

University of Padua
Department of Mathematics
Master degree course in Computer Science

Bioinformatics Project

Resequencing and analysis of *Lactobacillus casei* genome

Student: Enrico Savoca, Mat.: 1132805

Professor: Giorgio Valle

Vicenza, 03/02/2016

Academic year 2013/2014

Summary:

[1 Introduction](#)

[2 Preparation](#)

[3 Project tasks](#)

[3.1 Installation of tools](#)

[3.2 Alignment of reads on the genome](#)

[3.3 First analysis of mate pairs](#)

[3.3.1 Insert Lengths](#)

[3.3.2 Mean and standard deviation](#)

[3.4 Create tracks with sequence and physical coverage](#)

[3.5 Count kmers of a given length](#)

[4 Conclusions and observations](#)

[5 About the program](#)

[Sources and Bibliography](#)

[Appendix A: Hardware most relevant properties](#)

[Appendix B: Code](#)

1 Introduction

Bioinformatics is a large field of science based on several disciplines like computer science, statistics, biology and chemistry. It encompasses a lot of biological disciplines like genomics, analysis of gene and protein expression.

The aim of the project was to resequence a given genome in order to gather several informations about structural variations.

The given genome is of a bacterium called *Lactobacillus casei* and its genome is 3'079'196 bases long. A list of mate pairs were given to map them on the genome and produce a resequencing of the genome of a similar *Lactobacillus*. After resequencing, it's possible to compare the two different genomes, in order to find structural variations.

2 Preparation

A linux version was used to achieve the project tasks: Ubuntu 14.04.3 LTS.

Sometimes, in the paper, will be provided some data about performance of algorithms.

These informations are related to a specific hardware. See Appendix A for these informations.

The reference genome (.fasta) and the reads (.fastq) have been downloaded from the professor site and put in a chosen folder. The executable file "GenomeAnalyzer.out" has been put in the same folder in order to work.

To execute the .out file on the command line:

```
./GenomeAnalyzer.out
```

If a compilation is required, it must be assured that the folder contains all the following files as in this structure:

```
folder/  
  GenomeAnalyzer.cpp  
  cpp/  
    generate_deviations.cpp  
    kmer_frequency.cpp  
    coverage.cpp  
    genomic_inserts.cpp
```

To compile on the command line, using a c++ compiler (in this case was used "g++"):

```
g++ GenomeAnalyzer.cpp -o GenomeAnalyzer.out
```

3 Project tasks

It has been chosen to accomplish the first five tasks of the project and the last one:

- install BWA, samtools and IGV on your computer;
- use BWA to align the sample fastq reads on the L. casei reference genome;
- read the sam file and for each mate pair work out the length of the genomic insert; then calculate the mean and standard deviation of the inserts, possibly discarding those that are totally out of range; a plot of the lengths distribution may help to evaluate the sizes;
- create tracks (wig+tdf) with the sequence coverage;
- create tracks (wig+tdf) with the physical coverage;
- Count all kmers of a given length and create tracks with the kmer frequency; it would be nice to have different track with different values of k.

For each of these tasks, will be explained how they were fulfilled and which informations emerge from the results.

3.1 Installation of tools

The tools were installed through the apt-get command on linux terminal.

To install BWA:

```
sudo apt-get install bwa
```

To install Samtools:

```
sudo apt-get install samtools
```

A different path was followed to install IGV, because the tool version gathered through apt-get doesn't work well. This tool was downloaded from <https://www.broadinstitute.org/software/igv/download>.

3.2 Alignment of reads on the genome

The alignment on the reference genome, contained in the file `Lactobacillus_casei_genome.fasta`, was done using bwa. Before alignment of the reads on it, it's necessary to index the genome.

On terminal:

```
bwa index Lactobacillus_casei_genome.fasta
```

The alignment is done using two files with the fastq format. They contain the sequences and their quality. Each sequence is a read and on each file there's a mate. Thus, for each sequence in the first file, we have another sequence in the second one, that is its mate. In this case they are mate pairs.

The following command maps the reads on the reference genome:

```
bwa mem -t 2 Lactobacillus_casei_genome.fasta lact_sp.read1.fastq lact_sp.read2.fastq  
>> alignment.sam
```

This command aligns the reads on the *Lactobacillus casei* genome and puts the result on a file called "alignment.sam". The "-t" option is used to define how many threads to create to complete the operation; in this case 2. The execution required several minutes and at the end of it a file .sam was created. This file contains a huge number of rows and for each row there are information about each read. Pairs of consecutive rows are mate pairs.

3.3 First analysis of mate pairs

As third task, was required to calculate the inserts lengths, their mean and the standard deviation of the inserts.

3.3.1 Insert Lengths

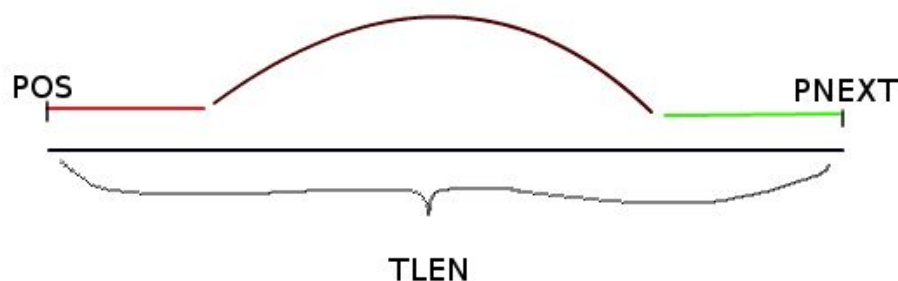
This value is calculated from a SAM file. Surfing the web to find the definition of insert length, it's possible to notice that there isn't a globally recognized definition of it. This size is considered in lots of ways. For example, some bioinformaticians call "insert length" the gap between two reads (the two mate pairs) and others call it the distance between them that encompasses their length. Even though there isn't a standard, the SAM Format specification offers some useful informations. In particular, three fields that are present in the SAM must be considered:

- TLEN: signed observed Template LENgth;
- POS: POSition of the first matching base;
- PNEXT: Position of the primary alignment of the NEXT read in the template.

According to TLEN definition, *"if all segments are mapped to the same reference, the unsigned observed template length equals the number of bases from the leftmost mapped base to the rightmost mapped base. The leftmost segment has a plus sign and the rightmost has a minus sign."*

Also, must be noticed that the distance between the two positions, POS and PNEXT, is equal to TLEN. Thus, it's possible to choose which values to use to calculate the requested length.

Firstly, was decided to consider as "insert" the distance between the leftmost and the rightmost based of two mate pairs.



For each read in the SAM file, we have its starting position (POS) and the starting position of the next one (PNEXT), that it's defined in the consecutive row if it is its mate. In the current

work, this distance is called “insert length” and it’s calculated, for each read, as the absolute value of the difference between POS and PNEXT.

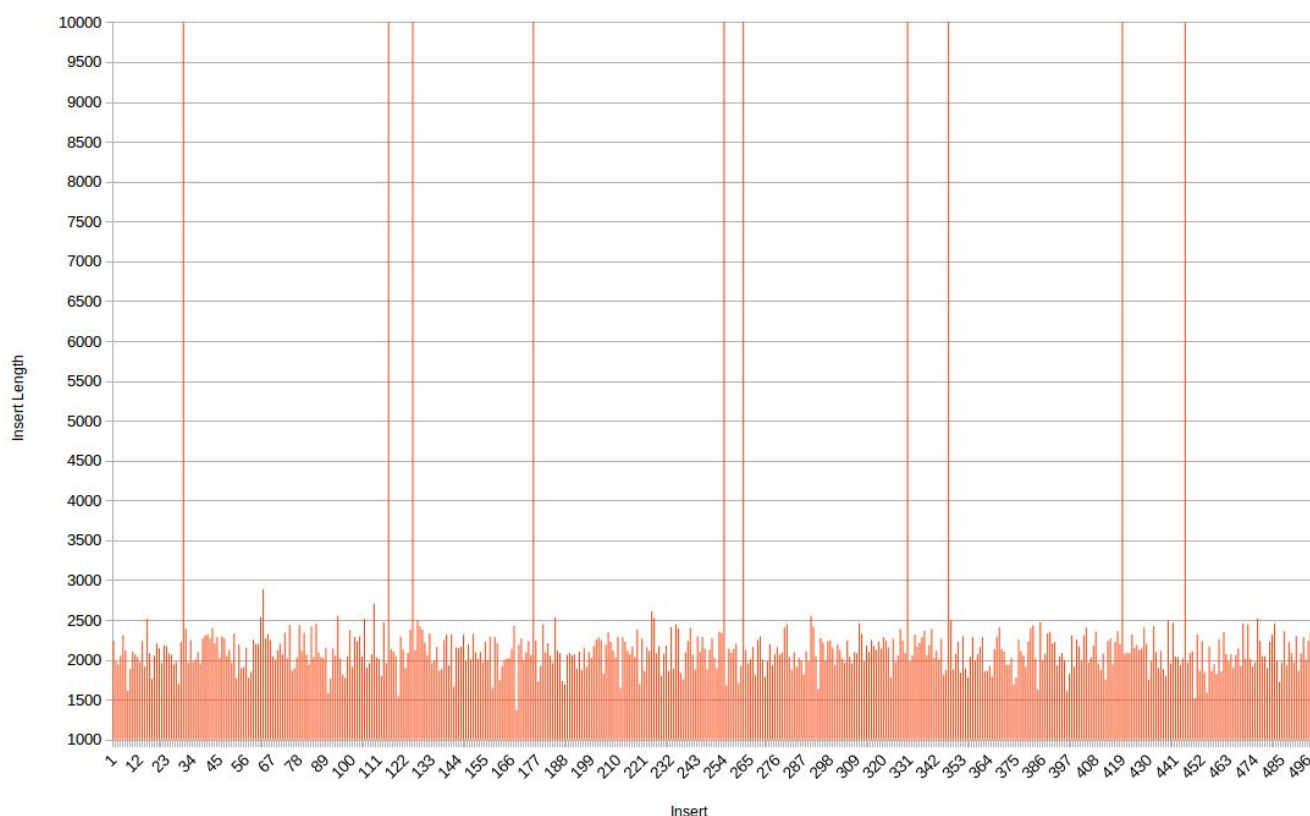
$$\text{Insert_Length} = \text{abs}(\text{POS} - \text{PNEXT})$$

For each mate pairs, it was calculated only one time, for one of the two reads: the one that has a plus sign in TLEN (the other with a minus sign has the same size).

Also TLEN could have been used for this operation instead of using POS and PNEXT, but according to the following page,

<http://sourceforge.net/p/samtools/mailman/message/32149215/>

it isn’t always equal to the insert length: this explain the choice of POS and PNEXT. After this analysis, it was created a program to calculate the insert lengths for each insert (in the Appendix B can be found the code of it). It produces a .csv file with a list of insert sizes that can be read by a program like calc or excel to create graphs. All insert sizes equal to zero were discarded during the execution and the result of the program was plotted in a bar chart. What is possible to notice from the following graph is that there are some values of insert lengths that are totally out of range. These values correspond to instrument errors of measuring. Indeed, comparisons between these erroneous values and the total number of values brings to notice that they are less than 1% of the total measures. Further analysis reported on the current document will demonstrate it better. Another information given by the graph is that all the inserts lengths are included between the values of 1500 and 3000. (For readability reasons, the range of Y axis on the chart was truncated to 10’000 even though some inserts have a value around 1’000’000. Also, In plotting, only the first 500 inserts and related insert lengths were taken).



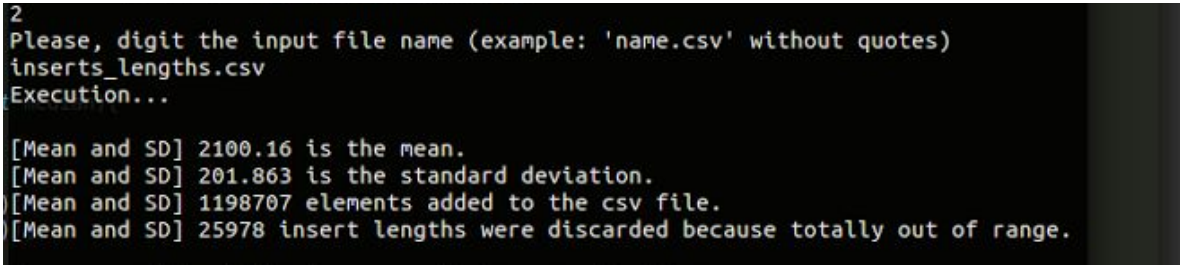
3.3.2 Mean and standard deviation

A module was programmed to accomplish the next part of the task: calculate mean and standard deviation of insert lengths.

A simple mean of the insert lengths could have brought to a really bad value of it. This because values of insert lengths that are totally out of range are summed too. These values bring the mean to reach high mistaken values, so they must be discarded.

To solve the problem, it was calculated the median first. The interesting property of the median is that its value depends in great part from the total number of inserts that have quite a similar value. Before this calculus, the list of inserts was sorted. Thanks to sorting, only the last few elements of the list have out-of-range values. The median value corresponds to the element in the middle of the list. Thus if the error is small, median is always included between the expected values.

The median was used to decide which values to keep to calculate the mean, and which one to discard.



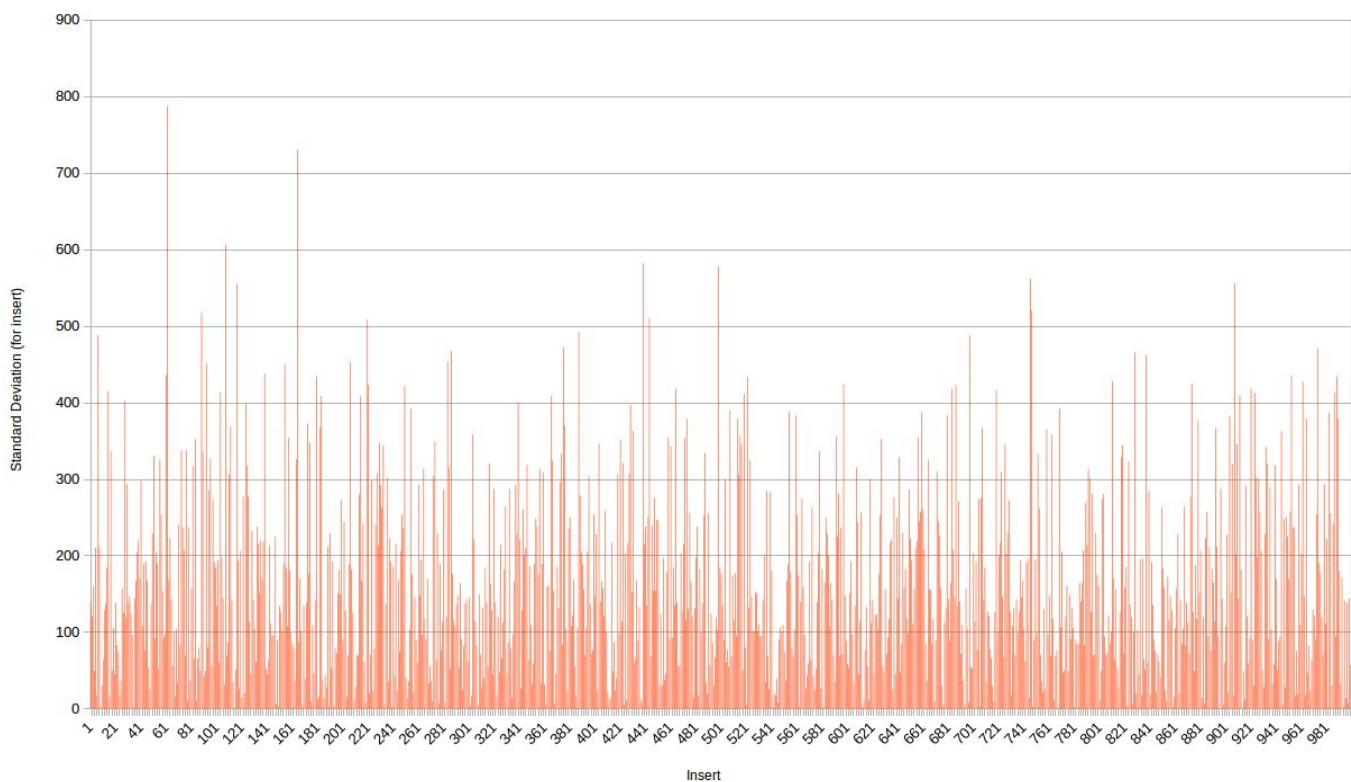
```
2
Please, digit the input file name (example: 'name.csv' without quotes)
inserts_lengths.csv
Execution...

[Mean and SD] 2100.16 is the mean.
[Mean and SD] 201.863 is the standard deviation.
[Mean and SD] 1198707 elements added to the csv file.
[Mean and SD] 25978 insert lengths were discarded because totally out of range.
```

The upper screenshot shows what is printed by a program execution using as input the insert lengths values calculated before.

The calculated mean is 2100.16; this means that, if the two reads have a length of 100 bases, the mean gap between them is 1900 bases long. The standard deviation value is 201.863. Another important information given from the program is about the number of erroneous values. Considering the plotted values, it's easy to calculate that they correspond to 0,02% of all the values, hence they can be ignored.

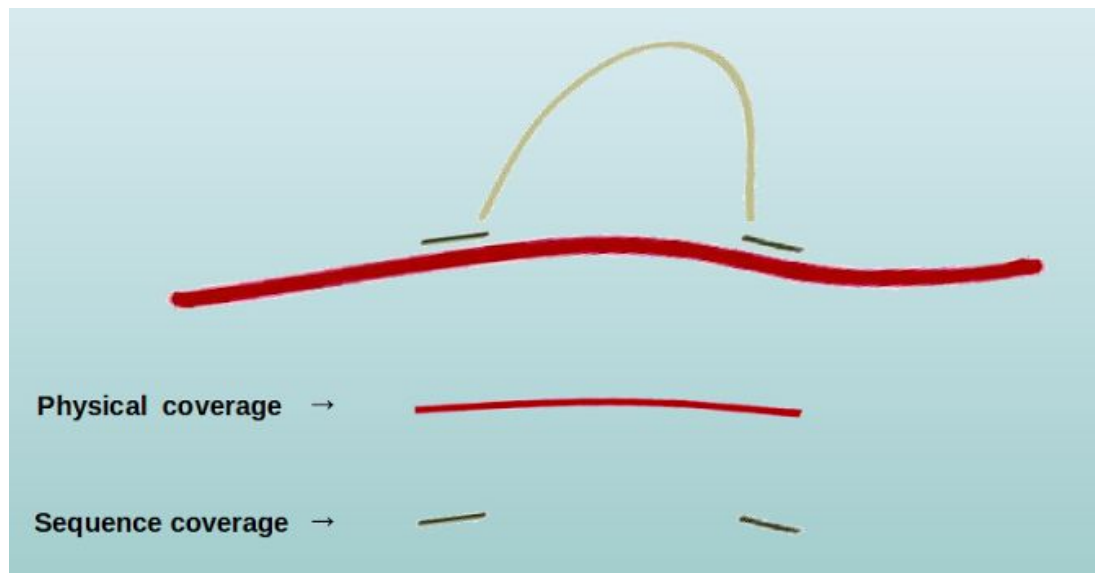
The graph below shows that the insert lengths have a value of 2100 ± 800 , so the values are included in an interval that goes from 1300 bases to 2900 bases (for readability reasons, only the first 1000 value were represented on the following graph).



3.4 Create tracks with sequence and physical coverage

These tasks (calculate the physical and sequence coverage) are described together because they bring to the same conclusions. Also, the analysis of the two files gives more informations than separate analysis of them.

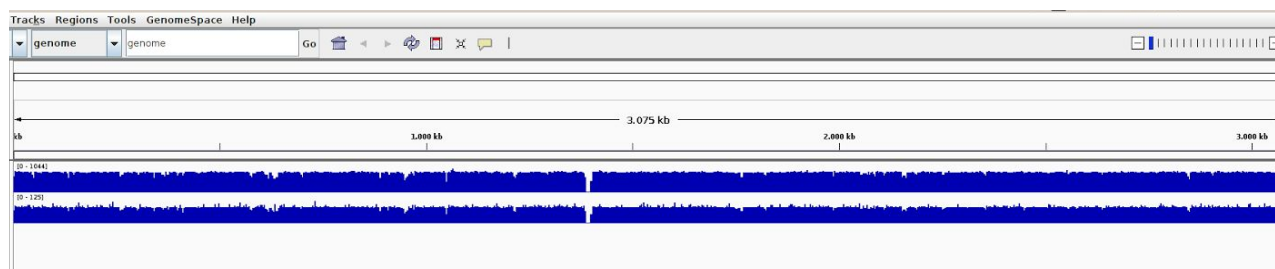
The sequence coverage is calculated from the sam file. For each read is defined a starting position in the genome. Each read covers the genome from its starting position for its fragment length. In the current case, for each read, we have, according to the SAM Format Specification, the segment sequence: SEQ (the sequence length is also readable from the CIGAR on the file). Assuming for instance that we have a sequence of length 100 and to have as starting position 15, then we must consider the genome as covered from the 15th base to the 115th one. This calculus is made for each read and let to calculate the sequence coverage on the genome. So, in the current case, if a base is covered by n reads, it will have n as value of sequence coverage. The calculus of the physical coverage is similar to the sequence one. Instead of consider only the reads, we use the entire inserts. So if a genome is covered by an insert, it is covered for the entire length of the insert. Hence, we evaluate the informations of one read for insert (the one with the plus sign on the TLEN field). If both the reads have been considered for each insert, the physical coverage would have been twice the correct one. This could be also noticed in the image below, taken from the slides of the professor.



A program has been created to calculate the sequence and the physical coverage. Both of the created algorithms firstly take from the SAM file the length of the genome and instantiate a vector of that dimension. Then while reading the file, the vector is populated with the values of coverage discussed before. After these operations, the results are put on two .wig files.

A WIG file (.wig) defines a data track and the extension is required by IGV to read it as a wiggle file. IGV is a genome browser that with its interface let to discover several informations on the genome: for instance, structural variations.

The following image shows the genome with the two tracks generated.



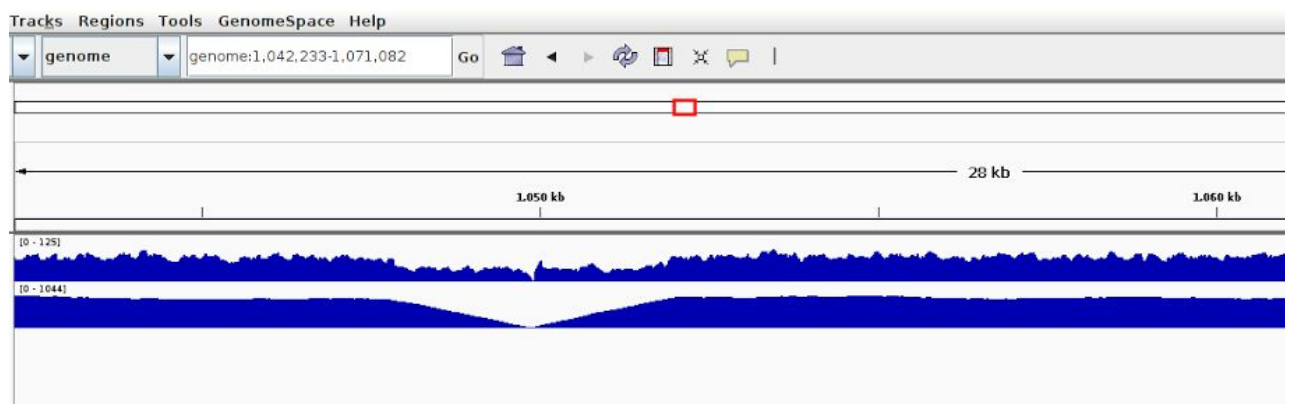
In the figure, the first one represents the sequence coverage, while the second is the physical coverage.

The first thing that is possible to notice is in the middle of the genome.

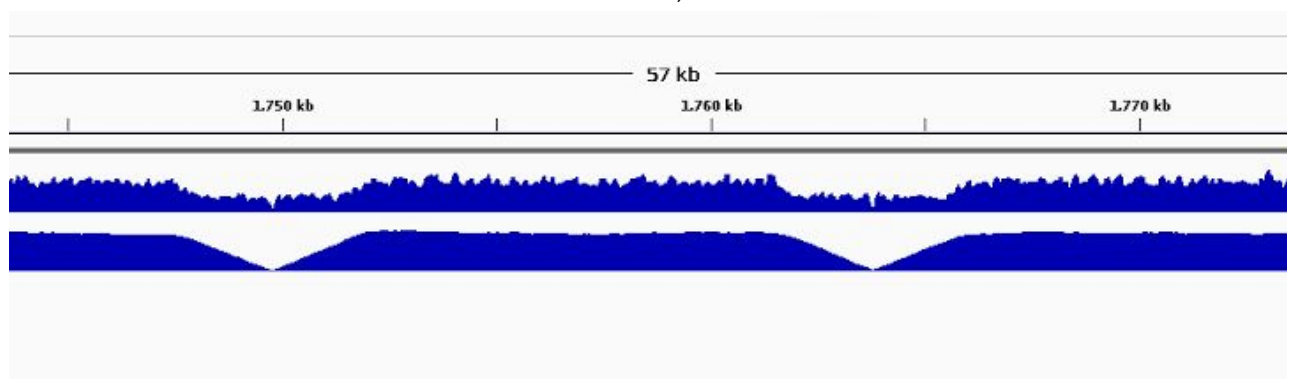


This type of structural variation is a long deletion and its length is of about 15kb (kilobases). Some others structural variations can be found using the genome browser:

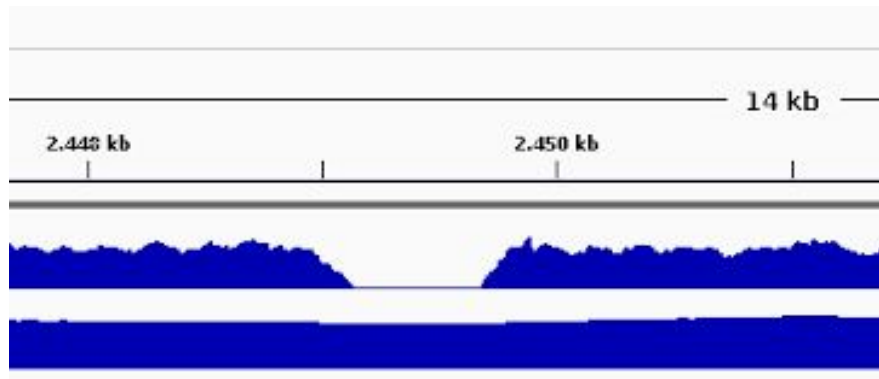
- A long insertion at position 1'050 kb;



- An inversion between 1'750 kb and 1'765 kb;



- A short deletion of less than 1 kb between 2'448 kb and 2'450 kb.



Other structural variations, even if not reported here, were found in the genome. They were mainly short insertions.

The study of these mutations compared with characteristics of the two bacteria can bring to discover various informations not only about these two organisms, but also about gene expression.

Firstly The two organisms can be studied from the point of view of the evolution. For instance, these structural variations could have be related to the diversification of the two organisms. In fact, evolution is strictly binded to genome mutations.

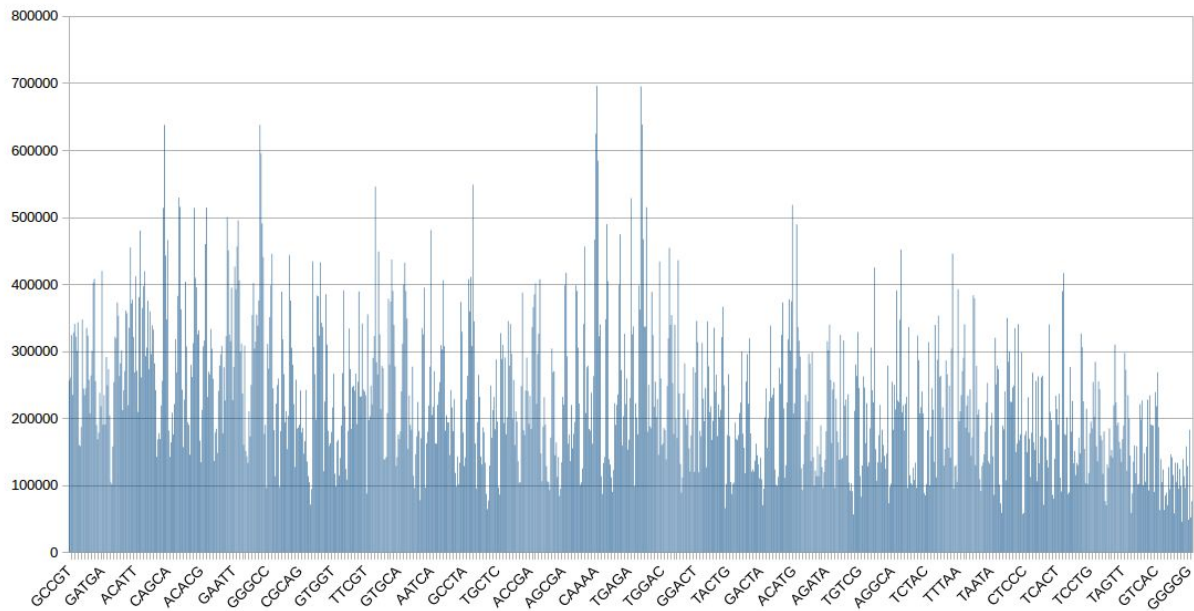
Another study of the structural variations could be done from the point of view of gene expressions, so it is related to more organisms. Comparing the proteins present in the two bacteria and considering their function, it's possible to understand why evolution occurred. Also, it's possible to discover which genes are responsible of the expression of some particular proteins that could be absent in one of the bacteria. For example considering the long deletion in the genome, we expect to find that some proteins are present only in the *Lactobacillus Casei* and they are absent in the other type of *lactobacillus* because not expressed.

The tasks required also to create .tdf files. These data files contain data that has been preprocessed for faster display in IGV.

3.5 Count kmers of a given length

The last accomplished task was to count the kmers of a given length, taken from the sam, and analyze them.

An algorithm that achieves this was created. It takes all the reads of the SAM file and, after its execution, puts the result on a .csv file. This program counts how many times each kmer can be found in the reads. The size chosen for the kmers was 5, so they are 5-mers. The considered process required more than 2 hours of execution to complete its task.



On the axis scale only some kmers are represented.

Considering the repetitions of each kmer we can state for each kmer that if a kmer x is repeated n times and another kmer y is repeated m times, we can quite certainly say that x is always less than 15-times m . That means that they are on the same order of size. Another conclusion can be deduced sorting the kmers on their repetitions. A function created for this aim sorts the list of kmers and the result is the following one:

Kmer	Frequency
AAAAA	695538
TTTTT	694591
TTTTG	637860
ATCAA	637366
TTGAT	637233
CAAAA	624352
ATTTT	594933
AAAAT	584138

These are the first 8 kmers and the list that includes them is sorted from the most frequent to the least one. Concerning the present list, we can see that the frequency of bases A and T is high instead of the other bases. With longer kmers it would have been possible to align them

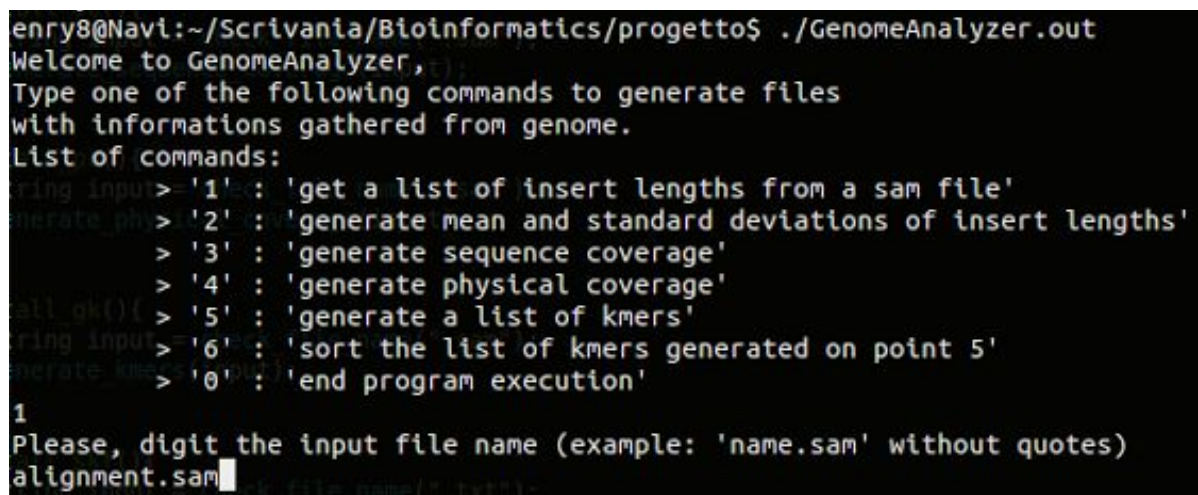
together to consider alignments like the following: for example the second with the third kmer (TTTTTG) or the first with the last kmer: AAAAAT.

4 Conclusions and observations

The analysis of the genome has provided several interesting results. Some points must be considered with relevance. For example, even between relatively small genomes of two similar bacteria there were found a lot of structural variations. Considering that, it's reasonable to think that the complexity of these can analysis increase rapidly and it's directly proportional with the genome length. With longer genomes, could be extremely difficult to compare them, but some heuristic techniques, together with the use of more computational resources, can help. Regarding the kmers analysis, a deeper study of them and other elaborations of the data could generate more relevant informations. For instance, it could be possible to establish which bases are more present in the genome or to calculate their distribution. Bioinformatics is a field of science that will grow a lot. There's a lot of things to discover and the improvement of computational resources can bring to results that weren't reachable in past. However, in my opinion the most important result that should be accomplished first it's to create a global standard. Approaching to this subject using the material found on internet is challenging. There isn't a common vision of some definitions like in a well defined science. Often, same things are called with different names. Thus, to conclude, this field must be seen as challenging because of the intrinsic difficult on approaching it, but also challenging because it gives really interesting results.

5 About the program

In order to offer a good tool for genome analysis, the component described above were integrated in a unique program. The behavioral design pattern "command pattern" was applied on the realization of the basic structure.



```
enry8@Navi:~/Scrivania/Bioinformatics/progetto$ ./GenomeAnalyzer.out
Welcome to GenomeAnalyzer,
Type one of the following commands to generate files
with informations gathered from genome.
List of commands:
bring input> '1' : 'get a list of insert lengths from a sam file'
enerate ph> '2' : 'generate mean and standard deviations of insert lengths'
> '3' : 'generate sequence coverage'
> '4' : 'generate physical coverage'
call_gk()> '5' : 'generate a list of kmers'
bring input> '6' : 'sort the list of kmers generated on point 5'
enerate_kme> '0' : 'end program execution'
1
Please, digit the input file name (example: 'name.sam' without quotes)
alignment.sam
```

As in the screenshot above, the program offers to the user a list of choices that he can select to generate useful data from the genome, like coverage values or lists of insert lengths. All the code can be read in the second appendix: Appendix B.

Sources and Bibliography

Material provided on the professor's site:

<http://didattica.cribi.unipd.it/genomica/bioinfoforinfo/15-16/index.xml>

SAM Format Specification

<https://samtools.github.io/hts-specs/SAMv1.pdf>

About mate pairs insert length

<http://thegenomefactory.blogspot.it/2013/08/paired-end-read-confusion-library.html>

Use of POS and PNEXT instead of TLEN

<http://sourceforge.net/p/samtools/mailman/message/32149215/>

Appendix A: Hardware most relevant properties

Hardware properties:

CPU: Dual core Intel Core i5 CPU M 480;

RAM memory: 4GB;

Graphic card (1): Intel Core Processor Integrated;

Graphic card (2): NVIDIA GF108M [GeForce GT 540M].

Appendix B: Code

GenomeAnalyzer.cpp

```
#include "cpp/genomic_inserts.cpp"
#include "cpp/generate_deviations.cpp"
#include "cpp/coverage.cpp"
#include "cpp/kmer_frequency.cpp"
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

string check_file_name(string extension){
    bool check=false;
    string input_name;
    do{
        cout<<"Please, digit the input file name (example: 'name'+extension+'\" without
quotes)\n";
        cin>>input_name;
        ifstream file;
        file.open(input_name.c_str());
        if((input_name.find(extension) != string::npos)&&(file))
```

```
        check = true;
    else
        cout<<"This file doesn't exist or the extension is wrong.\n";
        file.close();
    }while(!check);
    cout<<"Execution...\n";
    return input_name;
}

void call_gil(){
    string input = check_file_name(".sam");
    generate_genomic_inserts(input);
}

void call_gmsd(){
    string input = check_file_name(".csv");
    generate_deviations(input);
}

void call_sc(){
    string input = check_file_name(".sam");
    generate_sequence_coverage(input);
}

void call_pc(){
    string input = check_file_name(".sam");
    generate_physical_coverage(input);
}

void call_gk(){
    string input = check_file_name(".sam");
    generate_kmers(input);
}

void call_skf(){
    string input = check_file_name(".txt");
    sort_kmers(input);
}

int main(){
    int command;
    bool stop=false;
    cout<<"Welcome to GenomeAnalyzer,\n";
    do{
        command = -1;
        cout<<"Type one of the following commands to generate files \n"
            "with informations gathered from genome.\n"
            "List of commands:\n"
            "\t > '1' : 'get a list of insert lengths from a sam file'\n"
            "\t > '2' : 'generate mean and standard deviations of insert
```

```

lengths'\n"
        "\t > '3' : 'generate sequence coverage'\n"
        "\t > '4' : 'generate physical coverage'\n"
        "\t > '5' : 'generate a list of kmers'\n"
        "\t > '6' : 'sort the list of kmers generated on point ""5""'\n"
        "\t > '0' : 'end program execution' \n";
    cin>>command;
    switch (command){
        case(1):
            call_gil();
            break;
        case(2):
            call_gmsd();
            break;
        case(3):
            call_sc();
            break;
        case(4):
            call_pc();
            break;
        case(5):
            call_gk();
            break;
        case(6):
            call_skf();
            break;
        case(0):
            cout<<"Program execution ended. \n";
            stop = true;
            break;
    }
    }while(!stop);
}

```

genomic_inserts.cpp

```

#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <sstream>
#include <cstdlib>
using namespace std;

void generate_genomic_inserts(string input_name){
    ifstream input_genome;
    ofstream inserts_length;

```



```
input_genome.open(input_name.c_str());
inserts_length.open("inserts_lengths.csv");
bool analyze_line = false;
string line = "";
getline(input_genome, line); // it doesn't read the header row
while(getline(input_genome, line)){
    int position=0, next=0, orientation_check = 0;
    string word;
    istringstream iss(line, istringstream::in);
    int i = 0;
    while( iss >> word ){
        if(i==3){
            istringstream(word) >> position;
        }
        if(i==7){
            istringstream(word) >> next;
        }
        if(i==8){
            istringstream(word) >> orientation_check;
        }
        i++;
    }
    if(orientation_check>0)
        inserts_length<<abs(next-position)<<"\n";
}
input_genome.close();
inserts_length.close();
cout<<"[Insert-lengths] Done\n\n";
}
```

generate_deviations.cpp

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

int strToInt(string line){
    int number=0;
    istringstream ss(line);
    ss>>number;
    return number;
}
```

```
float getMedian(vector<int>& elems, int counter){
    int position = -1;
    float result = 0;
    position = counter/2;
    if(counter%2 == 1){
        result = elems[position];
    }
    else{
        result = (elems[position]+elems[position-1])/2;
    }
    return result;
}

float getSD(vector<int>& elems, float mean, float median){
    float result = 0;
    int counter = 0;
    vector<int>::const_iterator it;
    for(it= elems.begin(); it!=elems.end(); it++){
        if((*it < median*1.5) && (*it > median/2)){
            float diff = (*it)-mean;
            result+=pow(diff ,2);
            counter++;
        }
    }
    result = result/(counter);
    result = sqrt(result);
    return result;
}

void print_results(double mean, double sd, int counter, int discarded_out_of_range){
    cout<<"\n";
    cout<<"[Mean and SD] "<<mean<<" is the mean.\n";
    cout<<"[Mean and SD] "<<sd<<" is the standard deviation.\n";
    cout<<"[Mean and SD] "<<counter<<" elements added to the csv file.\n";
    cout<<"[Mean and SD] "<<discarded_out_of_range<<" insert lengths were discarded
because totally out of range.\n";
    cout<<"\n";
}

void generate_deviations(string input_name){
    ifstream inserts_lengths;
    ofstream deviations;

    double mean = 0, dev_mean = 0;

    int counter = 0;

    string line = "";
    vector<int> elements, ordered_elements, elements_deviation;
    inserts_lengths.open(input_name.c_str());
```

```

while(getline(inserts_lengths, line)){

    int number = strToInt(line);

    ordered_elements.push_back(number); // zeros must be considered to calculate
the median
    counter++;
    if(number != 0){ //discard inserts with length = 0
        elements.push_back(number);
    }
}
sort(ordered_elements.begin(), ordered_elements.end());
float median = getMedian(ordered_elements, counter);

counter = 0;
vector<int>::const_iterator it;
for(it= elements.begin(); it!=elements.end(); it++){
    if((*it < median*1.5) && (*it > median/2)){
        mean+=*it;
        counter++;
    }
}
mean = mean / (counter);
deviations.open("deviations.csv");
counter = 0;
int discard_out_of_range_elems = 0; // elements out of range
for(it=elements.begin(); it != elements.end(); ++it){
    if((*it < median*1.5) && (*it > median/2)){
        elements_deviation.push_back(abs(*it-mean));
        deviations<<(abs(*it-mean))<<"\n";
        counter++;
    }
    else discard_out_of_range_elems++;
}
float sd = getSD(elements, mean, median);
print_results(mean, sd, counter, discard_out_of_range_elems); // on terminal
inserts_lengths.close();
deviations.close();
}

```

coverage.cpp

```

#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <sstream>
#include <vector>

```

```
#include <cstdlib>
using namespace std;

int get_length(string line){
    istringstream iss(line, istringstream::in);
    string word = "";
    iss >> word;
    int position = line.find("LN:");
    if(position > -1){
        word = line.substr(position+3, line.length()-1);
    }
    istringstream ist(word);
    int length = 0;
    ist >> length;
    return length;
}

void generate_sequence_coverage(string input_name){
    ifstream input_genome;
    ofstream sequence_coverage;
    input_genome.open(input_name.c_str());
    sequence_coverage.open("sequence_coverage.wig");
    vector<int> sequence_coverage_vector;
    string line = "";
    int length=0;
    getline(input_genome, line);
    if(line[0] == '@'){
        istringstream iss(line, istringstream::in);
        length = get_length(line);
    }
    sequence_coverage_vector.resize(length, 0);
    while(getline(input_genome, line)){
        int position=0;
        string word;
        istringstream iss(line, istringstream::in);
        int i = 0;
        bool check_row = true;
        int orientation_check = 0;
        while( iss >> word ){
            int seq_length= 0;
            if(i==1){
                int bit_control = 0;
                istringstream(word) >> bit_control;
                if((bit_control & 3) != 3){
                    check_row = false;
                }
            }
            if(i==3){
                istringstream(word) >> position;
            }
        }
    }
}
```

```

        if(i==8){
            istringstream(word) >> orientation_check;
        }
        if(i==9){
            seq_length = word.length();
            if(check_row){
                if(orientation_check>0){
                    for(int it = position; it!= position+seq_length;
it++){
                        sequence_coverage_vector[it]++;
                    }
                }
                else{
                    for(int it = position; it!=0 && it!=
position-seq_length; it--){
                        sequence_coverage_vector[it]++;
                    }
                }
            }
        }
        i++;
    }

    sequence_coverage << "fixedStep chrom=genome start=1 step=1 span=1\n";
    for(vector<int>::iterator it = sequence_coverage_vector.begin(); it!=
sequence_coverage_vector.end(); it++)
        sequence_coverage << *it << "\n";
    input_genome.close();
    sequence_coverage.close();
    cout<<"[sequence_coverage] Done\n\n";
}

void generate_physical_coverage(string input_name){
    ifstream input_genome;
    ofstream physical_coverage;
    input_genome.open(input_name.c_str());
    physical_coverage.open("physical_coverage.wig");
    vector<int> physical_coverage_vector;
    string line = "";
    int length=0;
    getline(input_genome, line);
    if(line[0] == '@'){
        istringstream iss(line, istringstream::in);
        length = get_length(line);
    }
    physical_coverage_vector.resize(length, 0);
    while(getline(input_genome, line)){
        int start_position=0, end_position=0;
        string word;
        istringstream iss(line, istringstream::in);

```

```

        int i = 0;
        bool check_row = true;
        int orientation_check = 0;
        while( iss >> word ){
            int seq_length= 0;
            if(i==1){
                int bit_control = 0;
                istringstream(word) >> bit_control;
                if((bit_control & 3) != 3){
                    check_row = false;
                }
            }
            if(i==3){
                istringstream(word) >> start_position;
            }
            if(i==7){
                istringstream(word) >> end_position;
            }
            if(i==8){
                istringstream(word) >> orientation_check;
            }
            if(i==9){
                seq_length = word.length();
                if(check_row && orientation_check>0){
                    for(int it = start_position;
it!=physical_coverage_vector.size() && it!= end_position; it++){
                        physical_coverage_vector[it]++;
                    }
                }
            }
            i++;
        }
    }

    physical_coverage << "fixedStep chrom=genome start=1 step=1 span=1\n";
    for(vector<int>::iterator it = physical_coverage_vector.begin(); it!=
physical_coverage_vector.end(); it++)
        physical_coverage << *it << "\n";
    input_genome.close();
    physical_coverage.close();
    cout<<"[physical_coverage_vector] Done\n\n";
}

```

kmer_frequency.cpp

```

#include <iostream>
#include <fstream>
#include <string>
#include <cmath>

```

```
#include <sstream>
#include <cstdlib>
#include <vector>
#include <algorithm>
using namespace std;

class kmer{
private:
    string sequence;
    int frequency;
public:
    static const int dimension = 5;
    kmer(string seq){
        sequence = seq;
        frequency = 1;
    }

    kmer(string seq, int freq){
        sequence = seq;
        frequency = freq;
    }

    string get_sequence(){
        return sequence;
    }

    int get_frequency(){
        return frequency;
    }

    void increment_frequency(){
        frequency++;
    }

    bool operator<(const kmer& y) const{
        return(frequency < y.frequency);
    }
};

void get_kmers(string sequence, vector<kmer>& kmers){
    int size = sequence.size(), vector_size = kmers.size(), incr = 0;
    for(int i=0; i< size - kmer::dimension; i++){
        string analyz_kmer = sequence.substr(i, kmer::dimension);
        vector<kmer>::iterator it;
        for(it = kmers.begin(); (it!= kmers.end()) && (it->get_sequence() != analyz_kmer);
it++);
        if(it!=kmers.end()){
            it->increment_frequency();
        }
    }
}
```

```
        }else{
            kmer* x = new kmer(analyz_kmer);
            kmers.push_back(*x);
        }
    }
}

void generate_kmers(string input_name){
    cout<<"It can take some hours.\n";
    ifstream input_genome;
    ofstream kmers_output;
    vector<kmer> kmers;
    input_genome.open(input_name.c_str());
    string line = "";
    int c=0;
    while(getline(input_genome, line)){
        if(line[0] != '@'){
            int position=0, next=0;
            string sequence;
            istringstream iss(line, istringstream::in);
            int i = 0;
            while( iss >> sequence ){
                if(i==9){
                    get_kmers(sequence, kmers);
                }
                i++;
            }
        }
        c++;
    }
    kmers_output.open("kmers_frequency.csv");
    for(vector<kmer>::iterator it = kmers.begin(); it!= kmers.end(); it++){
        kmers_output<<it->get_sequence()<<"", "<<it->get_frequency()<<"\n";
    }
    input_genome.close();
    kmers_output.close();
    cout<<"[Generate Kmers frequency] Done\n\n";
}

void sort_kmers(string input_name){
    vector<kmer> kmers;
    ifstream input_genome;
    input_genome.open(input_name.c_str());
    string line = "";
    while(getline(input_genome, line)){
        istringstream iss(line, istringstream::in);
        string seq, temp; int freq;
        iss >> seq; iss >> temp; istringstream(temp) >> freq;
        kmers.push_back(kmer(seq, freq));
    }
}
```



```
sort(kmers.begin(), kmers.end());  
reverse(kmers.begin(), kmers.end());  
ofstream sorted_kmers;  
sorted_kmers.open("sorted_kmers_frequency.csv");  
for(vector<kmer>::iterator it = kmers.begin(); it!= kmers.end(); it++){  
    sorted_kmers<<it->get_sequence()<<" "<<it->get_frequency()<<"\n";  
}  
input_genome.close();  
sorted_kmers.close();  
cout<<"[Sort Kmers frequency] Done\n\n";  
}
```