.substring(1000, 1100);
  System.out.println("Sample text from secong author " + s);
}



 //Mandatory initialization
 private static void initialize()
 {
  for(int i = 0; i < 100; i++)
for(int j = 0; j < 2; j++)
  Points[i][j] = 0;
 }

 //SelfEng contains all possible n-grams.
 private static void initializeSelf(long N)
 {
  for(int i = 0; i< N; i++)
   SelfEnglish[i] = true;
 }

 //Returns the index   of the n-gram 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.
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.print( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}


//The boolean vector Self records the identity
//of the training text.
//At the beginning, every entry of Self is set to TRUE.
//Each n-gram is given an index, a number that
//identifies it in the boolean vector Self.
//The index of each n-gram of the trainingText is
//calculated and the corresponding entry in Self is
//set to false.
//The variable nTrainingSample decides
//how many chars of trainingText are taken.
```

```
private static void DefinitionOfSelf(int n,
                                int nTrainingSample,
                          StringBuffer trainingText)
{
 initializeSelf(N);
 nTrainingText = trainingText.length();
 int m = Math.min(nTrainingText, nTrainingSample);
 int start = 0;
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  trainingText.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.
if (ind >0)
SelfEnglish[ind] = false;
start = start +1;
  }

}

//
//The n-grams of testText are compared against
//those of the trainingText, whose structure was  kept at
//SelfEnglish. The proportion of non-self n-grams
//is reported. nSample chars are taken from testText.
private static double runTest(StringBuffer testText,
                                  int nSample)
{
 if (printAll)
 System.out.println( "Alien n-grams");
 int counter = 0;
 int l1 = testText.length();
 int m = Math.min(l1, nSample);
 int start = 0;
```

```
 int ind = 0;
 for(int c=0; c < m-n+1; c++)
  {
String ngram =  testText.substring(start,
                                   start+n);
ind = index(n,ngram);
if (ind >=0)
{
 if (   SelfEnglish[ind] == true)
{
 counter++ ;
 if (printAll)
 {
  System.out.print(" alien");
 }
}
}
if (printAll)  System.out.println();
start = start +1;
  }

 double prop = counter;
 prop = prop / (m-n+1);
 return prop;
}

//The training text is studied
private static void learning(int n,
        int nTrainingSample,
        StringBuffer trainingText)
{

 System.out.println( "\nNumb of chars taken from" +
         " trainngText = " +  nTrainingSample);
 DefinitionOfSelf(n,nTrainingSample, trainingText );
}

//A second text in English is studied
```

```
private static double calibration(StringBuffer testText,
        int nSample)
{
 propSelf = runTest(testText, nSample);

 return propSelf;
}

//A text in German is classified as self else not-self
private static double testing(StringBuffer testText,
        int nSample)
{
 propTest = runTest(testText, nSample);
 return propTest;
}

//Main operational work
private static void work()
{
 learning(n,nTrainingSample, trainingText );
 if (printTables)
 {
  System.out.println("LT = Length of testing text");
  System.out.println("PropEng = proportion of alien " +
                          "n-grams in  testerEng");
  System.out.println("PropGer = proportion of alien " +
                           "n-grams in  testerGer");
  System.out.println("LT  \t PropEng \t        PropGer");
 }
 boolean go = true;
 nPoints = 0;
 for(int h = 0; go; h++)
 {
  //How many chars of testerEng must be taken
  nTesterEngSample = deltaTestingText*h;
  propSelf = calibration(testerEng,nTesterEngSample);
  //How many chars of testerGer must be taken
  nTesterGerSample = deltaTestingText*h;
```

```
   propTest = testing(testerGer, nTesterGerSample);
   double difference = propTest - propSelf;
   if (printTables)
    System.out.println( nTesterEngSample + "\t"
   + propSelf+ "\t"+ propTest+ "\t" + difference);
   Points[h][0] = nTesterEngSample ;
   amplificationY = 100;
   Points[h][1] =
   (int) Math.round(propSelf*amplificationY);
   Points[h][2] =
   (int) Math.round(propTest*amplificationY);
   if ( (nTesterEngSample > testerEng.length()) ||
   (nTesterGerSample > testerGer.length() ))
   go = false;
   nPoints = nPoints +1;
  }
 }


 //This is the painting class
 public class Canvas2 extends JPanel
 {

private static final long serialVersionUID = 1L;

    //Constructor
    public Canvas2()
    {
     //initializes the frame;
     repaint();
    }



    //Data of a graphic is normalized to be placed
    //in a given rectangle
    private void rescale( int[][] PointsD)
    {
```

```
 //scale factor are defined
 double  fx;
 double  fy;
 xmax = Math.min(nTesterEngSample,nTesterGerSample);
 ymax = amplificationY;
 double rxmax = xmax;
 double rymax = ymax;
 fx = 150 * scale / rxmax;
 fy = -170 * scale / rymax;
 int tx = 0;
 int ty = (int) (scale * 170);
 //Scaling formulae
 //System.out.println("Coordinates in the graphic");
 for(int i = 1; i < nPoints; i++)
 {
  Points[i][0] =  (int) (PointsD[i][0]*fx +tx);
  Points[i][1] =  (int)  (PointsD[i][1]*fy +ty);
  Points[i][2] =  (int)  (PointsD[i][2]*fy +ty);

  /*System.out.println( Points[i][0] + "\t"
   + Points[i][1]+ "\t"+ Points[i][2]);*/
 }
}


//The graphic is drawn
private void draw(Graphics g )
{
 //Axes
 g.setColor(Color.BLACK);
 g.drawLine(deltaX, 0, deltaX, (int) (170*scale));
 g.drawLine(deltaX, (int) (170*scale),
     deltaX + (int ) (150*scale),
                (int) (170*scale));
 g.setColor(Color.RED);
 //Prop of alien n-grams in English Text
 for(int count = 1; count < nPoints-1; count++)
  g.drawLine(Points[count][0]+deltaX,
```

```
      Points[count][1]+deltaY,
         Points[count+1][0]+deltaX,
         Points[count+1][1]+deltaY);
 g.setColor(Color.BLUE);
 //Prop of alien n-grams in German Text
 for(int count = 1; count < nPoints-1; count++)
   g.drawLine(Points[count][0]+deltaX,
      Points[count][2]+deltaY,
      Points[count+1][0]+deltaX,
      Points[count+1][2]+deltaY);
 //Graphic number
 g.drawString(""+ graphNumber,   10 + deltaX,
                (int) (scale * 186 + 10));
}

//A footnote is drawn
private void footnote(Graphics g )
{
 g.drawString("Length of Training text =  " +
       deltaLearningText + "*graphNumber",
        10,   180 + 10);
 g.drawString("Proportion of alien"+ n + "-grams as " +
    " a function of the length of tested material. " +
  "Vertical scale: from 0 to 1." +
   "Horizontal scale: from 0 to " + xmax, 10, 204);
}



//Redraws graphics when either c or repaint() is called
public void paintComponent( Graphics g )
{
deltaX = 0;
deltaY = 0;
//go is the condition for halting the for-loop:
//the studied sub-text must not be larger than
//the complete training text
boolean go = true;
```

```
    for(int i = 1; go; i++)
     {
      System.out.println( "\n********GRAPHIC NUMBER " + i
                + "********");
      //Number of graphic
      graphNumber = i;
      //Length of studied testing texts
      nTrainingSample = deltaLearningText*graphNumber;
      //Computations
      work();
      //Adjusting to scale
      rescale(Points);
      //Draw a  graphic
       draw(g);
       //Shifts horizontally
      deltaX = deltaX + (int) (scale * 200);
      //If training test is coped with, halt
      if (nTrainingSample > nTrainingText) go = false;
      }
     footnote(g);
     }
 }// end of class Canvas
}//end of main class I36 TwoAuthors
```

Problems of Chapter 2

53, pag. 101.

This is our proposal: Let E be the set that contains the 18 most common n-grams that are distinctive of English. Let G be the set that contains the 18 most common n-grams that are distinctive of German. Let T be the set with the n-grams of the text that is given for scrutiny: if the intersection of T and E has more elements than that of G and T, the text is written in English, otherwise it is in German. We claim that our method is simple, direct, fair and proficient. The code follows:

```
/*Program I53 SDDiscriminator

 We use the symmetric distance between sets
```

```
together with the most common n-grams in a text
to discriminate whether a set is written
in English else in German. It
mimics what a human being does when recognizing
at once that a text is in English else in German.


The program has four steps:

Step one:
Two texts are given for examination.
All n-grams of each text are captured
and their absolute
frequencies are recorded.
This operation is done with the help
of an indexing procedure.

Step two:
The most common   words, up to certain limit,
are picked up
and this defines two sets A and B.
The elements that make the difference are listed.
Example: A = {1,2,3,4,5}  and
B = {4,5,6,7,8,9}.
The set with the elements that make the difference
is Delta = {1,2,3,6,7,8,9}.
This set consists of  those elements
that are in A but not in B
joined to  the  elements that are in
B but not in A.

Step three:
The symmetric distance between
the two texts is defined as
the number of elements in Delta, 7 in our example.

Step four, discrimination properly:
Let E be the set that contains the 10 most common
```

```
 n-grams that are distinctive of English.
 Let G be the set that contains the 10 most common
 n-grams that are distinctive of German.
 Let T be the set with the n-grams of the text
 that is given for scrutiny: if the intersection
 of T and E has more elements than that of G and T,
 the text is written in English, otherwise it is in German.

*/



import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.util.Random;
import java.lang.IllegalStateException;
import java.util.Scanner;

public class SDDiscriminator
{

 private static final long serialVersionUID = 1L;

 //The application is instantiated
 static SDDiscriminator p = new SDDiscriminator();

 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n=3;
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);
```

```
 //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector2
{
int L = (int) Math.pow(lengthAlf+1, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector2( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }

//Initialization by cloning from A
intVector2(intVector2 A)
 {
  for(int i = 0; i < L; i++)
   F[i] = A.F[i];
 }
}//end of class intVector

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


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
 private static intVector2  freq1, freq2
                                  = p.new intVector2();
 private static intVector2 Common1, Common2
                                  = p.new intVector2();

 //Training  text. A StringBuffer is a heavy-duty String.
```

```
private static StringBuffer EngText1, EngText2,
                              GerText,  randText;

private static  Boolean withBlanks;
//Number of studied common words
private static int nCommon;

private static int nEngText1, nEngText2,
                  nGerText, nText;
private static Boolean printAll;

//Machine to read from the hard disk
private static  Scanner input;

//Generator of random numbers
private static Random r  = new Random();


//**************MAIN**************

public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //To print additional information, change to true
 printAll = false;
 //To include   blank spaces among words
 withBlanks = false;
 //Number of reported most common words
 nCommon = 18;
 System.out.println( "\nLetters per n-gram = " + n );
 System.out.println( "Lenght of the alphabet = " +
                                  lengthAlf);
 System.out.println( "Number of n-grams in vector " +
                              "SelfEng = " + N);
 //Test
 /*
 System.out.println( "Index of AAA = " + index(3,"AAA"));
 System.out.println( "Index of AAÄ = " + index(3,"AAÄ"));
```

```
  System.out.println( "Index of ßßß = " + index(3,"ßßß"));
  */
  readTexts();
  work();
  discriminateText();
 }


 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }

 // File is read
 public  static  StringBuffer readData()
 {
  StringBuffer text =  new StringBuffer("");
  //try and catch pairs
    try
    {
       while ( input.hasNext() )
       {
        text = text.append(input.next());
        if (withBlanks)
        {
         //A blank space is inserted after each word
         text = text.append(" ");
        }
```

```
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

//Text is gotten
private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================
```

```
 //Training text in English
 private static void readEngText1()
 {
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
   "ImmuneDistances/BookDelight.txt";
  EngText1 =  getText(path);
  String s = EngText1.substring(0, 100);
  System.out.println(s);
 }

 //Reading  a second  text in English.
 private static void readEngText2()
 {
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
           "ImmuneDistances/SamuelButler.txt";
  EngText2 = getText(path);
 }

 //Reading a text in German
 private static void readGerText()
 {
   String path = "/home/jose/jose/AAjoser/Ciencia/" +
            "ImmuneDistances/DieGKomoedie.txt";
   GerText = getText(path);
 }

//A random text  with l chars is composed
 private static StringBuffer composeText(int L)
 {
  randText =  new StringBuffer("");

  for(int i = 0; i < L; i++)
  {
   //a random integer number
   int l = r.nextInt(lengthAlf);
   //A random char
```

```
   char c = Alphabet.charAt(l);
   //char is appended to text
   randText = randText.append(c);
  }
  return randText;
 }


 private static void readTexts()
 {
  if (withBlanks)
{
 Alphabet = Alphabet + ' ';
 N = Alphabet.length();
}
  readEngText1();
  nEngText1 = EngText1.length();
  System.out.println( "Lenght of trainingText = " +
                               nEngText1);
  readEngText2();
  nEngText2= EngText2.length();
  System.out.println( "Lenght of testerEng = " +
                               nEngText2);
  readGerText();
  nGerText = GerText.length();
  System.out.println( "Lenght of testerGer = " +
                               nGerText);
}

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



 //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;
  correct = false;
 }
   }
  if (printAll)
  System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
  return ind;
 }


 //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);
 c1 = Alphabet.charAt(p);
 if (withBlanks)
  {
   //Replace blank by a visible symbol
   if ( c1 == ' ') c1 = (int) (765);
  }
 indl = indl - p*k;
}
else c1 = 'A';
//System.out.println("c1 = " + c1);
ngram =  ngram+c1 ;

   }
  /*
  if (printAll)
  System.out.println( "ngram = " + ngram +
                   " index = "   + ind);*/
  return ngram;
 }


 /*
   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 intVector2 nGramsFrequency(int n,
```

```
                          StringBuffer Text)
 {
  //Zeroed initialization
  intVector2 F = p.new intVector2();
  nText = Text.length();
  int start = 0;
  int ind = 0;
  for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
 start = start +1;
   }
  return F;
 }



 //Returned vector contains the indexes of most frequent n-grams
 private static intVector2 commons(int n, intVector2 F)
 {
  intVector2 h = p.new intVector2(F);
  intVector2 C = p.new intVector2();
  int Max;
  String s;
  for(int counter = 0; counter < nCommon; counter++)
  {
   Max = 0;
   //Find the max
   for(int i = 0; i < N; i++)
if (h.F[i] > h.F[Max] )  Max = i;
   //n-gram associated to Max
```

```
   s = indexInverse(n, Max);
   //System.out.print(s + " ");
   System.out.println(s + "("+ Max + ") " +
        "Freq = " + h.F[Max] + " ");

   //The max leaves the game
   h.F[Max] = 0;
   //The Max is recorded
   C.F[counter] = Max;
  }
  return C;
 }

 //The symmetric distance among two sets
 // of integers C1 and C2 is calculated.
 //Both sets have the same number of elements = nCommon.

 private static int distance(int nCommon,
               intVector2 C1, intVector2 C2)
 {

  for(int i = 0; i < nCommon; i++)
   {
for(int j = 0; j < nCommon; j++)
 {
  //A coincidence  is deleted
  //(an index appears in both vectors)
  if (C1.F[j] == C2.F[i])
   {
 C1.F[j] = -1;
  C2.F[i] = -1;

   }
 }
   }
  int d = 0;
  System.out.println("Elements in A but not in B " +
   "with their frequencies");
```

```
  int counter1 = 0;
  for(int i = 0; i < nCommon; i++)
  if (C1.F[i] > 0)
  {
   d = d+1;
   counter1++;
   System.out.println(counter1 + " " + indexInverse(n,C1.F[i])
   + "  " + freq1.F[C1.F[i]]);
  }
  System.out.println("Elements in B but not in A " +
   "with their frequencies");
  int counter2 = 0;
for(int i = 0; i < nCommon; i++)
  if (C2.F[i] > 0)
      {
   d = d+1;
   counter2++;
   System.out.println(counter2 + " " + indexInverse(n,C2.F[i])
   + "  " + freq2.F[C2.F[i]]);
  }
  return d;
 }

 //The correctness of the program hangs
 //on the index and indexInverse functions.
 //One is the inverse of the other.
 //Test at random
 private static void runTest1()
 {
  //Test for index functions
  System.out.println("\nTest for index functions. index()"
              +   "\nand indexInverse() must" +
              " be inverse one to another");
  System.out.println("Initial ngram  + " +
       "\t Initial index + \tFinal ngram  \n");
  for(int i = 0; i < 1000; i++)
   {
String si = composeText(3).toString();
```

```
System.out.print( si + "\t");
int ind1 = index(3,si);
System.out.print( ind1 + "\t");
String  sf = indexInverse(3, ind1);
System.out.print(  sf+ "\t");
if (si.equals(sf)) System.out.print( "Equal n-grams ");
   else  System.out.print( "Different n-grams " );
int ind2 = index(3,sf);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.println(" Equal indexes ");
   else  System.out.println( "Different indexes " );
   }
 }

 //The correctness of the program hangs
 //on the index and indexInverse functions.
 //One is the inverse of the other.
 //Test on demand.
 private static void runTest2()
 {
  //Test for index functions
  System.out.println("\nTest for index functions. index()"+
  "\n and indexInverse() must be inverse one to another");


//String si = "ßßß";
String si = "ßZn";
System.out.print( si + "\t");
int ind1 = index(3,si);
System.out.print( ind1 + "\t");
String  sf = indexInverse(3, ind1);
System.out.print(  sf+ "\t");
if (si.equals(sf)) System.out.println("Equal n-grams");
   else  System.out.println( "Different n-grams" );
int ind2 = index(3,sf);
System.out.print( ind2 + "\t");
if (ind1 ==ind2) System.out.println( " Equal indexes");
else  System.out.println( "Different indexes" );
```

```
ind1 = 151393;
System.out.print( ind1 + "\t");
 si = indexInverse(3, ind1);
System.out.print(  si+ "\t");
ind2 = index(3,si);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.println(" Equal indexes ");
else  System.out.println( "Different indexes" );
sf = indexInverse(3, ind2);
System.out.print( sf + "\t");
if (si.equals(sf)) System.out.println("Equal n-grams ");
   else  System.out.println( "Different n-grams" );
ind1 = 151392;
System.out.print( ind1 + "\t");
 si = indexInverse(3, ind1);
System.out.print(  si+ "\t");
ind2 = index(3,si);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.print(" Equal indexes ");
else  System.out.print( "Different indexes" );
sf = indexInverse(3, ind2);
System.out.print( sf + "\t");
if (si.equals(sf)) System.out.println("Equal n-grams");
   else  System.out.println( "Different n-grams" );
ind1 = 151394;
System.out.print( ind1 + "\t");
 si = indexInverse(3, ind1);
System.out.print(  si+ "\t");
ind2 = index(3,si);
System.out.print( ind2 + "\t");
if (ind1 == ind2) System.out.print( " Equal indexes ");
else  System.out.println( "Different indexes" );
sf = indexInverse(3, ind2);
System.out.print( sf + "\t");
if (si.equals(sf)) System.out.print( "Equal n-grams ");
   else  System.out.println( "Different n-grams" );
```

```
 }

 //List of all n-grams and their indexes
 private static void runTest3()
 {
  for(int i = 000; i < 1000; i++)
   {
String si = indexInverse(3,i);
int ind = index(n,si);
System.out.println(i + " "  +  si  + " " + ind);
   }
 }

 //Operative part
 private static void work()
 {
  //A test for correctness is done.
  Boolean Test = false;
  if (Test) runTest3();
  else
   {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}
    //The frequency of n.grams in EngText1 is calculated
    freq1 = nGramsFrequency(n, EngText1);
    String nGram = "and";
    int ind = index(n,nGram);
    /*
    System.out.println("\nTest: Frequency in EngText1 " +
     "of n-gram " +  nGram + " = " + freq1.F[ind]);
     */
    //The nCommon most common n-grams are picked up
    System.out.println("\nA = " + nCommon + " most common "
                        + n +
              "-grams with  their indexes " +
```

```
            "\nand frequencies in EngText1");
   //The indexes of most common n-grams in EngText1
   //are recorded in Coomon1
   Common1 = commons(n,freq1);
   /*freq2 = study(n, EngText2);

   System.out.println("\n" + "B = " +  nCommon +
              " most common " + n
             + "-grams with their indexes  " +
          "and frequencies in EngText2"); */
   //The frequency of n.grams in GerText is calculated
   freq2 = nGramsFrequency(n, GerText);
   /*
   System.out.println("\nTest: Frequency in GerText" +
    " of n-gram " +   nGram + " = " + freq2.F[ind]);
    */
   //The nCommon most common n-grams are picked up
   System.out.println("\n" + "B = " +  nCommon +
         " most common " + n
        + "-grams with their indexes  " +
     "\nand frequencies in the German text");
   //The indexes of most common n-grams in EngText1
   //are recorded in Common2
   Common2 = commons(n,freq2);
   System.out.println("\nList of not coincident n-grams");
   int d;
    d = distance(nCommon, Common1, Common2);
    System.out.println("The symmetric distance between " +
     "the two texts is " + d);
    System.out.println(" for the " + nCommon +
     " most common " +
     n + "-grams" );

 }//end of else
}

 //The language in which a given text  is
//written is discriminated: English else German
```

```
 private static void discriminate(String text)
 {
  System.out.println("\nText to discriminate");
  System.out.println(text + "\n");
  int counterEnglish = 0;
  int counterGerman = 0;
  int m = text.length();
  int start = 0;
  boolean isInEnglish = false;
  boolean isInGerman = false;
  for(int c=0; c < m-n+1; c++)
   {
 String ngram =  text.substring(start,start+n);
 System.out.print(ngram);
 for (int i = 0; i < 18; i++)
  {
if (indexInverse(n,Common1.F[i]).equals(ngram))
  {
counterEnglish++;
isInEnglish   = true;
  }
if (indexInverse(n,Common2.F[i]).equals(ngram))
  {
counterGerman++;
isInGerman   = true;
  }
  }
 if (isInEnglish  == true)
 System.out.print(" very common in English ");
 else System.out.print(" not very common in English ");
 if (isInGerman  == true)
 System.out.print(" very common in German ");
 else System.out.print(" not very common in German ");
 System.out.println();
 start = start +1;
 isInEnglish = false;
 isInGerman = false;
   }
```

```
  System.out.println("Number of nGrams in text = " + m);
  System.out.println("Number of nGrams in English only = " +
                                        counterEnglish);
  System.out.println("Number of nGrams in German  only= " +
             counterGerman);
  if (counterEnglish > counterGerman)
  System.out.println("\nThe text is written in English");
  else System.out.println("\nThe text is written in German");
 }



 //Discriminating Manager
 private static void discriminateText()
 {
  String text1 = "A historian says Saudia Arabia's " +
   "\ndemolition of an Ottoman Empire-era portico " +
   "\nat Mecca, Islam's holiest site, to make way " +
   "\nfor more pilgrims is cultural vandalism" +
   "\n -- and that ultra-conservative Wahhabism" +
   "\n is partly to blame. " ;
  String text2 = "Außenminister Laurent Fabius sagte" +
   "\n in Paris, der Militäreinsatz solle ab April " +
   "\nunter der Führung der Vereinten Nationen laufen";
  //Zeroed initialization
  discriminate(text2);
  //discriminate(text2);
 }
}//end of main class I53 SDDiscriminator
```

Problems of Chapter 3

61, pagerefI61. A possible code for a test is the following:

```
/*Program I61 EuclideanDistanceTest
```

```
 A test to the  task below is carried.
 To that aim, the program is tested over
 a pair of  simple strings.

 The program calculates the Euclidean n-gram distance
 between two given small strings.
 The answer is known beforehand, so this is a test.

 The program has three steps:

 Step one:
 The frequencies of n-grams captured from a given text
 defines an n-tuple, an element of the form
 (a1,a2,...,an).  The whole operation projects the text
 into Rˆn, the set of n-tuples.
 This operation is done with the help
 of an indexing procedure.

 Step two:
 Two texts are given for examination.
 All n-grams of each text are captured
 and their absolute
 frequencies are recorded.

 Step three:
 The Euclidean n-gram distance between the two given texts
 is defined as the  Euclidean distance between
 the n-tuples of absolute frequencies of n-grams.


*/



import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
```

```
import java.lang.IllegalStateException;
import java.util.Scanner;

public class EuclideanDistanceTest
{
 //The application is instantiated
 static EuclideanDistanceTest p = new EuclideanDistanceTest();


 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n=3;
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);


 //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector4
{
int L = (int) Math.pow(lengthAlf+1, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector4( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }
```

```
}//end of class intVector

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


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
 private static intVector4  freq1, freq2
                                 = p.new intVector4();


 //Training  text. A StringBuffer is a heavy-duty String.
 private static StringBuffer EngText1, EngText2,
                               GerText;

 private static  Boolean withBlanks;


 private static int nEngText1, nEngText2,
                    nGerText, nText;
 private static Boolean printAll;

 //Machine to read from the hard disk
 private static  Scanner input;

 //Test with simple strings
 private static boolean test = false;

 private static StringBuffer s1,s2;

 //**************MAIN*************

 public static void main( String args[] )
 {
  System.out.println("Please, wait for a while");
  //To print additional information, change to true
```

```
  printAll = true;
  //To include   blank spaces among words
  withBlanks = false;

  System.out.println( "\nLetters per n-gram = " + n );
  System.out.println( "Lenght of the complete alphabet = " +
                                    lengthAlf);
  System.out.println( "Number of n-grams in vector " +
                                  "SelfEng = " + N);
  //The test is included
  test = true;
  if (test)  readTextsTest();
   else readTexts();
  work();
 }

//A test over simple pairs of strings is carried out
private  static  void readTest()
{
 //Very short alphabet
 Alphabet = "ab";
 lengthAlf = Alphabet.length();
 //Number of n-grams
 N = (int) Math.pow(lengthAlf, n);
 System.out.println( "Number of n-grams short alphabet = " + N);
 s1 = new StringBuffer ( "aaaaaaaaaaaaaaaaa" ) ;
 s2 = new StringBuffer ( "aaaaababaabbbaababbbabbb" ) ;
}

private static void readTextsTest()
{
 readTest();
 EngText1 = s1;
 GerText = s2;
 nEngText1 = EngText1.length();
 System.out.println( "EngText1 = " + EngText1 +
 ", Lenght  = " +
                                  nEngText1);
```

```
 nGerText = GerText.length();
 System.out.println( "GerText = " + GerText +
 ", Lenght = " +
                                 nGerText);
}



 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
       System.err.println( "File does not exists." );
       System.exit( 1 );
    }
 }

 // File is read
 public  static  StringBuffer readData()
 {
  StringBuffer text =  new StringBuffer("");
  //try and catch pairs
    try
    {
       while ( input.hasNext() )
       {
        text = text.append(input.next());
        if (withBlanks)
        {
         //A blank space is inserted after each word
         text = text.append(" ");
        }
```

```
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
{
   if ( input != null )
      input.close(); //
}

//Text is gotten
private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================
```

```
 //Training text in English
 private static void readEngText1()
 {
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
            "ImmuneDistances/BookDelight.txt";
  EngText1 =  getText(path);
  String s = EngText1.substring(0, 100);
  System.out.println(s);
 }

 //Reading  a second  text in English.
 private static void readEngText2()
 {
  String path = "/home/jose/jose/AAjoser/Ciencia/" +
           "ImmuneDistances/SamuelButler.txt";
  EngText2 = getText(path);
 }

 //Reading a text in German
 private static void readGerText()
 {
   String path = "/home/jose/jose/AAjoser/Ciencia/" +
            "ImmuneDistances/DieGKomoedie.txt";
   GerText = getText(path);
 }




 private static void readTexts()
 {
  if (withBlanks)
{
 Alphabet = Alphabet + ' ';
 N = Alphabet.length();
}
```

```
  readEngText1();
  nEngText1 = EngText1.length();
  System.out.println( "Lenght of trainingText = " +
                                  nEngText1);
  readEngText2();
  nEngText2= EngText2.length();
  System.out.println( "Lenght of testerEng = " +
                                  nEngText2);
  readGerText();
  nGerText = GerText.length();
  System.out.println( "Lenght of testerGer = " +
                                  nGerText);
}

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



 //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 intVector4 nGramsFrequency(int n,
                        StringBuffer Text)
{
 //Zeroed initialization
 intVector4 F = p.new intVector4();
 nText = Text.length();
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
start = start +1;
```

```
   }
  return F;
 }


 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
              intVector4 vect1, intVector4 vect2)
 {
  double sum = 0;
  double d = 0;
  if (printAll)
System.out.println("Frequency vectors to compare");
  for(int i = 0; i < N; i++)
   {
  long a = vect1.F[i]-vect2.F[i];
sum = sum + a*a;
d = Math.sqrt(sum);
if (printAll)
System.out.println(vect1.F[i] + "  " + vect2.F[i]);
   }
  return d;
 }

 //Operative part
 private static void work()
 {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}
    //Frequencies of n-grams in EngText1 are calculated
if (printAll)
 System.out.println("\nn-gram analysis of EngText1");
    freq1 = nGramsFrequency(n, EngText1);
    if (printAll)
```

```
     {
     System.out.println("Frequencies of n-grams");
      for(int i = 0; i < N; i++)
    System.out.println(freq1.F[i]);
      System.out.println();
     }
    System.out.println("\nn-gram analysis of GerText");
    freq2 = nGramsFrequency(n, GerText);
    if (printAll)
    {
    System.out.println("Frequencies of n-grams");
     for(int i = 0; i < N; i++)
  System.out.println(freq2.F[i]);
     System.out.println();
    }
    //Euclidean distance between freq1 and freq2 is calculated
     double d = euclideanDistance(N, freq1,freq2);
     System.out.println("\nThe Euclidean n-gram distance between " +
      "the two texts is " + d);
 }

}//end of main class I61 EuclideanDistanceTest
```

62, pag. 115.

```
/*Program I62 EuclideanDistanceFull

This program calculates the Euclidean n-gram distance
 between two given texts.

 The program includes the possibility to make a test
 over a pair of  simple strings.

 The program also allows the synthesis of random texts,
 composed as concatenation of chars picked at random
 from the given alphabet.
```

```
 The program has three steps:

 Step one:
 The frequencies of n-grams captured from a given text
 defines an n-tuple, an element of the form
 (a1,a2,...,an).  The whole operation projects the text
 into Rˆn, the set of n-tuples.
 This operation is done with the help
 of an indexing procedure.

 Step two:
 Two texts are given for examination.
 All n-grams of each text are captured
 and their absolute
 frequencies are recorded.

 Step three:
 The Euclidean n-gram distance between the two given texts
 is defined as the  Euclidean distance between
 the n-tuples of absolute frequencies of n-grams.

*/



import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.util.Random;
import java.lang.IllegalStateException;
import java.util.Scanner;

public class EuclideanDistanceFull
{
 //The application is instantiated
 static EuclideanDistanceFull p = new EuclideanDistanceFull();
```

```
 //English + German chars, only the most informative
 private static String Alphabet =
                   "AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ" +
  "aäbcdefghijklmnoöpqrstuüvwxyzß";
//length of the Alphabet
 private static  int lengthAlf = Alphabet.length();
//letters per n-gram
 private static  int n=3;
//Number of n-grams
 private static  int N = (int) Math.pow(lengthAlf, n);


 //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector5
{
int L = (int) Math.pow(lengthAlf+1, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector5( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector

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


 //The frequency of appearance of n-grams in the text.
 //If n>3,  the assigned memory must be expanded
```

```
private static intVector5  freq1, freq2
                                = p.new intVector5();


//Training  text. A StringBuffer is a heavy-duty String.
private static StringBuffer EngText1, EngText2,
                            GerText,  randomText;

private static  Boolean withBlanks;


private static int nEngText1, nEngText2,
                   nGerText, nText;
private static Boolean printAll;

//Machine to read from the hard disk
private static  Scanner input;

//Test with simple strings
private static boolean test = false;

private static StringBuffer s1,s2;


//Generator of random numbers
private static Random r  = new Random();

//**************MAIN**************

public static void main( String args[] )
{
 System.out.println("Please, wait for a while");
 //To print additional information, change to true
 printAll = false;
 //To include   blank spaces among words
 withBlanks = false;

 System.out.println( "\nLetters per n-gram = " + n );
```

```
  System.out.println( "Lenght of the complete alphabet = " +
                                      lengthAlf);
  System.out.println( "Number of n-grams in vector " +
                                "SelfEng = " + N);
  //The test is included
  test = false;
  if (test)  readTextsTest();
   else readTexts();
  int lengthRandomText = 10000;
  System.out.println( "Length of randomText = "
                               + lengthRandomText);
  randomText = composeText(lengthRandomText);
  work();
 }

//A test over simple pairs of strings is carried out
private  static  void readTest()
{
 //Very short alphabet
 Alphabet = "ab";
 lengthAlf = Alphabet.length();
 //Number of n-grams
 N = (int) Math.pow(lengthAlf, n);
 System.out.println( "Number of n-grams short alphabet = " + N);
 s1 = new StringBuffer ( "aaaaaaaaaaaaaaaaa" ) ;
 s2 = new StringBuffer ( "aaaaababaabbbaababbbabbb" ) ;
}



//A random text  with l chars is composed
private static StringBuffer composeText(int L)
{
StringBuffer randText =  new StringBuffer("");

for(int i = 0; i < L; i++)
{
 //a random integer number
```

```
 int l = r.nextInt(lengthAlf);
 //A random char
 char c = Alphabet.charAt(l);
 //char is appended to text
 randText = randText.append(c);
}
return randText;
}



private static void readTextsTest()
{
 readTest();
 EngText1 = s1;
 GerText = s2;
 nEngText1 = EngText1.length();
 System.out.println( "EngText1 = " + EngText1 +
 ", Lenght  = " +
                                  nEngText1);
 nGerText = GerText.length();
 System.out.println( "GerText = " + GerText +
 ", Lenght = " +
                                  nGerText);
}



 // File is opened
private  static  void openFile(String path)
 {
  //try and catch pair
    try
    {
       input = new Scanner( new File( path ) );
    }
    catch ( FileNotFoundException fileNotFoundException )
    {
```

```
      System.err.println( "File does not exists." );
      System.exit( 1 );
   }
}

// File is read
public  static  StringBuffer readData()
{
 StringBuffer text =  new StringBuffer("");
 //try and catch pairs
   try
   {
      while ( input.hasNext() )
      {
       text = text.append(input.next());
       if (withBlanks)
       {
        //A blank space is inserted after each word
        text = text.append(" ");
       }
      }
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}

// File is closed
private static void closeFile()
```

```
{
   if ( input != null )
      input.close(); //
}

//Text is gotten
private static StringBuffer getText(String path)
{
StringBuffer text =  new StringBuffer("");
  openFile(path);
  text = readData();
  closeFile();
  return text;
}


//================================================
//============PATH TO FILES WITH BOOKS============
//====Make the changes through your own paths=====
//================================================


//Training text in English
private static void readEngText1()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
           "ImmuneDistances/BookDelight.txt";
 EngText1 =  getText(path);
 String s = EngText1.substring(0, 100);
 System.out.println(s);
}

//Reading  a second  text in English.
private static void readEngText2()
{
 String path = "/home/jose/jose/AAjoser/Ciencia/" +
          "ImmuneDistances/SamuelButler.txt";
 EngText2 = getText(path);
```

```
 }

 //Reading a text in German
 private static void readGerText()
 {
   String path = "/home/jose/jose/AAjoser/Ciencia/" +
            "ImmuneDistances/DieGKomoedie.txt";
   GerText = getText(path);
 }




 private static void readTexts()
 {
  if (withBlanks)
{
 Alphabet = Alphabet + ' ';
 N = Alphabet.length();
}
  readEngText1();
  nEngText1 = EngText1.length();
  System.out.println( "Lenght of trainingText = " +
                               nEngText1);
  readEngText2();
  nEngText2= EngText2.length();
  System.out.println( "Lenght of testerEng = " +
                               nEngText2);
  readGerText();
  nGerText = GerText.length();
  System.out.println( "Lenght of testerGer = " +
                               nGerText);
}

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

```
//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 intVector5 nGramsFrequency(int n,
                          StringBuffer Text)
 {
  //Zeroed initialization
  intVector5 F = p.new intVector5();
  nText = Text.length();
  int start = 0;
  int ind = 0;
  for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
 start = start +1;
   }
  return F;
 }


 //The Euclidean distance among two vectors of integer type
 //(declared as an object intVector3) is calculated
 private static double euclideanDistance(int N,
               intVector5 vect1, intVector5 vect2)
 {
  double sum = 0;
  double d = 0;
  if (printAll)
System.out.println("Frequency vectors to compare");
  for(int i = 0; i < N; i++)
   {
  long a = vect1.F[i]-vect2.F[i];
sum = sum + a*a;
```

```
d = Math.sqrt(sum);
if (printAll)
System.out.println(vect1.F[i] + "  " + vect2.F[i]);
   }
  return d;
 }

 //Operative part
 //Operative part
 private static void work()
 {
if (withBlanks)
{
 char c = (int) (765);
 System.out.println("Each blank is replaced by " + c);
}
//EngText1 vs GerText
    //Frequencies of n-grams in EngText1 are calculated
if (printAll)
 System.out.println("\nn-gram analysis of EngText1");
    freq1 = nGramsFrequency(n, EngText1);
    if (printAll)
     {
     System.out.println("Frequencies of n-grams");
      for(int i = 0; i < N; i++)
    System.out.println(freq1.F[i]);
      System.out.println();
     }
    if (printAll)
    System.out.println("\nn-gram analysis of GerText");
    freq2 = nGramsFrequency(n, GerText);
    if (printAll)
    {
    System.out.println("Frequencies of n-grams");
     for(int i = 0; i < N; i++)
  System.out.println(freq2.F[i]);
     System.out.println();
    }
```

```
    //Euclidean distance between freq1 and freq2
     double d = euclideanDistance(N, freq1,freq2);
     System.out.println("\nThe Euclidean n-gram distance" +
      " between the two texts, \nEngText1 and " +
      "GerText, is " + d);
  //EngText1 vs RandomText
     //Frequencies of n-grams in EngText1 are calculated
  if (printAll)
   System.out.println("\nn-gram analysis of EngText1");
     freq1 = nGramsFrequency(n, EngText1);
     if (printAll)
      {
      System.out.println("Frequencies of n-grams");
       for(int i = 0; i < N; i++)
     System.out.println(freq1.F[i]);
       System.out.println();
      }
     if (printAll)
     System.out.println("\nn-gram analysis of randomText");
     freq2 = nGramsFrequency(n, randomText);
     if (printAll)
     {
     System.out.println("Frequencies of n-grams");
      for(int i = 0; i < N; i++)
    System.out.println(freq2.F[i]);
      System.out.println();
     }
     //Euclidean distance between freq1 and freq2
      d = euclideanDistance(N, freq1,freq2);
      System.out.println("\nThe Euclidean n-gram " +
       "distance  between the two texts, " +
       "\nEngText1 and randomText,  is " + d);
 }

}//end of main class I62 EuclideanDistanceFull
```

Problems of Chapter 4

78 pag. 137 The next code reads a genome from a file and executes an n-gram analysis:

```
//Program I78 GenomeNGram
//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content
//and carries an n-gram analysis.

//Fasta .faa format required.

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;


public class GenomeNGram
{

 //The application is instantiated
 private  static GenomeNGram p =
                     new GenomeNGram();
  private static  Scanner input;

  private static String nameOfFolder =
  "/home/jose/jose/AAjoser/Ciencia/" +
            "GenomesComplete/GenomesSample/";
  private static String nameOfFile =
  "Arcobacter_L_uid158135.faa";

  private static  String path = nameOfFolder + nameOfFile;

 private static StringBuffer proteome = new StringBuffer("");
```

```
  //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);

  private static boolean print1 = false;
  private static boolean print2 = false;


  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector6
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector6( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector
```

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

//The frequency of appearance of n-grams in the text.
//If n>3,  the assigned memory must be expanded
private static intVector6  freq1
                                = p.new intVector6();


   // We read  record from file
   public  static  StringBuffer readData(String path)
                         throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
       {
       //Filtering of first line of each subsequence
       if (s.charAt(0) != '>' )
     {
       System.out.println(s);
       text = text.append(s);
     }
       }

       fr.close();
      }
      catch ( NoSuchElementException elementException )
      {
         System.err.println( "File is corrupted." );
         input.close();
         System.exit( 1 );
      }
```

```
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}


//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 (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}
```

```
//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 ;

  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                    " index = "   + ind);
 return ngram;
}


/*
```

```
    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 intVector6 nGramsFrequency(int n,
                            StringBuffer Text)
  {
   //Zeroed initialization
   intVector6 F = p.new intVector6();
   int nText = Text.length();
   int start = 0;
   int ind = 0;
   for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
   start = start +1;
    }
   return F;
  }



  public static void main(String[] args) throws IOException
  {
print1 = false;
print2 = true;
```

```
 System.out.println("n-gram analysis of " + nameOfFile);
 readData(path);
 proteome = readData(path);
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  System.out.println("Length of alphabet = "
             + Alphabet.length());
 if (print2)
 {
System.out.println("n-gram analysis of " + nameOfFile);
System.out.println("length of proteome = " +
proteome.length());
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
                + "\t" + freq1.F[i]);
  }
   }
}//End of main class I78 GenomeNGram
```

82, 146. The next code uses a JFrame and a JScrollPane to allow the scrolling of the large image that is output by the program when n = 5.

```
//Program I82 nGramVisor2
//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content,
//carries an n-gram analysis and
//makes a graphic report
//according to a coloring code.

//Fasta .faa format is required.


import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
```

```
import java.awt.Graphics2D;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;




public class nGramVisor2 extends JFrame
{

 private static final long serialVersionUID = 1L;


 //========Fundamental parameters========

//Modify them as it pleases you:

//letters per n-gram
private static  int n = 3;
//Width of pixel
private static int widthOfPixel = 5;
//Number of pixels per row
int pixelsPerRow = 120;
//Width of frame
private static int widthOfFrame = 700;
//Height of frame
private static int heightOfFrame = 800;
//Directory where your files are kept into
```

```
private static String nameOfFolder =
"/home/jose/jose/AAjoser/Ciencia/" +
         "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";


private static boolean print1 = false;
private static boolean print2 = false;



//======= End of fundamental parameters



//Constructor of main class:
//Primary instructions for initialization
public nGramVisor2()
{
 //Title
 super("n-gram visor");
 setSize(widthOfFrame, heightOfFrame);
 //Enabling exit by clicking the terminator icon
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 //Inner painting class is added
 getContentPane().add(new JScrollPane(new Painter()));
}

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

```
 //Instantiation of
 nGramVisor2 visor2 = new nGramVisor2();
 visor2.setVisible(true);
}



//========== INNER DRAWING CLASS ==============


class Painter extends JPanel
{

 private static final long serialVersionUID = 1L;

 //Constructor = initialization
 public Painter ()
  {
   setSize(getPreferredSize());
   Painter.this.setBackground(Color.white);
  }

 //Size of the drawing area
 public Dimension getPreferredSize()
 {
  return new Dimension(widthOfFrame, heightOfFrame);
 }

 //Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.


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

```
  Graphics2D g2d = (Graphics2D) g;
  try
   {
    //n-gram frequencies are found and kept in freq1
findFrequencies();
   }
  catch (IOException e)
   {
e.printStackTrace();
   }

 int max = findMax();
 System.out.println("Scale goes from 0 to " + max);
 System.out.println("Code for intensity from less to more" +
": \n black (absence), gray, blue, cyan, yelow," +
" orange, pink, red");
 int nPoints = freq1.F.length;
 boolean go = true;
 int nRows = nPoints / pixelsPerRow +1;
 int counter = 0;
 for (int i = 0; (i < nRows) & go; i++)
  {
   for (int j = 0; (j < pixelsPerRow) & go; j++)
    {
     int index = i*pixelsPerRow + j;
     //Everything begins in black = absence
     g2d.setColor(Color.BLACK);
     //Colors are acquired as frequency increases
     if ( freq1.F[index] >  0)
        g2d.setColor(Color.GRAY);
     if ( freq1.F[index] >  max/7)
        g2d.setColor(Color.BLUE);
     if ( freq1.F[index] >  2* max/7)
        g2d.setColor(Color.CYAN);
     if ( freq1.F[index] >  3* max/7)
        g2d.setColor(Color.YELLOW);
     if ( freq1.F[index] >  4* max/7)
        g2d.setColor(Color.ORANGE);
```

```
     if ( freq1.F[index] >  5* max/7)
        g2d.setColor(Color.PINK);
     if ( freq1.F[index] >  6* max/7)
        g2d.setColor(Color.RED);
     //Pixel is drawn
     g2d.fillOval( j*widthOfPixel,  i*widthOfPixel,
      widthOfPixel, widthOfPixel );
     if (counter >= nPoints-1) go = false;
     counter = counter+1;
     //System.out.println("index = " + index );
    }
   }
}//End of paintComponent

}//End of Painter

//======== End of inner class Painter =======




 //======== Operative part ============


  //Machine for reading from hard disc
  private static  Scanner input;

  //Path to file
  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer proteome =
                        new StringBuffer("");
```

```
  //length of the Alphabet
  private static  int lengthAlf = Alphabet.length();

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



  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector8
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector8( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector


//========End of inner class intVector======


//Instantiation of nGramVisor needed
//because of inner classes
```

```
private  static nGramVisor2 x =  new nGramVisor2();

private static intVector8  freq1
                                = x.new intVector8();



//This is the first method to be executed
public void init()
{

 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
         + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);
 print1 = false;
 print2 = false;
}


public static void findFrequencies() throws IOException
{
 readData(path);
 proteome = readData(path);
 System.out.println("length of proteome = "
         + proteome.length());
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  {
 System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
                + "\t" + freq1.F[i]);
  }
 findMax();
```

```
}


   // We read  record from file
   public  static  StringBuffer readData(String path)
                        throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
       {
       //Filtering of first line of each subsequence
       if (s.charAt(0) != '>' )
     {
       //System.out.println(s);
       text = text.append(s);
     }
       }

       fr.close();
      }
      catch ( NoSuchElementException elementException )
      {
         System.err.println( "File is corrupted." );
         input.close();
         System.exit( 1 );
      }
      catch ( IllegalStateException stateException )
      {
         System.err.println( "Reading aborted." );
         System.exit( 1 );
      }
```

```
   return text;
}


//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 (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

//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 ;

  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                    " index = "   + ind);
 return ngram;
}


/*
  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 intVector8 nGramsFrequency(int n,
                            StringBuffer Text)
   {
    //Zeroed initialization
    intVector8 F = x.new intVector8();
    int nText = Text.length();
    int start = 0;
    int ind = 0;
    for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
    start = start +1;
     }
    return F;
   }


   private static int findMax()
   {
 int max = 0;
 for(int i = 0; i< freq1.F.length; i++)
 if (freq1.F[i] > max ) max = freq1.F[i];
 return max;
   }


}//End of main class I82 nGramVisor2
```

84, page 146. The immediate reason for the existence of holes resides in the fasta code, which is the following:

```
A  alanine                P  proline
B  aspartate/asparagine   Q  glutamine
C  cystine                R  arginine
D  aspartate              S  serine
E  glutamate              T  threonine
F  phenylalanine          U  selenocysteine
G  glycine                V  valine
H  histidine              W  tryptophan
I  isoleucine             Y  tyrosine
K  lysine                 Z  glutamate/glutamine
L  leucine                X  any
M  methionine             *  translation stop
N  asparagine             -  gap of indeterminate length
```

This code has special symbols X * - that in general might be absent of the sequence list of many proteins. On the other hand, most proteins have sequences based on the ordinary amino acids of the genetic code. Nevertheless, some amino acids might undergo biochemical modification after their synthesis in the ribosomes. The role of these symbols as hole generators can be unveiled if only one change the alphabet to the next one:

Alphabet = ”ACDEFGHIKLMNPQRSTVWY”;

From the full alphabet we have deleted the symbols: B (aspartate/asparagine), U (selenocysteine) and Z (glutamate/glutamine).

With this shorter alphabet large connected holes disappear for n = 2 and 3. For n= 4 or 5 we have a complex chart that deserves better study. This is one more reason for thinking of better tools. Nevertheless, large connected holes is what one must expect for large values of n: the number of possible n-grams grows as l^n while the number of actual n-grams in a proteome is bounded above by the length of the genome, most possibly some megabytes.

85 pag. 146. The code that prints absent or n-grams with a given absolute frequency follows. It also reports a frequency table of frequency ranges.

```
//Program I85 nGramVisor3
//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content,
//carries an n-gram analysis and
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//might also  be reported
//together with a frequency table of frequency ranges.


//Fasta .faa format is required.


import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;


public class nGramVisor3 extends JFrame
{

 private static final long serialVersionUID = 1L;
```

```
 //========Fundamental parameters========


//Modify them as it pleases you:

//letters per n-gram
private static  int n = 4;
//Width of pixel
private static int widthOfPixel = 1;
//Number of pixels per row
int pixelsPerRow = 800;
//Width of frame
private static int widthOfFrame = 1000;
//Height of frame
private static int heightOfFrame = 700;
//Directory where your files are kept into
private static String nameOfFolder =
"/home/jose/jose/AAjoser/Ciencia/" +
         "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                   "ACDEFGHIKLMNPQRSTVWY";

private static int rawFreqTable[] = new int[100];
```

```
private static int freqTable[] = new int[100];
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;


//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public nGramVisor3()
{
 //Title
 super("n-gram visor");
 setSize(widthOfFrame, heightOfFrame);
 //Enabling exit by clicking the terminator icon
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 //Inner painting class is added
 getContentPane().add(new JScrollPane(new Painter()));

 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
         + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);
}

public static void main(String[] args)
{
 //Instantiation of
 nGramVisor3 visor3 = new nGramVisor3();
 visor3.setVisible(true);
}



//========== INNER DRAWING CLASS ==============
```

```
class Painter extends JPanel
{

 private static final long serialVersionUID = 1L;

 //Constructor = initialization
 public Painter ()
  {
   setSize(getPreferredSize());
   Painter.this.setBackground(Color.white);
  }

 //Size of the drawing area
 public Dimension getPreferredSize()
 {
  return new Dimension(widthOfFrame, heightOfFrame);
 }

 //Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.


 public void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
  try
   {
    //n-gram frequencies are found and kept in freq1
findFrequencies();
   }
  catch (IOException e)
   {
```

```
e.printStackTrace();
   }

 int max = findMax();
 System.out.println("Frequency scale goes from 0 to " + max);
 System.out.println("Code for intensity from less to more frequency" +
": \n black (absence), white (freq = 1), gray, blue, cyan, yelow," +
" orange, pink, red");
 int nPoints = freq1.F.length;
 boolean go = true;
 int nRows = nPoints / pixelsPerRow +1;
 int counter = 0;
 for(int i = 0; i < 100; i++)
 rawFreqTable[i] = 0;
 for (int i = 0; (i < nRows) & go; i++)
  {
   for (int j = 0; (j < pixelsPerRow) & go; j++)
    {
     int index = i*pixelsPerRow + j;
     if ( freq1.F[index] ==  0)
      {
      g2d.setColor(Color.BLACK);
      rawFreqTable[0]++;
      /*System.out.println(index + "   "
                 + indexInverse(n, index));
                 */
      titlesFreqTable[0] = "freq = 0";
      }

     if ( freq1.F[index] ==  1)
     {
      g2d.setColor(Color.WHITE);
      rawFreqTable[1]++;
      /*System.out.println(index + "   "
                + indexInverse(n, index));
                */
     titlesFreqTable[1] = "freq = 1";
     }
```

```
 //Colors are acquired as frequency increases
 if ( freq1.F[index] >  1)
  {
    g2d.setColor(Color.GRAY);
   rawFreqTable[2]++;
  titlesFreqTable[2] = " 1 < freq <= " +  max/7;
  }
 if ( freq1.F[index] >  max/7)
  {
    g2d.setColor(Color.BLUE);
    rawFreqTable[3]++;
 titlesFreqTable[3] = max/7 + " < freq <= " +  2*max/7;
  }
 if ( freq1.F[index] >  2* max/7)
  {
 g2d.setColor(Color.CYAN);
 rawFreqTable[4]++;
titlesFreqTable[4] =  2*max/7 + " < freq <= " +  3*max/7;
  }
 if ( freq1.F[index] >  3* max/7)
  {
 g2d.setColor(Color.YELLOW);
 rawFreqTable[5]++;
titlesFreqTable[5] =   3*max/7 + "  < freq <= " + 4* max/7;
  }
 if ( freq1.F[index] >  4* max/7)
  {
 g2d.setColor(Color.ORANGE);
 rawFreqTable[6]++;
titlesFreqTable[6] =  4*max/7 + "  < freq <= " +  5*max/7;
  }
 if ( freq1.F[index] >  5* max/7)
  {
 g2d.setColor(Color.PINK);
rawFreqTable[7]++;
titlesFreqTable[7] =  5*max/7 + " < freq <= " +  6*max/7;
```

```
     }
    if ( freq1.F[index] >  6* max/7)
     {
    g2d.setColor(Color.RED);
    rawFreqTable[8]++;
   titlesFreqTable[8] = 6*max/7 + "  < freq <= " +  max;
     }
    //Pixel is drawn
    g2d.fillOval( j*widthOfPixel,  i*widthOfPixel,
     widthOfPixel, widthOfPixel );
    if (counter >= nPoints-1) go = false;
    counter = counter+1;
    //System.out.println("index = " + index );
   }
  }

/*
 System.out.println("rawFrequency table ");
 for(int j = 0; j < 9; j++)
  System.out.println(j + "   " + rawFreqTable[j]);
 */
 freqTable[0] = rawFreqTable[0];
 freqTable[1] = rawFreqTable[1];

 for(int j = 2; j < 8; j++)
 freqTable[j] =
      rawFreqTable[j] - rawFreqTable[j+1];
 freqTable[8] = rawFreqTable[8];

 System.out.println("Frequency table of " + " " + n +  "-grams");
 for(int j = 0; j < 9; j++)
  System.out.println(j +  " " + freqTable[j]
   + " for range  " +  titlesFreqTable[j]);

 //Test
 int sumNGrams = 0;
 for(int j = 0; j < 9; j++)
   sumNGrams = sumNGrams + freqTable[j];
```

```
  System.out.println("Sum =  " + sumNGrams + " " + n +  "-grams"


}//End of paintComponent

}//End of Painter

//======== End of inner class Painter =======




 //======== Operative part ============


  //Machine for reading from hard disc
  private static  Scanner input;

  //Path to file
  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer proteome =
                        new StringBuffer("");




  //length of the Alphabet
  private static  int lengthAlf = Alphabet.length();

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

```
  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector9
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector9( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector


//========End of inner class intVector======


//Instantiation of nGramVisor needed
//because of inner classes

private  static nGramVisor3 x =  new nGramVisor3();

private static intVector9  freq1
                                = x.new intVector9();
```

```
public static void findFrequencies() throws IOException
{
 readData(path);
 proteome = readData(path);
 System.out.println("length of proteome = "
         + proteome.length());
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  {
 System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
                + "\t" + freq1.F[i]);
  }
 findMax();
}


   // We read  record from file
   public  static  StringBuffer readData(String path)
                         throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
       BufferedReader br = new BufferedReader(fr);
       String s = "";

       while((s = br.readLine()) != null)
       {
       //Filtering of first line of each subsequence
       if (s.charAt(0) != '>' )
     {
       //System.out.println(s);
```

```
    text = text.append(s);
  }
    }

    fr.close();
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return text;
}


//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;
  System.out.println("Alien : " + p);
 }
  }
 if (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

//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 ;

  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                   " index = "   + ind);
 return ngram;
}


/*
  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 intVector9 nGramsFrequency(int n,
                         StringBuffer Text)
{
 //Zeroed initialization
 intVector9 F = x.new intVector9();
 int nText = Text.length();
 System.out.println("Length  of text = " + nText);
 int start = 0;
 int ind = 0;
 for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
    start = start +1;
     }
    return F;
   }


   private static int findMax()
   {
 int max = 0;
 for(int i = 0; i< freq1.F.length; i++)
 if (freq1.F[i] > max ) max = freq1.F[i];
 return max;
   }
}//End of main class I85 nGramVisor3
```

87, page 161. The next code adds the possibility to read in the bottom bar the color of the clicked pixel.

```
//Program I87 nGramVisor5
//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content,
//carries an n-gram analysis and
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//might also  be reported
//together with a frequency table of frequency ranges.
//Feedback information is  displayed.
//Scrolling is possible but because of memory demands,
```

```
//it works very poorly.
//The color of the clicked pixel can be read
//at the bottom bar.

//This code is more evolvable than those of
//previous versions.

//Fasta .faa format is required.


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;


import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;


public class nGramVisor5 extends JFrame
                          implements MouseListener
{

 private static final long serialVersionUID = 1L;
```

```
 //========Fundamental parameters========


//Modify them as it pleases you:

//letters per n-gram
private static  int n = 4;
//Width of pixel
private static int widthOfPixel = 1;
//Number of pixels per row
static int pixelsPerRow = 500;
//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 1000;
//Directory where your files are kept into
private static String nameOfFolder =
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                   "ACDEFGHIKLMNPQRSTVWY";
```

```
private static int freqTable[] = new int[100];
private static int max = 0;
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;

private JLabel barMessage;

//Initialization of  color variables
private static Color color = new Color(4);
private static Color OurColors[ ] = {Color.BLACK,
Color.WHITE, Color.GRAY, Color.BLUE, Color.CYAN,
Color.YELLOW, Color.ORANGE, Color.PINK, Color.RED};
private static String NamesColors[ ] = {"BLACK",
"WHITE", "GRAY", "BLUE", "CYAN",
"YELLOW", "ORANGE", "PINK", "RED"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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



//Instantiation of main class
private  static nGramVisor5 x =  new nGramVisor5();

//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public nGramVisor5()
{
 //Title
 //super("n-gram visor");
```

```
 JFrame frame = new JFrame("n-gram visor");
 //Enabling exit by clicking the terminator icon
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setSize(widthOfFrame, heightOfFrame);

 frame.setVisible(true);
 barMessage=new JLabel( "Right-click over a given pixel" );
 frame.add( barMessage, BorderLayout.SOUTH );
 JScrollPane js = new JScrollPane(new Painter());
 //Inner painting class is added

 js.addMouseListener(this);
 frame.getContentPane().add(js);
 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
         + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of n-grams = " + N);

}



public static void main(String[] args)
 {
//Nothing as yet
 }



//========== INNER DRAWING CLASS ==============


static class Painter extends JPanel
{
 private static final long serialVersionUID = 1L;
```

```
 //Constructor = initialization
 public Painter ()
  {

   //setSize(getPreferredSize());
   Painter.this.setBackground(Color.white);
   setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
  }


 //Size of the drawing area
 public Dimension getPreferredSize()
 {
  return new Dimension(widthOfFrame, heightOfFrame);
 }


 //Assigns a frequency range to the n-gram
 //given by index.
 //Builds the frequency table by ranges.
 //counting is a boolean flag for counting
 public static int classification(int index,
                           boolean counting)
 {
  int rangeNumber = -1;
  //Colors are assigned according to frequency
  if ( freq1.F[index] ==  0)
{

 if (counting) freqTable[0]++;
      /*System.out.println(index + "   "
                 + indexInverse(n, index));
                 */
 rangeNumber = 0;

}
```

```
  if ( freq1.F[index] ==  1)
{
  if (counting) freqTable[1]++;
      /*System.out.println(index + "   "
                + indexInverse(n, index));
                */
  rangeNumber = 1;
}


  if (( freq1.F[index] >  1)  &
              ( freq1.F[index] <  max/7))
 {
  if (counting) freqTable[2]++;
  rangeNumber = 2;
  }
  if (( freq1.F[index] >=  max/7)  &
              ( freq1.F[index] <  2* max/7))
  {
  if (counting) freqTable[3]++;
   rangeNumber = 3;
  }
  if (( freq1.F[index] >= 2* max/7)  &
              ( freq1.F[index]  < 3 * max/7))
 {
  if (counting) freqTable[4]++;
  rangeNumber = 4;
 }
  if (( freq1.F[index] >=  3*max/7)  &
             ( freq1.F[index] <  4* max/7))
 {
  if (counting) freqTable[5]++;
  rangeNumber = 5;
 }
  if (( freq1.F[index] >=  4*max/7)  &
             ( freq1.F[index] <  5* max/7))
 {
  if (counting) freqTable[6]++;
```

```
 rangeNumber = 6;
}
 if (( freq1.F[index] >= 5* max/7)  &
 ( freq1.F[index] <  6* max/7))
 {
 if (counting) freqTable[7]++;
 rangeNumber = 7;
 }
 if (( freq1.F[index] >=  6*max/7)  &
         ( freq1.F[index] <=    max))
 {
 if (counting) freqTable[8]++;
  rangeNumber = 8;
 }
 return rangeNumber;
}


public void definitions()
{
 titlesFreqTable[0] = "freq = 0";
 titlesFreqTable[1] = "freq = 1";
 titlesFreqTable[2] = " 1 < freq < " +  max/7;
 titlesFreqTable[3] = max/7 + " <= freq < " +  2*max/7;
 titlesFreqTable[4] = 2*max/7 + " <= freq < " +  3*max/7;
 titlesFreqTable[5] = 3*max/7 + "  <= freq < " + 4* max/7;
 titlesFreqTable[6] = 4*max/7 + "  <= freq < " +  5*max/7;
 titlesFreqTable[7] = 5 *max/7 + " <= freq < " +  6*max/7;
 titlesFreqTable[8] = 6*max/7 + "  <= freq <= " +  max;
}

public void report()
{
 System.out.println("Frequency table of " + " " + n
                                      +  "-grams");
 for(int j = 0; j < 9; j++)
  System.out.println(j +  " " + freqTable[j]
   + " for range  " +  titlesFreqTable[j]
```

```
     + "  " + NamesColors[j]);
  //Test
  int sumNGrams = 0;
  for(int j = 0; j < 9; j++)
    sumNGrams = sumNGrams + freqTable[j];
  System.out.println("Sum =  " + sumNGrams + " " + n
                                    +  "-grams");

 }


//Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.
 public void paintComponent(Graphics g)
 {
  Graphics2D g2d = (Graphics2D) g;
  try
   {
    //n-gram frequencies are found and kept in freq1
findFrequencies();
   }
  catch (IOException e)
   {
e.printStackTrace();
   }
 max = findMax();
 definitions();
 System.out.println("Frequency scale goes from 0 to "
     + max + "\nCode for intensity from less to" +
  " more frequency: \n black (absence), " +
  "white (freq = 1), gray, blue, cyan, yelow," +
" orange, pink, red");
 int nPoints = freq1.F.length;
 boolean go = true;
 int nRows = nPoints / pixelsPerRow +1;
```

```
 int counter = 0;
 boolean counting = true;
 for (int j = 0; (j < nRows) & go; j++)
  {
   for (int i = 0; (i < pixelsPerRow) & go; i++)
    {
     int index = j*pixelsPerRow + i;
     int range = classification(index, counting);
     if (print1)
     System.out.println("index = " + index  + " range = "
     + range  + "Frequency " + freq1.F[index]);
     g2d.setColor(OurColors[range]);
     //Pixel is drawn
     g2d.fillOval( i*widthOfPixel,  j*widthOfPixel,
      widthOfPixel, widthOfPixel );
     if (counter >= nPoints-1) go = false;
     counter = counter+1;
    }
   }
 report();
}//End of paintComponent




}//End of Painter

//======== End of inner class Painter =======



 //======== Operative part ============


  //Machine for reading from hard disc
  private static  Scanner input;
```

```
  //Path to file
  private static  String path = nameOfFolder + nameOfFile;

  private static StringBuffer proteome =
                        new StringBuffer("");
```

```
  //======= Inner class intVector ========
//This inner class defines a vector with
//integer numbers as a new type
private  class intVector10
{
int L = (int) Math.pow(lengthAlf, n);
int F[] = new int[L];

//An instance of intVector can be
//initialized in various ways:

//Automatic zeroed initialization
intVector10( )
  {
   for(int i = 0; i < L; i++)
  F[i] = 0;
  }


}//end of class intVector


//========End of inner class intVector======
```

```
private static intVector10  freq1
                                = x.new intVector10();



public static void findFrequencies() throws IOException
{
 readData(path);
 proteome = readData(path);
 System.out.println("length of proteome = "
        + proteome.length());
 freq1 = nGramsFrequency(n, proteome);
 if (print2)
  {
 System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
 System.out.println(i + "\t" +  indexInverse(n, i)
               + "\t" + freq1.F[i]);
  }
 findMax();
}


   // We read  record from file
   public  static  StringBuffer readData(String path)
                        throws IOException
   {
   StringBuffer text = new StringBuffer("");
  //try and catch gates
      try
      {
       FileReader fr = new FileReader(path);
```

```
      BufferedReader br = new BufferedReader(fr);
      String s = "";

      while((s = br.readLine()) != null)
      {
      //Filtering of first line of each subsequence
      if (s.charAt(0) != '>' )
    {
      //System.out.println(s);
      text = text.append(s);
    }
      }

      fr.close();
     }
     catch ( NoSuchElementException elementException )
     {
        System.err.println( "File is corrupted." );
        input.close();
        System.exit( 1 );
     }
     catch ( IllegalStateException stateException )
     {
        System.err.println( "Reading aborted." );
        System.exit( 1 );
     }
     return text;
  }


  //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;
  System.out.println("Alien : " + p);
 }
  }
 if (print1)
 System.out.println( "ngram = " + ngram + " index = "
                                          + ind);
 return ind;
}

//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 ;

  }
 if (print1)
 System.out.println( "ngram = " + ngram +
                   " index = "   + ind);
 return ngram;
}


/*
  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 intVector10 nGramsFrequency(int n,
                          StringBuffer Text)
{
 //Zeroed initialization
 intVector10 F = x.new intVector10();
 int nText = Text.length();
 System.out.println("Length  of text = " + nText);
```

```
   int start = 0;
   int ind = 0;
   for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
   start = start +1;
    }
   return F;
  }


  private static int findMax()
  {
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
  }


//==========Mouse methods==================


 public void mouseClicked( MouseEvent event )
 {
  int i = (int) Math.floor(event.getX()/widthOfPixel);
  int j = (int) Math.floor(event.getY()/widthOfPixel);
  int index = j*pixelsPerRow + i;
  int freq = freq1.F[index];
  boolean counting = false;
  int range = Painter.classification(index, counting);
```

```
   String nameOfColor = NamesColors[range];

   String nGram = indexInverse(n,index);
   barMessage.setText( String.format(
           "Clicked  at [%d, %d]," +
           " index = %d, freq = %d, n-gram = %s, %s",
           i, j, index, freq, nGram, nameOfColor) );
  }

  public void mousePressed( MouseEvent event ){  }
  public void mouseReleased( MouseEvent event ){  }
  public void mouseEntered( MouseEvent event ){  }
  public void mouseMoved( MouseEvent event ){  }
  public void mouseDragged( MouseEvent event ){  }
  public void mouseExited( MouseEvent event ){  }


}//End of main class I87 nGramVisor5
```

96, page 186. We changed some names as follows:

```
//program I96 nGramVisor7

//This program shows how to control repaint()
//with two buttons.

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
import javax.swing.JPanel;


public class nGramVisor7 extends JPanel
 {
  private static final long serialVersionUID = 1L;
private boolean colorFlag;

    public nGramVisor7()
     {
        this.setPreferredSize(new Dimension(640, 480));
     }

    public void toggle()
     {
       colorFlag = !colorFlag;
     }

    @Override
    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        if (colorFlag) {
            g2.setColor(Color.red);
        } else {
            g2.setColor(Color.blue);
        }
        g2.drawLine(0, 0, getWidth(), getHeight());
        System.out.println("GGG");
    }

    private void starter()
    {
      JFrame f = new JFrame("n-gram visor");
```

```
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(this, BorderLayout.CENTER);
        f.add(new buttonPanel(this), BorderLayout.SOUTH);
        f.pack();
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                new nGramVisor7().starter();
            }
        });
    }
}


class buttonPanel extends JPanel {


private static final long serialVersionUID = 1L;

public buttonPanel(final nGramVisor7 v)
{
        this.add
        (new JButton
        (new AbstractAction("Click")
         {

private static final long serialVersionUID = 1L;

@Override
            public void actionPerformed(ActionEvent e)
{
                v.toggle();
```

```
                v.repaint();
            }
          }
        ));

        this.add
        (new JButton
        (new AbstractAction("Clpppck")
         {

private static final long serialVersionUID = 1L;

@Override
            public void actionPerformed(ActionEvent e)
{
                v.toggle();
                v.repaint();
            }
          }
        ));
    }

}//End of main class I96 nGramVisor7
```

97, page 186. Given our present state of development in the eJ community, any explanation of horizontal transfer will do it. By contrast, a lot of work expects everyone that wants to pass from fuzzy ideas to concrete results. At the mean time, the next code shows us that merging of the idea of Internet with our code may produce a viable program although it represents in its actual state a rather imperfect solution.

```
//Program I97 nGramVisor8

//This program merges the idea from Internet
//to our code to see whether or not we have a solution.

//Intention:
```

```
//This program reads a  proteome from a file,
//ignores commentaries,
//displays its content in various slides,
//carries an n-gram analysis and
//makes a graphic report
//according to a coloring code.
//n-grams with zero  frequency,
//might also  be reported
//together with a frequency table of frequency ranges.
//Feedback information is  displayed.
//Scrolling is possible but because of memory demands,
//it works very poorly.
//The color of the clicked pixel can be read
//at the bottom bar.

//To overcome troubles with too heavy graphics,
//we cut the  great graphic into pieces that could be
//shown without scrolling while offering the
//possibility to navigate along slides.

//So, we add two buttons, one for the next page and
//the other for the previous page.


//Fasta .faa format is required.



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;

import javax.swing.AbstractAction;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;



public class nGramVisor8 extends JPanel
 {
  private static final long serialVersionUID = 1L;

//========Fundamental parameters========



//Modify them as it pleases you:

//letters per n-gram
private static  int n = 3;
//Width of pixel
private static int widthOfPixel = 10;
//Number of pixels per row
private static int pixelsPerRow = 40;
private static int pixelsPerPage;

private static int rowsPerPage;
private static int pageCounter;
```

```
private static int counter;
private static int nPoints;


//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 1000;

//Number of pixels  per page in the vertical direction
private static int usefulPageHeight = 800;

//Directory where your files are kept into
private static String nameOfFolder =
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                   "ACDEFGHIKLMNPQRSTVWY";

private static int freqTable[] = new int[100];
private static int max = 0;
private static String titlesFreqTable[] = new String[100];
```

```
private static boolean print1 = false;
private static boolean print2 = false;

private JLabel barMessage;

//Initialization of  color variables

private static Color OurColors[ ] = {Color.BLACK,
Color.WHITE, Color.GRAY, Color.BLUE, Color.CYAN,
Color.YELLOW, Color.ORANGE, Color.PINK, Color.RED};
private static String NamesColors[ ] = {"BLACK",
"WHITE", "GRAY", "BLUE", "CYAN",
"YELLOW", "ORANGE", "PINK", "RED"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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

private static boolean counting = true;
private static  int numberOfpages =0;
private static boolean go = true;

private static JButton nextPage =
                  new JButton("Next page");
private static JButton previousPage =
new JButton("Previous page");

//Instantiation of main class
private  static nGramVisor8 x =  new nGramVisor8();



//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
```

```
public nGramVisor8()
 {
this.setPreferredSize(new Dimension(1200, 900));
this.setBackground(Color.white);
//A cursor is chosen
setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
 }


//Size of the drawing area
public Dimension getPreferredSize()
{
 return new Dimension(widthOfFrame, heightOfFrame);
}

private void starter()
{
  JFrame f = new JFrame("n-gram visor");
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  f.add(this, BorderLayout.BEFORE_FIRST_LINE);

  f.add(new buttonPanel2(this), BorderLayout.SOUTH);
  barMessage=new JLabel( "Right-click over a given pixel" );
  f.pack();
  f.setLocationRelativeTo(null);
  f.setVisible(true);
  MouseWork l = new MouseWork();
 f.addMouseListener(l);
  System.out.println("n-gram analysis of " + nameOfFile);
  System.out.println("Length of alphabet = "
          + Alphabet.length());
  System.out.println("Number of letters per n-gram = " + n);
  System.out.println("Number of n-grams = " + N);
  rangeTable();
  rowsPerPage = usefulPageHeight/widthOfPixel;
  pixelsPerPage = rowsPerPage * pixelsPerRow;
  nPoints = freq1.F.length;
```

```
  numberOfpages = nPoints/(pixelsPerPage);
  counter = 0;
  pageCounter = 0;
}


public static void main(String[] args)
{
  //Thread inauguration: assign second priority to this
  EventQueue.invokeLater
(new Runnable()
     {

        @Override
        public void run()
        {
            new nGramVisor8().starter();
        }
     }
);
}//End of main

//Draws a page of the great graphic
public static void drawPage(Graphics g2d, int pageCounter)
{
counter = pageCounter*pixelsPerPage;
System.out.println("counter at drawpage= " + counter);
 for (int j = 0; (j < rowsPerPage) & go; j++)
 {
  for (int i = 0; (i < pixelsPerRow) & go; i++)
   {
    int index = pageCounter *pixelsPerPage +
           j*pixelsPerRow + i;
    int range = classification(index);
    if (print1)
    System.out.println("index = " + index  + " range = "
    + range  + "Frequency " + freq1.F[index]);
```

```
    g2d.setColor(OurColors[range]);
    //Pixel is drawn
    g2d.fillOval( i*widthOfPixel, j*widthOfPixel,
     widthOfPixel, widthOfPixel );
    if (counter >= nPoints-1) go = false;
    counter = counter+1;
   }
  }
}


//Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.
@Override
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
  //Embellishment
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

  drawPage(g2d,  pageCounter);

 }


 //Specifications for the frequency table
 public static void definitions()
  {
  titlesFreqTable[0] = "freq = 0";
  titlesFreqTable[1] = "freq = 1";
  titlesFreqTable[2] = " 1 < freq < " +  max/7;
  titlesFreqTable[3] = max/7 + " <= freq < " +  2*max/7;
  titlesFreqTable[4] = 2*max/7 + " <= freq < " +  3*max/7;
```

```
titlesFreqTable[5] = 3*max/7 + "  <= freq < " + 4* max/7;
titlesFreqTable[6] = 4*max/7 + "  <= freq < " +  5*max/7;
titlesFreqTable[7] = 5 *max/7 + " <= freq < " +  6*max/7;
titlesFreqTable[8] = 6*max/7 + "  <= freq <= " +  max;
}


//Assigns a frequency range to the n-gram
//given by index.
//Builds the frequency table by ranges.
//counting is a boolean flag for counting
public static int classification(int index)
{
int rangeNumber = -1;
//Colors are assigned according to frequency
if ( freq1.F[index] ==  0)
 {

  if (counting) freqTable[0]++;
       /*System.out.println(index + "   "
                   + indexInverse(n, index));
                   */
  rangeNumber = 0;

 }

if ( freq1.F[index] ==  1)
 {
   if (counting) freqTable[1]++;
       /*System.out.println(index + "   "
                  + indexInverse(n, index));
                  */
   rangeNumber = 1;
 }


if (( freq1.F[index] >  1)  &
               ( freq1.F[index] <  max/7))
```

```
    {
     if (counting) freqTable[2]++;
     rangeNumber = 2;
     }
  if (( freq1.F[index] >=  max/7)  &
                 ( freq1.F[index] <  2* max/7))
     {
     if (counting) freqTable[3]++;
      rangeNumber = 3;
     }
  if (( freq1.F[index] >= 2* max/7)  &
                 ( freq1.F[index]  < 3 * max/7))
    {
     if (counting) freqTable[4]++;
     rangeNumber = 4;
    }
  if (( freq1.F[index] >=  3*max/7)  &
                ( freq1.F[index] <  4* max/7))
    {
     if (counting) freqTable[5]++;
     rangeNumber = 5;
    }
  if (( freq1.F[index] >=  4*max/7)  &
                ( freq1.F[index] <  5* max/7))
    {
     if (counting) freqTable[6]++;
     rangeNumber = 6;
    }
  if (( freq1.F[index] >= 5* max/7)  &
     ( freq1.F[index] <  6* max/7))
     {
     if (counting) freqTable[7]++;
     rangeNumber = 7;
     }
  if (( freq1.F[index] >=  6*max/7)  &
              ( freq1.F[index] <=    max))
     {
     if (counting) freqTable[8]++;
```

```
    rangeNumber = 8;
   }

//Counting must be done just once
counting = false;
return rangeNumber;
}



public static void report()
{
System.out.println("Frequency table of " + " " + n
                                        +  "-grams");
for(int j = 0; j < 9; j++)
    System.out.println(j +  " " + freqTable[j]
     + " for range  " +  titlesFreqTable[j]
      + "  " + NamesColors[j]);
//Test
int sumNGrams = 0;
for(int j = 0; j < 9; j++)
     sumNGrams = sumNGrams + freqTable[j];
System.out.println("Sum =  " + sumNGrams + " " + n
                                       +  "-grams");

}



//Frequencies are grouped in ranges
public static void rangeTable()
{
  try
    {
     //n-gram frequencies are found and kept in freq1
 findFrequencies();
    }
   catch (IOException e)
    {
```

```
   e.printStackTrace();
      }
    max = findMax();
    definitions();
    System.out.println("Frequency scale goes from 0 to "
        + max + "\nCode for intensity from less to" +
     " more frequency: \n black (absence), " +
     "white (freq = 1), gray, blue, cyan, yelow," +
   " orange, pink, red");

    nPoints = freq1.F.length;

    for (int index = 0; (index < nPoints) & go; index++)
     {
      int range = classification(index);
      if (print1)
      System.out.println("index = " + index  + " range = "
        + range  + "Frequency " + freq1.F[index]);

     }
    report();
  }




    //======== Operative part ============


     //Machine for reading from hard disc
     private static  Scanner input;

     //Path to file
     private static  String path = nameOfFolder + nameOfFile;

     private static StringBuffer proteome =
```

```
                          new StringBuffer("");


      //======= Inner class intVector ========
   //This inner class defines a vector with
   //integer numbers as a new type
   private  class intVector10
   {
   int L = (int) Math.pow(lengthAlf, n);
   int F[] = new int[L];

   //An instance of intVector can be
   //initialized in various ways:

   //Automatic zeroed initialization
   intVector10( )
     {
      for(int i = 0; i < L; i++)
      F[i] = 0;
     }


   }//end of class intVector


   //========End of inner class intVector======



    private static intVector10  freq1
    = x.new intVector10();




public static void findFrequencies() throws IOException
{
```

```
readData(path);
proteome = readData(path);
System.out.println("length of proteome = "
+ proteome.length());
freq1 = nGramsFrequency(n, proteome);
if (print2)
{
System.out.println(proteome);
System.out.println("Index, n-gram, and frequency");
for(int i = 0; i < freq1.F.length; i++)
System.out.println(i + "\t" +  indexInverse(n, i)
    + "\t" + freq1.F[i]);
}
findMax();
}


// We read  record from file
public  static  StringBuffer readData(String path)
    throws IOException
{
StringBuffer text = new StringBuffer("");
//try and catch gates
try
{
FileReader fr = new FileReader(path);
BufferedReader br = new BufferedReader(fr);
String s = "";

while((s = br.readLine()) != null)
{
//Filtering of first line of each subsequence
if (s.charAt(0) != '>' )
{
//System.out.println(s);
text = text.append(s);
}
}
```

```
fr.close();
}
catch ( NoSuchElementException elementException )
{
System.err.println( "File is corrupted." );
input.close();
System.exit( 1 );
}
catch ( IllegalStateException stateException )
{
System.err.println( "Reading aborted." );
System.exit( 1 );
}
return text;
}


//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;
System.out.println("Alien : " + p);
}
}
if (print1)
System.out.println( "ngram = " + ngram + " index = "
                          + ind);
return ind;
}

//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 ;
```

```
}
if (print1)
System.out.println( "ngram = " + ngram +
 " index = "   + ind);
return ngram;
}


/*
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 intVector10 nGramsFrequency(int n,
       StringBuffer Text)
{
//Zeroed initialization
intVector10 F = x.new intVector10();
int nText = Text.length();
System.out.println("Length  of text = " + nText);
int start = 0;
int ind = 0;
for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
```

```
start = start +1;
}
return F;
}


private static int findMax()
{
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
}


//==========Mouse methods==================

public class MouseWork  implements MouseListener
{
 public void mouseClicked( MouseEvent event )
  {
   int i = (int) Math.floor(event.getX()/widthOfPixel);
   int j = (int) Math.floor(event.getY()/widthOfPixel);
   int index = j*pixelsPerRow + i;
   int freq = freq1.F[index];
   //counting = false;
   int range = classification(index);
   String nameOfColor = NamesColors[range];

   String nGram = indexInverse(n,index);
   barMessage.setText( String.format(
           "Clicked  at [%d, %d]," +
            "Page number " + "page" +
       " index = %d, freq = %d, n-gram = %s, %s",
       i, j, index, freq, nGram, nameOfColor) );
}

public void mousePressed( MouseEvent event ){  }
```

```
public void mouseReleased( MouseEvent event ){  }
public void mouseEntered( MouseEvent event ){  }
public void mouseMoved( MouseEvent event ){  }
public void mouseDragged( MouseEvent event ){  }
public void mouseExited( MouseEvent event ){  }

 }//End of inner class MouseWork



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



//Buttons are implemented
class buttonPanel2 extends JPanel
{


private static final long serialVersionUID = 1L;

public buttonPanel2(final nGramVisor8 v)
{
        //Button = previous page
this.add
        (new JButton
        (new AbstractAction("Previous page")
         {

private static final long serialVersionUID = 1L;

@Override
            public void actionPerformed(ActionEvent e)
{
                nGramVisor8.pageCounter--;
                v.repaint();
            }
          }
```

```
        ));

        //Button = next page
        this.add
        (new JButton
        (new AbstractAction("next page")
         {

private static final long serialVersionUID = 1L;

@Override
            public void actionPerformed(ActionEvent e)
{
nGramVisor8.pageCounter++;
                v.repaint();
            }
          }
        ));

        //Button = message by mouselistener
        this.add
        (new JButton
        (new AbstractAction("Mouse listener action goes here")
         {

private static final long serialVersionUID = 1L;

@Override
            public void actionPerformed(ActionEvent e)
{
                //v.toggle();
                v.repaint();
            }
          }
        ));
    }

}//End of inner class buttonPanel2
```

```
}//End of class I 97 nGramVisor8
```

102, page 224. A method for the synthesis of proteomes with letters concatenated at random was added and its output was given for graphical displaying. The result was disappointing: the table of frequencies got messy but the most terrible thing was that that table strongly changed if the length of the synthesized text was changed. The reason is that our charts depend on a classification by absolute frequency ranges. But these depend directly on the length of analyzed text. So, with one length, one gets a very different chart than beginning with a text with different length. To solve this trouble, we decided to choose a length equal to 5 times the number of possible n-grams. In that way, all n-grams must have 5 representatives apart from sampling effects. In all cases the result was an n-gram chart that is highly uniform all over its extension. The code follows:

```
//Program I102 nGramVisora11

//This program reads a  proteome from a file,
//or synthesizes one by random concatenation
//of the letters of a given alphabet,
//ignores commentaries,
//makes a graphic report
//according to a coloring code.
//Codes are different for true proteomes
//than for false ones.

//Code is reported to the console
//together with a frequency table of frequency ranges.

//Feedback information is  displayed.
//The color of the clicked pixel can be read
//at the SOUTH.

//To overcome troubles with too heavy graphics,
//we cut the  great graphic into pieces that could be
//shown without scrolling while offering the
```

```
//possibility to navigate along slides.
//So, we add two buttons, one for the next page and
//the other for the previous page.

/*

 To decide that a proteome must be synthesized
 as random concatentaion of letters, choose
 in line 82 the next instruction:

isProteomeTrue = false;

To choose a true proteome from a file, set it to true:

isProteomeTrue = true;

*/
//Fasta .faa format is required.


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Random;
import java.util.Scanner;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;



public class nGramVisora11 extends JPanel
 {
  private static final long serialVersionUID = 1L;

//========Fundamental parameters========
//Modify them as it pleases you:


//A true proteome is read from a file,
//a false one is synthesized by randomness.
//The length of synthesized proteome is supposed to be
//five times the number of possible n-grams.
private static boolean isProteomeTrue = false;

//letters per n-gram
private static  int n = 4;
//Width of pixel
private static int widthOfPixel = 1;
//Number of pixels per row
private static int pixelsPerRow = 500;
private static int pixelsPerPage;

private static int rowsPerPage;
private static int page;

private static int shownPixels;
private static int nPoints;
```

```
//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 800;

//Number of pixels  per page in the vertical direction
private static int usefulPageHeight = 500;

//Directory where your files are kept into
private static String nameOfFolder =
           "/home/jose/jose/AAjoser/Ciencia/" +
           "GenomesComplete/GenomesSample/";
//Name of the species whose proteome is to be analyzed
private static String nameOfFile =
"Arcobacter_L_uid158135.faa";

//The alphabet conforms to  fasta format
//for amino acids (.faa extension).
//Full .faa alphabet:
/*
 //Full .faa alphabet:
private static String Alphabet =
                   "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                   "ACDEFGHIKLMNPQRSTVWY";

private static int freqTable[] = new int[100];
private static int max = 0;
private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;
```

```
//Initialization of  color variables

private static Color OurColors[ ] = {Color.BLACK,
Color.WHITE, Color.GRAY, Color.BLUE, Color.CYAN,
Color.YELLOW, Color.ORANGE, Color.PINK, Color.RED};
private static String NamesColors[ ] = {"BLACK",
"WHITE", "GRAY", "BLUE", "CYAN",
"YELLOW", "ORANGE", "PINK", "RED"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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

private static boolean counting = true;
private static  int numberOfPages =0;
private static boolean go = true;

private static String message = "";

private JLabel label = new JLabel("Click over a pixel");

//Generator of random numbers
private static Random r  = new Random();


//Instantiation of main class
private  static nGramVisora11 x =  new nGramVisora11();

JFrame f = new JFrame("n-gram visor");


//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public nGramVisora11()
```

```
 {
this.setPreferredSize(
new Dimension(widthOfFrame, heightOfFrame));
this.setBackground(Color.white);
//A cursor is chosen
setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
 }



private void starter()
{
 System.out.println("n-gram analysis of " + nameOfFile);
 System.out.println("Length of alphabet = "
          + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of possible n-grams = " + N);
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.add(this, BorderLayout.BEFORE_FIRST_LINE);
 f.add(new buttonPanel5(this), BorderLayout.SOUTH);
 f.add(label);
 f.pack();
 f.setLocationRelativeTo(null);
 f.setVisible(true);
 MouseWork2 l = new MouseWork2();
 f.addMouseListener(l);
 rangeTable();
}


public static void main(String[] args)
{

  //Thread inauguration: assigns second priority to this
  EventQueue.invokeLater
(new Runnable()
     {
```

```
        @Override
        public void run()
        {
            new nGramVisora11().starter();
        }
    }
);
}//End of main

//buttonsHousekeeping
private static void buttonsHousekeeping()
{
 if (page <= 0)
 {
  nGramVisora11.buttonPanel5.previousPage.setEnabled(false);
  nGramVisora11.buttonPanel5.nextPage.setEnabled(true);
 }
 if (page >= numberOfPages)
  {
   nGramVisora11.buttonPanel5.previousPage.setEnabled(true);
   nGramVisora11.buttonPanel5.nextPage.setEnabled(false);
  }
 if ((page >= 0) & (page <=numberOfPages)) go = true;
else go = false;
 if (numberOfPages == 0)
  {
   nGramVisora11.buttonPanel5.previousPage.setEnabled(false);
   nGramVisora11.buttonPanel5.nextPage.setEnabled(false);
  }

}

//Draws a page of the great graphic
private static void drawPage(Graphics g2d, int page)
{
 shownPixels = page*pixelsPerPage;
 buttonsHousekeeping();
```

```
 //Page is displayed
 for (int j = 0; (j < rowsPerPage) & go; j++)
  {
   for (int i = 0; (i < pixelsPerRow) & go; i++)
{
 int index = page *pixelsPerPage +
           j*pixelsPerRow + i;
 int range = classification(index, isProteomeTrue);
 if (print1)
 System.out.println("index = " + index  + " range = "
    + range  + "Frequency " + freq1.F[index]);

 g2d.setColor(OurColors[range]);
    //Pixel is drawn
 g2d.fillOval( i*widthOfPixel, j*widthOfPixel,
     widthOfPixel, widthOfPixel );
 if (shownPixels >= nPoints-1) go = false;
 shownPixels = shownPixels+1;
}
  }
}


//Drawing tool
//Second method to be executed and
//then on always ready to get into action.
//The vector of frequencies is displayed in
//graphic form according to a coloring code.
@Override //Repainting is allowed
protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
  //Embellishment =  clogging code
  if (n==3)
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
```

```
  drawPage(g2d,  page);

 }


//Specifications for the frequency table
private static void definitions()
 {
  if (isProteomeTrue)
  {
  titlesFreqTable[0] = "freq = 0";
  titlesFreqTable[1] = "freq = 1";
  titlesFreqTable[2] = " 1 < freq < " +  max/7;
  titlesFreqTable[3] = max/7 + " <= freq < " +  2*max/7;
  titlesFreqTable[4] = 2*max/7 + " <= freq < " +  3*max/7;
  titlesFreqTable[5] = 3*max/7 + "  <= freq < " + 4* max/7;
  titlesFreqTable[6] = 4*max/7 + "  <= freq < " +  5*max/7;
  titlesFreqTable[7] = 5 *max/7 + " <= freq < " +  6*max/7;
  titlesFreqTable[8] = 6*max/7 + "  <= freq <= " +  max;
  }
  else
  {
for(int i = 0; i < 8; i++)
titlesFreqTable[i] = " freq = " + i;
titlesFreqTable[8] = 8 + "   freq >= 8 " ;

  }

 }


  //Assigns a frequency range to the n-gram
  //given by index.
  //Builds the frequency table by ranges.
  //counting is a boolean flag for counting
  private static int classification(int index)
  {
  int rangeNumber = -1;
```

```
//Colors are assigned according to frequency

if ( freq1.F[index] ==  0)
 {

  if (counting) freqTable[0]++;
       /*System.out.println(index + "   "
                  + indexInverse(n, index));
                  */
  rangeNumber = 0;

 }

if ( freq1.F[index] ==  1)
 {
   if (counting) freqTable[1]++;
       /*System.out.println(index + "   "
                  + indexInverse(n, index));
             */
   rangeNumber = 1;
 }


if (( freq1.F[index] >  1)  &
               ( freq1.F[index] <  max/7))
  {
   if (counting) freqTable[2]++;
   rangeNumber = 2;
   }

if (( freq1.F[index] >=  max/7)  &
               ( freq1.F[index] <  2* max/7))
   {
   if (counting) freqTable[3]++;
    rangeNumber = 3;
   }
if (( freq1.F[index] >= 2* max/7)  &
               ( freq1.F[index]  < 3 * max/7))
```

```
  {
   if (counting) freqTable[4]++;
   rangeNumber = 4;
  }
if (( freq1.F[index] >=  3*max/7)  &
              ( freq1.F[index] <  4* max/7))
  {
   if (counting) freqTable[5]++;
   rangeNumber = 5;
  }
if (( freq1.F[index] >=  4*max/7)  &
              ( freq1.F[index] <  5* max/7))
  {
   if (counting) freqTable[6]++;
   rangeNumber = 6;
  }
if (( freq1.F[index] >= 5* max/7)  &
   ( freq1.F[index] <  6* max/7))
   {
   if (counting) freqTable[7]++;
   rangeNumber = 7;
   }
if (( freq1.F[index] >=  6*max/7)  &
           ( freq1.F[index] <=    max))
   {
   if (counting) freqTable[8]++;
    rangeNumber = 8;
   }


return rangeNumber;
}


private static int classification2(int index)
{
 int rangeNumber = -1;
 //Colors are assigned according to frequency
```

```
   for(int i = 0; i < 8; i++)
   {
   if ( freq1.F[index] ==  i)
   {
    if (counting) freqTable[i]++;
    rangeNumber = i;
   }
   }
   if (freq1.F[index] >= 8)
     {
  if (counting) freqTable[8]++;
  rangeNumber = 8;
     }

   return rangeNumber;
  }

//Assigns a frequency range to the n-gram
//given by index.
//Builds the frequency table by ranges.
//counting is a boolean flag for counting
private static int classification(int index, boolean isProteomeT
 {
 int output = 0;
if (isProteomeTrue) output =  classification(index);
else output = classification2(index);
return output;
 }


//Frequency table is reported
 private static void report(int max)
  {

   System.out.println("Frequency table of " + " " + n
                                          +  "-grams");
   for(int j = 0; j < 9; j++)
      System.out.println(j +  " " + freqTable[j]
```

```
     + " for range  " +  titlesFreqTable[j]
      + "  " + NamesColors[j]);
//Test
int sumNGrams = 0;
for(int j = 0; j < 9; j++)
     sumNGrams = sumNGrams + freqTable[j];
System.out.println("Sum =  " + sumNGrams + " " + n
                                      +  "-grams");

}


//Frequencies are grouped in ranges
private static void rangeTable()
{
  try
    {
     //n-gram frequencies are found and kept in freq1
 findFrequencies();
    }
   catch (IOException e)
    {
 e.printStackTrace();
    }
  max = findMax(freq1);
  definitions();


  System.out.println("Frequency scale goes from 0 to "
      + max + "\nCode for intensity from less to" +
   " more frequency: \n black (absence), " +
   "white (freq = 1), gray, blue, cyan, yelow," +
 " orange, pink, red");

  nPoints = freq1.F.length;

  for (int index = 0; (index < nPoints) & go; index++)
   {
```

```
      int range = classification(index, isProteomeTrue);
      if (print1)
      System.out.println("index = " + index  + " range = "
        + range  + "Frequency " + freq1.F[index]);

     }
     //Counting of frequencies must be done just once
     counting = false;
  report(max);
 }


   //======== Operative part ============


     //Machine for reading from hard disc
     private static  Scanner input;

     //Path to file
     private static  String path = nameOfFolder + nameOfFile;

     private static StringBuffer proteome =
                            new StringBuffer("");


     //======= Inner class intVector ========
  //This inner class defines a vector with
  //integer numbers as a new type
  private  class intVector12
  {
  int L = (int) Math.pow(lengthAlf, n);
  int F[] = new int[L];

  //An instance of intVector can be
  //initialized in various ways:

  //Automatic zeroed initialization
  intVector12( )
```

```
     {
      for(int i = 0; i < L; i++)
      F[i] = 0;
     }


   }//end of class intVector


   //========End of inner class intVector======



 private static intVector12  freq1
    = x.new intVector12();

 //Frequencies of n-grams are calculated
 private static StringBuffer randomProteome()
  {
   StringBuffer randText =  new StringBuffer("");
   char c = ' ';
   //N = number of possible n-grams
   for(int i = 0; i < 5*N; i++)
{
 int k = r.nextInt(lengthAlf);
 c = Alphabet.charAt(k);
 //char is appended to text
 randText = randText.append(c);
 }
  return randText;
 }



//Frequencies of n-grams are calculated
private static void findFrequencies() throws IOException
{
 if (isProteomeTrue)
```

```
 proteome = readData(path);
  else proteome = randomProteome();
 rowsPerPage = usefulPageHeight/widthOfPixel;
 pixelsPerPage = rowsPerPage * pixelsPerRow;
 //Bug has been corrected!
 freq1 = nGramsFrequency(n, proteome);
 nPoints = freq1.F.length;

 numberOfPages =  nPoints/pixelsPerPage;


 if (numberOfPages * pixelsPerPage == nPoints)
        numberOfPages--;
 System.out.println("Number of pages = " + numberOfPages);
 System.out.println("length of proteome = "
  + proteome.length());

 if (print2)
  {
   System.out.println(proteome);
   System.out.println("Index, n-gram, and frequency");
 for(int i = 0; i < freq1.F.length; i++)
   System.out.println(i + "\t" +  indexInverse(n, i)
       + "\t" + freq1.F[i]);
}


}


// We read  record from file
private  static  StringBuffer readData(String path)
    throws IOException
{
StringBuffer text = new StringBuffer("");
//try and catch gates
try
{
```

```
FileReader fr = new FileReader(path);
BufferedReader br = new BufferedReader(fr);
String s = "";

while((s = br.readLine()) != null)
{
//Filtering of first line of each subsequence
if (s.charAt(0) != '>' )
{
//System.out.println(s);
text = text.append(s);
}
}

fr.close();
}
catch ( NoSuchElementException elementException )
{
System.err.println( "File is corrupted." );
input.close();
System.exit( 1 );
}
catch ( IllegalStateException stateException )
{
System.err.println( "Reading aborted." );
System.exit( 1 );
}
return text;
}


//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;
System.out.println("Alien : " + p);
}
}
if (print1)
System.out.println( "ngram = " + ngram + " index = "
                          + ind);
return ind;
}

//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 ;

}
if (print1)
System.out.println( "ngram = " + ngram +
 " index = "   + ind);
return ngram;
}


/*
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 intVector12 nGramsFrequency(int n,
       StringBuffer Text)
{
//Zeroed initialization
intVector12 F = x.new intVector12();
int nText = Text.length();
```

```
System.out.println("Length  of working text = " + nText);
int start = 0;
int ind = 0;
for(int c=0; c < nText-n+1; 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.F[ind] = F.F[ind] + 1;
start = start +1;
}
findMax(freq1);
return F;
}


//Maximal frequency is found
private static int findMax(intVector12 freq1)
{
int max = 0;
for(int i = 0; i< freq1.F.length; i++)
if (freq1.F[i] > max ) max = freq1.F[i];
return max;
}


//==========Mouse methods==================

private class MouseWork2  implements MouseListener
{
 public void mouseClicked( MouseEvent event )
  {
   //System.out.println(event.getX()+ " " + event.getY());
   int i = (int) Math.floor(event.getX()/widthOfPixel);
```

```
   //28 = Interference  from label
   int j = (int) Math.floor((event.getY()-28)/widthOfPixel);
   int index = page *pixelsPerPage +
      j*pixelsPerRow + i;
   int freq = freq1.F[index];
   //counting = false;
   int range = classification(index, isProteomeTrue);
   String nameOfColor = NamesColors[range];
   String nGram = indexInverse(n,index);
    message = String.format(
          "Clicked  at [%d, %d]," +
              " Page = %d," +
      " index = %d, freq = %d, n-gram = %s, %s",
      i, j, page, index, freq, nGram, nameOfColor);
   label.setText(message);
}

public void mousePressed( MouseEvent event ){  }
public void mouseReleased( MouseEvent event ){  }
public void mouseEntered( MouseEvent event ){  }
public void mouseExited( MouseEvent event ){  }

 }//End of inner class MouseWork


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


//Buttons are implemented
static class buttonPanel5 extends JPanel
{


private static final long serialVersionUID = 1L;
final static JButton previousPage =
                              new JButton("Previous page");
final static JButton nextPage = new JButton("Next page");
//Constructor
```

```
private buttonPanel5(final nGramVisora11 v)
{
        //Button = previous page

this.add(previousPage);
        previousPage.addActionListener(new ActionListener()
        {
          public void actionPerformed(ActionEvent e)
          {
           nGramVisora11.page--;
              v.repaint();
              if (page <= 0)
                {
               previousPage.setEnabled(false);
               nextPage.setEnabled(true);
               go = false;
                }
              else
                {
               previousPage.setEnabled(true);
               go = true;
                }
          }
        });

         //Button = next page

         this.add(nextPage);
       nextPage.addActionListener(new ActionListener()
       {
         public void actionPerformed(ActionEvent e)
         {
          nGramVisora11.page++;
             v.repaint();
             if (page >= numberOfPages)
             {
            nextPage.setEnabled(false);
           previousPage.setEnabled(true);
```

```
            go = false;
             }
           else
             {
            nextPage.setEnabled(true);
            previousPage.setEnabled(true);
            go = true;
             }
      }
    });



    }//End of constructor of buttonPanel3

}//End of inner class buttonPanel3

 }//End of class I102 nGramVisora11
```

107, page 242.

```
/*
Program I107 twoFreqtables2

This program tests the null hypothesis that
n-gram frequencies fits
a uniform distribution.
If this hypothesis is accepted,
randomness is the explanation of our beings.
Otherwise, we are invited to formulate more
complex, more attractive hypotheses so,
we are invited to make science.

Specifically:

This program calculates the   frequency tables
of n-grams of two true proteomes and
```

```
other two that are synthetic
and displays the corresponding graphics.

True proteomes are read from a file,
Fasta .faa format is required.
False proteomes are  synthesized by random concatenation
of the letters of a given alphabet,
the length of the first one is equal to
the length of the fist real proteome
and likewise with the second.


Relative frequencies result from dividing
absolute frequencies by the number of n-grams
that are contained in the studied proteome.
In that way, relative frequencies add up to one
and are independent of the length of the text.

Relative frequencies are classified by ranges
and so we get the frequency tables
whose bar charts are displayed.

A Smirnov Kolmogorov test is carried out
to test the hypothesis that
n-gram frequencies fits
a uniform distribution.

Two tries are made.

*/


import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
```

```
import java.awt.Graphics2D;
import java.awt.RenderingHints;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Random;
import java.util.Scanner;


import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;




public class twoFreqTables2 extends JPanel
 {
  private static final long serialVersionUID = 1L;

//========Fundamental parameters========
//=====Modify them as it pleases you====

//letters per n-gram
private static  int n = 2;

//Width of frame
private static int widthOfFrame = 1200;
//Height of frame
private static int heightOfFrame = 800;

//Width of bars in bar chart
private static int widthOfBar = 50;
```

```
//Directory where your files are kept into
private static String nameOfFolder =
          "/home/jose/jose/AAjoser/Ciencia/" +
          "GenomesComplete/GenomesSample/";
//Names of two species whose proteomes are to be compared
private static String nameOfFile1 =
            "Arcobacter_L_uid158135.faa";
private static String nameOfFile2 =
"Zymomonas_mobilis_ATCC_10988_uid55403.faa";
private static String nameOfFile3 =
"Synthetic-1";
private static String nameOfFile4 =
"Synthetic-2";


/*The alphabet conforms to  fasta format
for amino acids (.faa extension).

 //Full .faa alphabet:
private static String Alphabet =
                  "ABCDEFGHIKLMNPQRSTUVWYZX*-";
 */

//Short .faa alphabet
private static String Alphabet =
                  "ACDEFGHIKLMNPQRSTVWY";



private static String titlesFreqTable[] = new String[100];

private static boolean print1 = false;
private static boolean print2 = false;



//Initialization of  color variables
```

```
private static Color OurColors[ ] =
   {Color.BLACK, Color.BLUE, Color.CYAN, Color.GRAY,
Color.GREEN, Color.MAGENTA, Color.ORANGE,
Color.PINK, Color.RED,    Color.YELLOW,
Color.black, Color.blue, Color.cyan, Color.gray,
Color.green, Color.magenta, Color.orange,
Color.PINK, Color.RED, Color.WHITE,  Color.YELLOW};
private static String NamesColors[ ] =
   {"BLACK","BLUE", "CYAN","GRAY",
"GREEN", "MAGENTA", "ORANGE",
"PINK", "RED",    "YELLOW",
"black", "blue", "cyan", "gray",
"green", "magenta", "orange",
"pink", "red", "white", "yellow"};

//length of the Alphabet
private static  int lengthAlf = Alphabet.length();

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


private static JLabel label = new JLabel("TTTT");


//Machine for reading from hard disc
private static  Scanner input;

//Path to file
private static  String path1 = nameOfFolder + nameOfFile1;
private static  String path2 = nameOfFolder + nameOfFile2;
//Proteomes: two real, two synthetic
private static StringBuffer proteome1 =
                        new StringBuffer("");
private static StringBuffer proteome2 =
            new StringBuffer("");
private static StringBuffer proteome3 =
            new StringBuffer("");
```

```
private static StringBuffer proteome4 =
         new StringBuffer("");

//Vectors to keep relative frequencies
private static double[]  freq1
                      = new double[N] ;
private static double[]   freq2
                     = new double[N];
private static double[]  freq3
                     = new double[N];
private static double[]   freq4
                    = new double[N];
//Maximal values of freq-ith vectors
private static double max1, max2, max3, max4;
//Global maximal value = the greatest max
private static double max;
private static int nMarks;
private static double delta;

//Frequencies are classified by ranges
private static int freqTable1[] = new int[100];
private static int freqTable2[] = new int[100];
private static int freqTable3[] = new int[100];
private static int freqTable4[] = new int[100];

//Generator of random numbers
private static Random r  = new Random();


JFrame f = new JFrame(
"Relative frequency tables of two proteomes");


//======= End of fundamental parameters

//Constructor of main class:
//Primary instructions for initialization
public twoFreqTables2()
```

```
 {
  this.setPreferredSize(
new Dimension(widthOfFrame, heightOfFrame));
  this.setBackground(Color.white);
  //A cursor is chosen
  setCursor (Cursor.
   getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
 }


//This i how the application begins
private void starter() throws IOException
{
 System.out.println("Comparative n-gram analysis of \n"
                   + nameOfFile1 + "\n and \n"
            + nameOfFile2 + "\n");
 System.out.println("Length of alphabet = "
          + Alphabet.length());
 System.out.println("Number of letters per n-gram = " + n);
 System.out.println("Number of possible n-grams = " + N);
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.add(this, BorderLayout.BEFORE_FIRST_LINE);
 f.add(label);
 f.pack();
 f.setLocationRelativeTo(null);
 f.setVisible(true);
 //Proteomes are read from files or are synthesized
 readProteomes();
 //All the work is done
 study();

}


public static void main(String[] args)
{

  //Thread inauguration: assigns second priority to this
```

```
  EventQueue.invokeLater
(new Runnable()
     {
        @Override
        public void run()
        {
            try {
new twoFreqTables2().starter();
} catch (IOException e) {
e.printStackTrace();
}
        }
     }
);
}//End of main



//Draws bar charts
private static void drawCharts(Graphics g2d)
{
 int xo = 10;
 int yo = 120;
 int maxHeight = 100;
 int height;
 //Proteome1
 for (int i = 0; (i < nMarks) ; i++)
  {
   double r = freqTable1[i];
   height = (int) ((r/max )* maxHeight);
   g2d.setColor(OurColors[i]);
   g2d.drawRect(xo + i*widthOfBar, yo-height,
   widthOfBar, height);
  }
 //Proteome2
   for (int i = 0; (i < nMarks) ; i++)
{
```

```
   double r = freqTable2[i];
  height = (int) ((r/max )* maxHeight);
 g2d.setColor(OurColors[i]);
     g2d.drawRect(xo + i*widthOfBar, 2*yo-height,
           widthOfBar, height);
}
 //Proteome3
   for (int i = 0; (i < nMarks) ; i++)
{
   double r = freqTable3[i];
  height = (int) ((r/max )* maxHeight);
 g2d.setColor(OurColors[i]);
     g2d.drawRect(xo + i*widthOfBar, 3*yo-height,
      widthOfBar, height);
}
 //Proteome4
   for (int i = 0; (i < nMarks) ; i++)
{
   double r = freqTable4[i];
  height = (int) ((r/max )* maxHeight);
 g2d.setColor(OurColors[i]);
     g2d.drawRect(xo + i*widthOfBar, 4*yo-height,
      widthOfBar, height);
}
label.setText( nameOfFile1 + ";    " + nameOfFile2 + ";    "
+ nameOfFile3 + ";    " + nameOfFile4 + ". ");
label.setVisible(true);
}


//Drawing tool
//Second method to be executed and
//then on always ready to get into action.

protected void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D) g;
```

```
  //Embellishment
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

  drawCharts(g2d);

 }



//Title specifications for the frequency table
private static void definitions(double max)
{
System.out.println(  "maxh = "+ max);
 //Scale is found
 double k = max;
 double j = 0;
 while (k < 1)
 {
  k =  k * 10;
  j++;
 }

 delta = 1;
 delta = delta / Math.pow(10, j+1);
 System.out.println("j = " + j + " Delta = "+ delta);

 nMarks =   (int) (max/delta)  + 2;
 if (nMarks > 20)
  {
delta = delta * 10;
nMarks = nMarks/10+2;
  }
 for(int i = 0; i < nMarks; i++)
titlesFreqTable[i] = " freq = " + i + " * "  + delta;

}
```

```
//Builds the frequency table by ranges.
//counting is a boolean flag for counting
private static int[] classification(double[] freq,
                              double max)
{
int freqTable[] = new int[100];
boolean go = true;
for(int index = 0; index < freq.length; index++)
{
 for(int i = 0; (i < (nMarks+4) & (go)); i++)
  {
if (( freq[index] >=  i*delta)  &
             ( freq[index] <  (i+1)*delta))
     {
      freqTable[i]++;
      go = false;
     }
  }
 go = true;
}
return freqTable;
}


//Frequency table is reported
private static void reportTable(int[] freqTable)
 {
   System.out.println("Frequency table of " + " " + n
                  +  "-grams  ");
   for(int j = 0; j < nMarks; j++)
      System.out.println(j +  " " + freqTable[j]
       + " for range  " +  titlesFreqTable[j]
        + "  " + NamesColors[j]);
  //Test
  int sumNGrams = 0;
  for(int j = 0; j < nMarks; j++)
```

```
        sumNGrams = sumNGrams + freqTable[j];
  System.out.println("Sum =  " + sumNGrams + " " + n
                                        +  "-grams");
  }


//Frequency tables are  reported
private static void reportTables()
{
 System.out.println("\n  " + nameOfFile1);
 reportTable(freqTable1);
 System.out.println("\n  " + nameOfFile2);
 reportTable(freqTable2);
 System.out.println("\n  " + nameOfFile3);
 reportTable(freqTable3);
 System.out.println("\n  " + nameOfFile4);
 reportTable(freqTable4);
 }


//Frequencies are grouped in ranges
private static void rangeTables() throws IOException
 {
  max1 = findMax(freq1);
  max2 = findMax(freq2);
  max3 = findMax(freq3);
  max4 = findMax(freq4);
  //A global max is found to define a universal scale
  max = max1;
  if (max < max2) max = max2;
  if (max < max3) max = max3;
  if (max < max4) max = max4;
  System.out.println("\nGlobal max of relative freq =" +
                        "  "+ max + "\n");
  definitions(max);
  freqTable1 = classification(freq1, max);
  freqTable2 = classification(freq2, max);
  freqTable3 = classification(freq3, max);
```

```
  freqTable4 = classification(freq4, max);
  max1 = findMax(freqTable1);
  max2 = findMax(freqTable2);
  max3 = findMax(freqTable3);
  max4 = findMax(freqTable4);
//A global max is found to define a universal scale
  max = max1;
  if (max < max2) max = max2;
  if (max < max3) max = max3;
  if (max < max4) max = max4;
  System.out.println("\nGlobal max of absolute freq =" +
                       "  "+ max + "\n");
 }


//All the work is done
private static void study() throws IOException
{
 //n-gram frequencies are found and kept in freq1
 findFrequencies();
 rangeTables();
 reportTables();
 //Smirnov-Kolmogorov test
 System.out.println("\nKolmogorov-Smirnov Test: " +
  "proteome1 vs randomness");
 SKTest(freqTable1,freqTable3);

 System.out.println("\nKolmogorov-Smirnov Test: " +
  "proteome2 vs randomness");

 SKTest(freqTable2,freqTable4);
}


 //======== Operative part ============

 //Frequencies of n-grams are calculated
 private static StringBuffer randomProteome(int length)
```

```
  {
   StringBuffer randText =  new StringBuffer("");
   char c = ' ';
   //N = number of possible n-grams
   for(int i = 0; i < length; i++)
{
 int k = r.nextInt(lengthAlf);
 c = Alphabet.charAt(k);
 //char is appended to text
 randText = randText.append(c);
 }
  return randText;
 }

 //True proteomes are read from files,
 //false ones are synthesized.
 private static void readProteomes() throws IOException
 {
  proteome1 = readData(path1);
  proteome2 = readData(path2);
  int length = proteome1.length();
  proteome3 = randomProteome(length);
  length = proteome2.length();
  proteome4 = randomProteome(length);
 }

//Frequencies of n-grams are calculated
private static void findFrequencies() throws IOException
{
System.out.println("\nProteome 1: " + nameOfFile1);
 //n is the number of letters in n-grams
 freq1 = nGramsFrequency(n, proteome1);
 System.out.println("\nProteome 2: " + nameOfFile2);
 freq2 = nGramsFrequency(n, proteome2);
 System.out.println("\nProteome 3: synthetic");
 freq3 = nGramsFrequency(n, proteome3);
 System.out.println("\nProteome 4: synthetic");
 freq4 = nGramsFrequency(n, proteome4);
```

```
 if (print2)
  {
   System.out.println(proteome1);
   System.out.println("Index, n-gram, and frequency");
   for(int i = 0; i < freq1.length; i++)
   System.out.println(i + "\t" +  indexInverse(n, i)
       + "\t" + freq1[i]);
  }
}


// We read  record from file
private  static  StringBuffer readData(String path)
    throws IOException
{
StringBuffer text = new StringBuffer("");
//try and catch gates
try
{
FileReader fr = new FileReader(path);
BufferedReader br = new BufferedReader(fr);
String s = "";

while((s = br.readLine()) != null)
{
//Filtering of first line of each subsequence
if (s.charAt(0) != '>' )
{
//System.out.println(s);
text = text.append(s);
}
}

fr.close();
}
catch ( NoSuchElementException elementException )
```

```
{
System.err.println( "File is corrupted." );
input.close();
System.exit( 1 );
}
catch ( IllegalStateException stateException )
{
System.err.println( "Reading aborted." );
System.exit( 1 );
}
return text;
}


//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;
System.out.println("Alien : " + p);
}
}
```

```
if (print1)
System.out.println( "ngram = " + ngram + " index = "
                        + ind);
return ind;
}

//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 ;

}
if (print1)
System.out.println( "ngram = " + ngram +
```

```
 " index = "   + ind);
return ngram;
}


/*
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;
  }
 findMax(F);
```

```
 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);
 double max = findMax(F);
 System.out.println("Max of  frequencies "+ max);
 return F;
}


//Maximal frequency is found
private static double findMax(double[] F)
{
 double max = 0;
 for(int i = 0; i< F.length; i++)
 if (F[i] > max ) max = F[i];
 //System.out.println("Max of  frequencies "+ max);
 return max;
}

//Maximal frequency is found
private static double findMax(int[] F)
{
int max = 0;
for(int i = 0; i< F.length; i++)
if (F[i] > max ) max = F[i];
//System.out.println("Max of  frequencies "+ max);
return max;
}


//Smirnov -Kolmogorov pre-test:
//Two vector of absolute frequencies are received,
//The corresponding distribution functions are calculated,
//The maximal value of the absolute difference of
//the two distribution functions is output
```

```
private static double SKTest(int[] Freq1, int[] Freq2)
{
 //System.out.println("Kolmogorov-Smirnov Test");
 double maxD = 0;
 double[]  DistFunction1 = new double[100];
 double[]  DistFunction2 = new double[100];
 double[]  CumFunction1 = new double[100];
 double[]  CumFunction2 = new double[100];
 double[]  AbsDiff = new double[100];

 CumFunction1[0] = Freq1[0];
 //Cumulative absolute frequencies
 for(int i = 1; i < nMarks; i++)
   CumFunction1[i] = CumFunction1[i-1] + Freq1[i];

 CumFunction2[0] = Freq2[0];
 for(int i = 1; i < nMarks; i++)
  CumFunction2[i] = CumFunction2[i-1] + Freq2[i];

 //Distribution functions
 for(int i = 1; i < nMarks; i++)
 {
  DistFunction1[i] =
  CumFunction1[i]/CumFunction1[nMarks-1];
  DistFunction2[i] =
  CumFunction2[i]/CumFunction2[nMarks-1];
  AbsDiff[i] =
  Math.abs(DistFunction1[i] - DistFunction2[i]);
  if ( AbsDiff[i] > maxD) maxD = AbsDiff[i];
 }

  System.out.printf("%7s %14s   %14s  " +
              "%7s %14s   %14s %14s %n",
"Freq1[i]", "CumFunction1[i]", "DistFunction1[i]",
"Freq2[i]", "CumFunction2[i]", "DistFunction2[i]",
"AbsDiff[i]");
```

```
 for ( int i = 0; i < nMarks; i++ )
  System.out.printf("% 7d %14.3f   %14.3f  " +
         "  % 7d   %14.3f     %14.3f  %14.3f %n",
Freq1[i], CumFunction1[i],DistFunction1[i],
Freq2[i], CumFunction2[i],DistFunction2[i],
AbsDiff[i]);
System.out.println( );
System.out.println("Max absolute difference = " +
"Math.abs(DistFunction1[i] - DistFunction2[i])  " +
"\n = " + maxD);
double criticalValue = 1.63/Math.sqrt(CumFunction2[nMarks-1]);
System.out.println("Critical value for significance 0.01 = " +
"\n 1.63/Math.sqrt(CumFunction2[nMarks-1]) = "
+  criticalValue);
if (maxD > criticalValue  )
System.out.println("THE TWO SAMPLES DO NOT COME FROM" +
" THE SAME DISTRIBUTION");
else
System.out.println("THE TWO SAMPLES  COME FROM" +
"THE SAME DISTRIBUTION");

 return maxD;
}

 }//End of main class I107 twoFreqtables2
```

116, page 252.

```
//Program I116 EuclideanDistMatrix2

/*
 The program 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,
 next it calculates the matrix of Euclidean n-gram distances
```

```
 between all proteomes in the chosen folder.

 Sequences must  have extension  .faa (fasta amino acids).

 The entries in the output matrix are suitable amplified
 and rounded to get an easy reading.

 */


import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;



import java.lang.IllegalStateException;


public class EuclideanDistMatrix2
{

 //Machine for reading from hard disc
 private static  Scanner input;

 //Folder with proteomes with .faa extension.
 //Update this to your circumstances
 private static String nameOfFolder =
  "/home/jose/jose/AAjoser/Ciencia/" +
            "GenomesComplete/GenomesSample";

 //Full path to a specific file
 private static  String path;

 private static File[] listOfFiles;
```

```
 //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=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];

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

 //We make run for 7 proteomes
 private static double[][] nGramFreqMatrix = new double[L][7];

 //We make run for 7 proteomes
 private static double[][] nDistanceMatrix = new double[L][7];


 private static Boolean printAll;


 //All files in the folder that is addressed
 //by path are found.
 //Extension .faa is required.
 public static void getNamesOfFiles(String path)
  {
```

```
File folder = new File(path);
listOfFiles = folder.listFiles();

String file;
System.out.println("List of species: \n");
for (int i = 0; i < listOfFiles.length; i++)
 {
  if (listOfFiles[i].isFile())
{
 file = listOfFiles[i].getName();
 //Filter for .faa format. Change this on need.
 if (file.endsWith(".faa") )
       System.out.println(i + " " + file);
 }
  }
}

  // We read  record from file
  public  static  StringBuffer readData(String path)
                  throws IOException
  {
   //A proteome is kept in a stringBUffer (heavy duty string)
   StringBuffer genome = new StringBuffer("");
   //try and catch gates
     try
     {
      FileReader fr = new FileReader(path);
      BufferedReader br = new BufferedReader(fr);
      String s = "";

      while((s = br.readLine()) != null)
      {
      //Filtering of first line of each subsequence
      if (s.charAt(0) != '>' )
    {
      //System.out.println(s);
          genome = genome.append(s);
    }
```

```
    }

    fr.close();
   }
   catch ( NoSuchElementException elementException )
   {
      System.err.println( "File is corrupted." );
      input.close();
      System.exit( 1 );
   }
   catch ( IllegalStateException stateException )
   {
      System.err.println( "Reading aborted." );
      System.exit( 1 );
   }
   return genome;
}


//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;
  }
 findMax(F);
 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);
 double max = findMax(F);
 System.out.println("Max of  frequencies "+ max);
 return F;
}


//Maximal frequency is found
private static double findMax(double[] F)
{
 double max = 0;
 for(int i = 0; i< F.length; i++)
 if (F[i] > max ) max = F[i];
 //System.out.println("Max of  frequencies "+ max);
 return max;
}


//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;
 if (printAll)
System.out.println("Relative 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)
  System.out.println(F1[i] + "  " + F2[i]);
    }
   return d;
  }



 //Reports a matrix with the Euclidean n-gram distances
 //between
 //all possible pairs of proteomes in listOfFiles
 public static void calculateMatrix() throws IOException
 {
  StringBuffer proteome = new StringBuffer("");
  //n-gram analysis of all proteomes
  for (int i = 0; i < listOfFiles.length; i++)
   {
String file = listOfFiles[i].getName();
path = nameOfFolder + "/" + file;
proteome  = readData(path);
freq1 = nGramsFrequency(n, proteome);
//Results are kept in a matrix
for (int j = 0; j < freq1.length; j++)
 nGramFreqMatrix[j][i] = freq1[j];
   }
  //Euclidean n-gram distances are found
  int L = (int) Math.pow(lengthAlf, n);
  for (int i = 0; i < listOfFiles.length; i++)
   {
 for (int k = 0; k < L; k++)
 freq1[k] = nGramFreqMatrix[k][i];

 for (int j = 0; j < listOfFiles.length; j++)
  {
```

```
 //Euclidean distance between freq1 and freq2
 for (int k = 0; k < L; k++)
 freq2[k] = nGramFreqMatrix[k][j];
 nDistanceMatrix[i][j] =
 euclideanDistance(N, freq1,freq2);
  }
   }
 }


 public static void amplifyMatrix()
{

 //Min is found
 double min = 10;
 for (int i = 0; i < listOfFiles.length; i++)
 {
for (int j = i+1; j < listOfFiles.length; j++)
 if (nDistanceMatrix[i][j] < min)
 min = nDistanceMatrix[i][j];
 }

 //Amplifying factor
 double factor = 10;
 double OrderOfFactor = 10;
 //Factor is calculated
 double g = min;
 while (g < OrderOfFactor)
  {
  g = g*factor;
  factor = 10*factor;
}



 for (int i = 0; i < listOfFiles.length; i++)
  {
for (int j = 0; j < listOfFiles.length; j++)
```

```
 nDistanceMatrix[i][j] =  factor* nDistanceMatrix[i][j];
 System.out.println();
  }
}


public static void printMatrix()
{
 System.out.println( "\nAmplified and rounded " +
              " Matrix of Euclidean n-gram distances" );
 for (int i = 0; i < listOfFiles.length; i++)
  {
for (int j = 0; j <= i; j++)
 System.out.print((int) nDistanceMatrix[i][j] + "\t");
 System.out.println();
  }
}


 public static void main(String[] args) throws IOException
  {
getNamesOfFiles(nameOfFolder);
//To print additional information, change to true
printAll = false;
System.out.println( "\nLetters per n-gram = " + n );
System.out.println( "Lenght of the complete alphabet = " +
                                      lengthAlf);
System.out.println( "Number of n-grams in vector " +
                                  "SelfEng = " + N);

    calculateMatrix();
    amplifyMatrix();
    printMatrix();
   }

}//End of main class I116 EuclideanDistMatrix2
```

Bibliography

[1] Darnell J, Lodish H, Baltimore D (1986) *Molecular cell biology*. Scientific American Books.

[2] Gelbukh A, Sidorov G (2001) Zipf and Heaps Laws' Coefficients Depend on Language
http://www.gelbukh.com/CV/Publications/2001/CICLing-2001-Zipf.htm
Verified 21/V/2013

[3] Hao B 2013 Publications
http://power.itp.ac.cn/ hao/
Verified 21/V/2013

[4] Hao B 2013b Server
http://tlife.fudan.edu.cn/cvtree/
Verified 21/V/2013

[5] Microsoft Web N-gram Services (2013)
http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx
Verified 21/V/2013

[6] mxtodis123 (2013)
http://reclaimingthedarkgoddess.blogspot.com/2010/08/bachue.html
Verified 21/V/2013

[7] NCBI (2013) FASTA
http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml
Verified 21/V/2013

[8] Nettle D (1999) Linguistic Diversity, Oxford University Press, Oxford.

[9] NIH (2013)
http://www.niaid.nih.gov/topics/immunesystem/Pages/default.aspx
Verified 21/V/2013

[10] Powers (2013) Applications and Explanations of Zipf's Law
http://acl.ldc.upenn.edu/W/W98/W98-1218.pdf
Verified 20/V/2013

[11] Rodríguez J (2010) Estadística y decisiones. Vol II, multivariado.
http://www.evoljava.com/dl/EstVol2.pdf
Verified 21/V/2013

[12] Wikipedia 2013 Zipf's law

`http://en.wikipedia.org/wiki/Zipf%27s_law`

Verified 21/V/2013

Index