Università di Padova

Dipartimento di Matematica

Corso di Laurea Triennale in Informatica

Progetto di Bioinformatica

Genoma Analyzer

software per il risequenziamento genetico

Alessandro Pol

Matricola: 1052596

Indice

Introduzione

Lo scopo del progetto consiste nello sviluppare un set di funzioni che permettano di individuare variazioni strutturali tra due genomi e ricavare dati statistici sul loro risequenziamento. A tale scopo vengono forniti il genoma del batterio Lactobacillus Casei (*Lactobacillus_casei_genome.fasta*) e due file contenenti le pair-end reads di un organismo della stessa specie (*lact_sp.read1.fastq* e *lact_sp.read2.fastq*).

Il seguente documento si dividerà in due parti: nella prima parte verrà mostrato come preparare l' input installando ed usando strumenti per la gestione dei dati provenienti dai sequenziatori; nella seconda parte verranno mostrate le funzionalità del software, spiegando le soluzioni algoritmiche che sono state adottate per svolgere i quesiti richiesti.
Infine verranno date le conclusioni riguardo i dati raccolti e le considerazioni finali.

Elaborazione Input

Obiettivi

La prima parte si occuperà di svolgere i seguenti punti:

- Installare BWA, samtools e IGV;
- Usare BWA per allineare le read con il genoma;
- Convertire il file SAM ottenuto in BAM;
- Ordinare e indicizzare il file BAM;
- Mostrare la coverage e le mate pairs in IGV;
- Trovare manualmente anomalie nell'allineamento.

Installazione BWA, samtools e IGV

Per installare BWA aprire il terminale e digitare il seguente comando:

sudo apt-get install bwa

Per samtools

sudo apt-get install samtools

IGV è stato installato via terminale ma scaricando il pacchetto dal sito:

https://www.broadinstitute.org/software/igv/download

Allineamento delle reads con il genoma

La prima operazione da eseguire è indicizzare il genoma usando BWA. Con tale operazione verranno mappate le mate pairs con il genoma rendendo così il risequenziamento più efficiente.
Aprire il terminale, posizionarsi nella cartella contenente il genoma e digitare il seguente comando:

bwa index Lactobacillus_casei_genome.fasta

Possiamo ora allineare le mate pairs con il genoma. Il genoma è contenuto in un file *.fasta* formato da un header e l'intera sequenza genomica. Le reads sono mate pairs e quindi divise in due file distinti. Ogni file

contiene la sequenza della read, la qualità del rilevamento ed un identificativo in cui viene specificato se si tratta della read destra (1) o sinistra (2). Le read sono divise in modo che ogni file contenga esclusivamente le reads di un lato.
Per allineare le reads al genoma usiamo il seguente comando:

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

Con tale comando indichiamo il numero di thread usati (due in questo caso), il genoma di riferimento ed i due file delle reads. L'ultimo attributo specifica che il risultato sarà salvato nel file *alignment.sam.*

Il file *.sam* conterrà ogni read dei *.fastq* file elencate a coppie. Ogni reads contiene informazioni riguardanti la mappatura ed il posizionamento all'interno del genoma. Tale file sarà l'input principale del software in esame.

Visualizzare la mappatura in IGV

Per poter avere una rappresentazione grafica della la mappatura tra le mate pairs ed il genoma di riferimento, utilizziamo il programma IGV.

Prima di tutto è necessario convertire il file SAM in uno BAM in modo da renderlo compatibile con il programma. Successivamente, ordinare le mate pairs in base al loro posizionamento nel genoma. Aprire il terminale e con samtools eseguire la seguente riga di comando

samtools view -bS alignment.sam | samtools sort -o lact_sorted

Infine, indicizzare il file *lact_sorted* per velocizzare la lettura dei dati durante la visualizzazione

samtools index lact_sorted

Aprire IGV, caricare il genoma ed il file *lact_sorted.bam.* Appariranno cosi le mate pairs mappate con il genoma di riferimento.

Trovare variazioni strutturali

Scorrendo il genoma possiamo trovare manualmente delle variazioni strutturali. Le zone rosse segnalano punti in cui ci sono delle *delezioni* mentre quelle blu segnalano le *inserzioni*.

Nella seguente si può notare un *inversione* tra le basi posizionate tra 1760 - 1890 kb

delle *delezioni*,

ed infine delle *inserzioni* corte.

Genoma Analyzer

Obiettivi

- Creare una WIG file con la physical covarage;
- Creare una WIG file con la sequence coverage;
- Calcolare la lunghezza di ogni insert presente nel file SAM. Calcolare media e deviazione standard scartando i dati fuori range e stampare i risultati;
- Creare una traccia con una percentuale di insert che hanno una lunghezza di n volte la deviazione standard sopra e sotto la media.

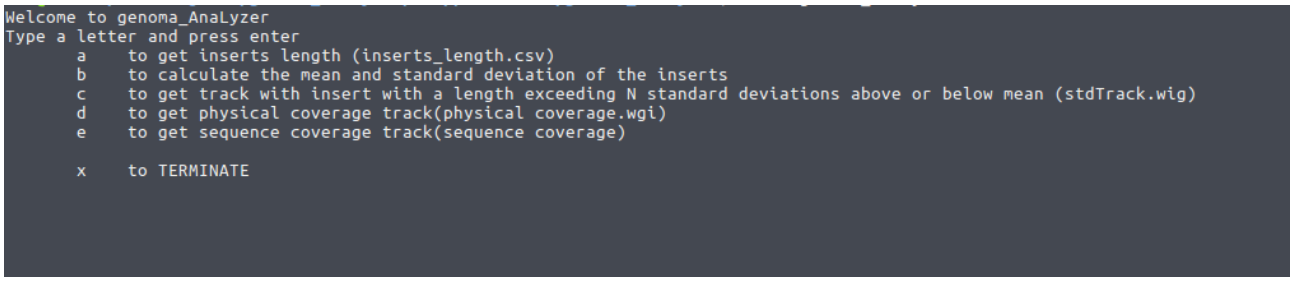
Installazione Scala e avvio programma

Il programma è stato sviluppato in Scala. Dato che la compilazione genera file bytecode, possiamo eseguire il programma usando la JVM. Apriamo la cartella contenente il file *genomaanalyzer.jar* e copiamo all'interno il file *alignment.sam* precedentemente creato.

Aprire il terminale e avviare il programma digitando

java - jar genomaanalyzer.jar

apparirà la seguente schermata



Physical Coverage e Sequence Coverage

La copertura delle mate-pairs nel genoma viene calcolata utilizzando il file SAM. Dalle specifiche del file interessano principalmente due campi:

- POS: indice che fa rimento alla prima base a sinistra mappata nel genoma.
- TLEN: lunghezza della mate-pair.

con questi indici sappiamo esattamente in che punto del genoma inizia una insert ed in che punto finisce.

Con tali informazioni possiamo calcolare la copertura del genoma semplicemente inizializzando un array di contatori della stessa lunghezza e per ogni insert presente, aumentare i contatori nei quali l'insert è mappata. Tale metodo si dimostra lento e poco efficiente in quanto ripete uguali operazioni nelle stesse celle.

È stata usata un' idea alternativa che permette di scorrere l'array una volta sola senza ripetere più operazioni.

L'idea sta nello scorrere in maniera sequenziale l'array e per ogni cella salvare il numero di insert che coprono la rispettiva base genomica. Per fare questo è necessario avere delle tabelle in cui verrà tenuta traccia degli indici iniziali e finali di ogni insert ed un contatore che memorizzi il numero di insert presenti all'indice *i* dell' array genomico.

A tale scopo sono stati create tre entità apposite:

- **weight:** contatore che rappresenta il numero di insert che copre una base all'iterazione *i;*
- **TableStart**: mappa ad accesso veloce in cui le chiavi sono gli indici iniziali di ogni insert mentre il valore corrisponde alla lista dei rispettivi indici finali. Viene inizializzata all'inizio delle operazioni leggendo ogni riga del file SAM. Vengono inserite solo le insert che rispettano un certa tipologia di copertura richiesta;
- **TableEnd**: mappa ad accesso veloce in cui le chiavi sono gli indici finali di ogni insert mentre il valore corrisponde al numero di insert che terminano in quel punto.

Ad ogni iterazione dell'array genomico, viene interrogata *tableStart* per verificare se all'indice *i* cominciano nuove insert. In caso positivo viene aumentato *weigth* in base al numero di indici finali presenti (cioè al

numero di insert che cominciano da quel punto). Gli indici finali ricavati vengono inseriti in *tableEnd* ed in caso di doppioni il contatore apposito aumenta.
Una volta che il peso è aggiornato, viene salvato nell'indice *i* nell'array genomico.
Prima del termine dell'iterazione viene interrogata la tabella *tableEnd* per verificare se nell'indice *i* termina qualche insert. In caso positivo viene sottratto il peso corrispettivo a weigth ed eliminate riga dalla tabella.

Una volta esaminato tutto il genoma viene creato i file in cui verranno scritti i dati raccolti nell' array.

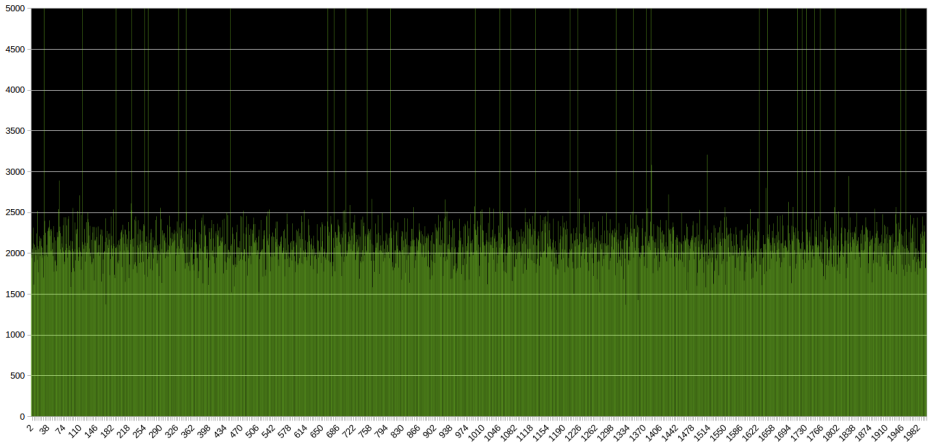
Con tale sistema possiamo calcolare differenti coperture semplicemente cambiando il modo in cui viene popolata *tableStart*. Calcolando la *physical coverage* veranno inserite solo le mate-pair con TLEN maggiore di zero così da considerare l'intera mate-pairs una volta per coppia. Per la *sequence coverage* invece vengono inserite entrambe le estremità dando come indice la posizione iniziale e come fine la lunghezza della sequenza sequenziata (SEQ). Infine, per le traccie con le insert filtrate tramite deviazione standard, vengono inserite solo le mate-pair che rispettano la lunghezza determinata dall'utente.

Insert Lenght

Insert Lenght rappresenta il numero di basi presenti in una mate pairs dall'estremo sinistro a quello destro mappate nel genoma. Come descritto in precedenza tale informazione è reperibile dal file SAM sotto la voce TLEN.

Per estrarre le insert length, è stato creato un modulo in cui per ogni mate pair presente nel file SAM, venga estratto il campo TLEN. Dato che le reads sono doppie verranno considerate solo le linee in cui TLEN è positivo. La funzione restituisce il file *insert_length.csv* contenente la lista delle insert, quest'ultima può essere usata da editor per fogli elettronici per produrre grafici.

Nell' immagine seguente viene visualizzato il grafico contenente l'insert lenght delle prime duemila mate pairs. Si noti che le insert variano per la maggior parte dalle 2000 a 2500 basi. È facile notare come siano presenti delle anomalie osservando reads che hanno lunghezza maggiore alle diecimila basi.



Media e Deviazione Standard

Prima di procedere al calcolo della media e della deviazione standard è necessario togliere i valori fuori range visti nel grafico precedente.

Verrà usata la mediana della lista dei valori del file *insert_lenght.csv* come valore di riferimento.
Verrà ordinata la lista in ordine crescente e poi prelevato il valore posizionato al metà. Vengono quindi cancellati gli estremi della lista che sono superiori o inferiori in rapporto con la mediana.

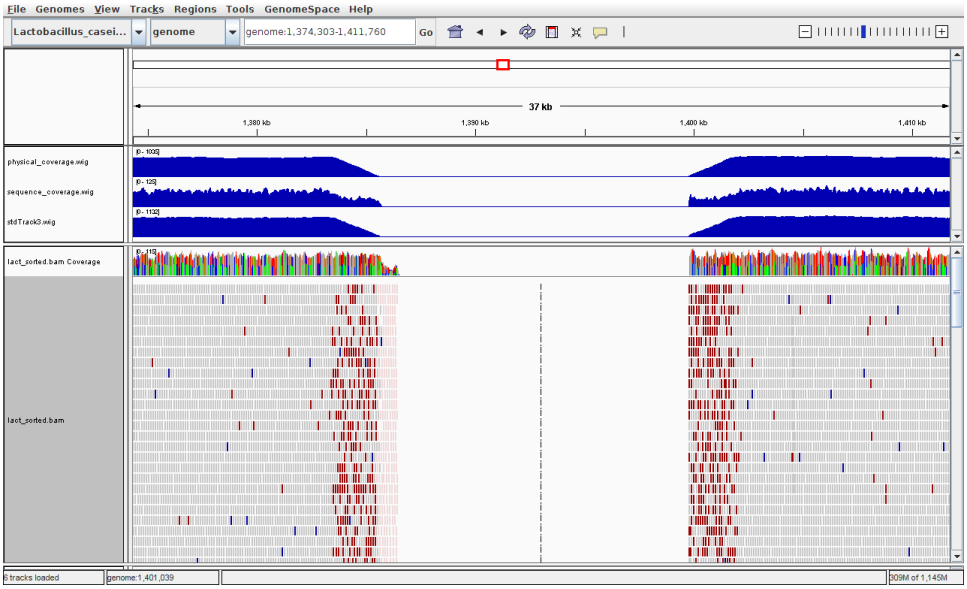
Alle fine viene calcolata la media e la deviazione standard su dati affidabili. Nell'immagine seguente viene visualizzato uno screenshot con il risultato dell'elaborato.

```
Mean= 2101.1765
Standard Deviations= 201.84079
Number inserts delete: 22377
```

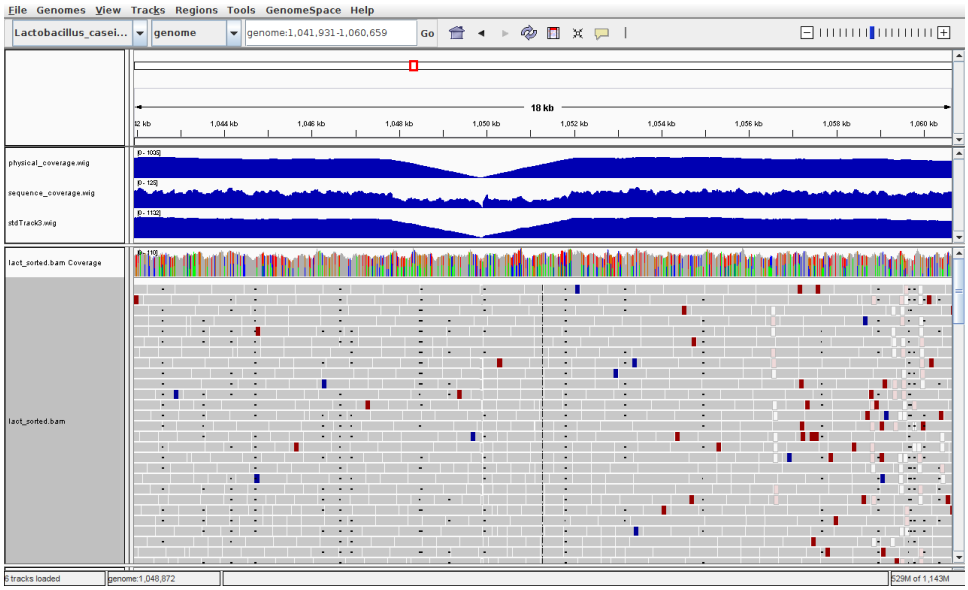
Visualizzazione tracce IGV

Presentiamo ora i risultati ottenuti dal calcolo della copertura. Carichiamo le tracce prodotte in IGV: *physical_coverage.wig*, *sequence_coverage*, *trackstd3.wig* che rappresentano la physical coverage, la sequence coverage e la traccia con deviazione standard pari a tre ed infine la mate-pairs mappate.

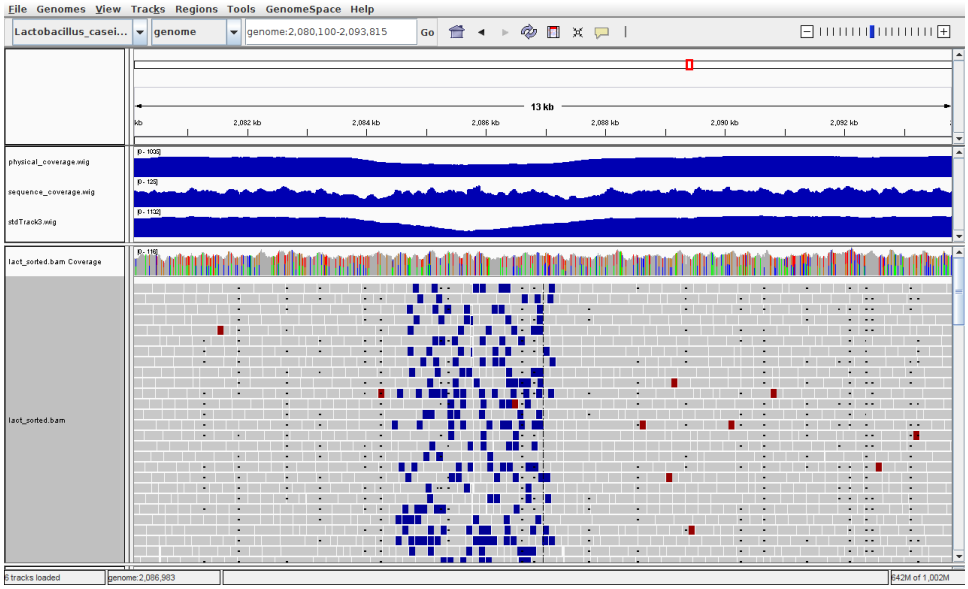
Grazie physical e sequence coverage possiamo trovare facilmente diverse variazione strutturali. In questa immagine per esempio c'è una *delezione:*



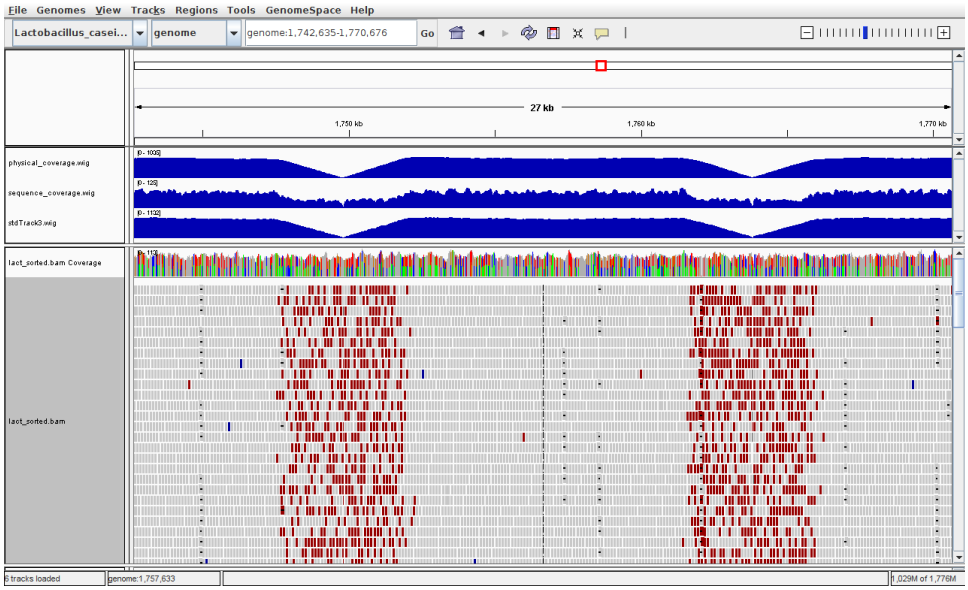
un inserzione lunga:



un inserzione corta:



ed infine un inversione



All’interno del genoma sono presenti altre variazioni strutturali per la maggior parte si trattano di piccole inserzioni.

Conclusioni

Lo sviluppo di tale progetto mi ha aiutato a capire le enormi potenzialità dell’informatica nel mondo della biologia. In un contesto in cui l’informazione è racchiusa in stringhe di milioni di carratteri risulta

impossibile per l’essere umano ricavare dati affidabili ed in tempi utili senza l’ausilio di algoritmi informatici.

Per il progetto è stato scelto di usare Scala. Il motivo è dato dal tentativo di imparare un nuovo linguaggio di programmazione in un contesto in cui il livello di prestazioni e affidabilità richieste è alto. Essendo un linguaggio moderno Scala offre soluzioni semplici ed eleganti a problemi comuni che in altri linguaggi avrebbero richiesto la stesura di più codice.

L’idea per calcolare la coverage è stata facilmente implementata grazie a questa caratteristica. È bastato infatti sviluppare una funzione per ogni metodo di copertura ed in base alla scelta dell’utente tale funzione veniva passata come parametro ad una funzione che scorreva il genoma.
Con il l’algoritmo presentato il calcolo della physcal coverage impiega un tempo inferiore ai metodi classici. Da aggiungere però che tale guadagno è riscontrabile nel calcolo di coperture dove le insert esaminate sono ampie in quanto il numero di operazioni è proporzionale al numero di mate pairs contenute in *tableEnd e tableStart*.

In appendice sono disponibili le istruzioni su come avviare il programma ed il codice sorgente.

Appendice 1

Di seguito verranno pubblicate gli screen shot su come eseguire le funzioni del programma. I comandi in verde sono quelli inserite dall'utente.

Calcolare insert length

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Welcome to genoma_AnaLyzer

Type a letter and press enter

       a    to get inserts length (inserts_length.csv)
       b    to calculate the mean and standard deviation of the inserts
       c    to get track with insert with a length exceeding N standard deviations above or below mean (stdTrack.wig)
       d    to get physical coverage track(physical coverage.wgi)
       e    to get sequence coverage track(sequence coverage)

       x    to TERMINATE

a
type a SAM file name (include extension) to analyze
alignment.sam
process complete
```

Calcolare media e deviazione standard

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Welcome to genoma_AnaLyzer

Type a letter and press enter

       a    to get inserts length (inserts_length.csv)
       b    to calculate the mean and standard deviation of the inserts
       c    to get track with insert with a length exceeding N standard deviations above or below mean (stdTrack.wig)
       d    to get physical coverage track(physical coverage.wgi)
       e    to get sequence coverage track(sequence coverage)

       x    to TERMINATE

b
type a CSV file name (include extension) to analyze
insert_length.csv

       Mean= 2101.1765
       Standard Deviations= 201.84079
       Number inserts delete: 22377

process complete
```

Creare physical coverage

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Welcome to genoma_AnaLyzer

Type a letter and press enter

       a    to get inserts length (inserts_length.csv)
       b    to calculate the mean and standard deviation of the inserts
       c    to get track with insert with a length exceeding N standard deviations above or below mean (stdTrack.wig)
       d    to get physical coverage track(physical coverage.wgi)
       e    to get sequence coverage track(sequence coverage)

       x    to TERMINATE

d
type a SAM file name (include extension) to analyze
alignment.sam
process complete
```

Creare coverage paramettrizzata dal deviazione std

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Welcome to genoma_AnaLyzer

Type a letter and press enter

       a    to get inserts length (inserts_length.csv)
       b    to calculate the mean and standard deviation of the inserts
       c    to get track with insert with a length exceeding N standard deviations above or below mean (stdTrack.wig)
       d    to get physical coverage track(physical coverage.wgi)
       e    to get sequence coverage track(sequence coverage)

       x    to TERMINATE

c
insert n
3
type:
   p      to insert parameter manually (fast)
   a      to calculate automatically (slow)
p
insert mean
2101.1765
insert standard deviation
201.84079
type a SAM file name (include extension) to analyze
alignment.sam
process complete
```

Appendice 2

Di seguito verranno elencati i vari sorgenti commentati.

genoma_analyzer. Scala

Tale classe ha lo scopo di interfacciare l'utente con le funzionalità del software. Sono presenti funzioni per creare un'interfaccia a riga di comando, chiamare funzioni disponibili, richiedere elaborati e stampare il risultato.

```
import java.io.FileNotFoundException

object genoma_analyzer {
  private var live: Boolean = true

//function which request a file sam name and it returns an iterable rappresentation
  private  def request_file_sam(): samFile = {
    println("type a SAM file name (include extension) to analyze")
    new samFile(io.StdIn.readLine())
  }

//function which request a file csv name and it returns an iterable rappresentation
  private def request_file_csv(): csvFile = {
    println("type a CSV file name (include extension) to analyze")
    new csvFile(io.StdIn.readLine())
  }

//function which call and print mean and standard deviation funcitons
  private def printMeanSTD(): Unit = {
    val tuple=StatisticsFunctionsContainer.getMeandStD(request_file_csv())
    println("\n      Mean= " + tuple._1.toFloat +"\n"+"      Standard Deviations= " + tuple._2.toFloat + "\n" + "
Number inserts delete: " + tuple._3+"\n")
  }

//function  which get parameter to create the track with dev_std parameter
  private def StdTrackParameter() = {
    println("insert n")
    val n=io.StdIn.readInt()
    println("type: \n   p     to insert parameter manually (fast)\n   a     to calculate automatically (slow)")
    val select=io.StdIn.readLine()
    if(select=="p"){
      println("insert mean")
      val mean=io.StdIn.readDouble()
      println("insert standard deviation")
      val std=io.StdIn.readDouble()
      coverageCalculator.getStdTrack(request_file_sam(),n,mean,std)
    }
    else{
      coverageCalculator.getStdTrack(request_file_sam(),n)
    }
  }

//fuction which detect and execute  user's request
  private def matchTest(request: Char): Any = request match {
    case 'a' => { StatisticsFunctionsContainer.getInsertsTrack(request_file_sam());println("process complete")}
    case 'b' => {printMeanSTD();println("process complete")}
    case 'c' => {StdTrackParameter();println("process complete")}
    case 'd' => {coverageCalculator.getPhysicalCoverage(request_file_sam());println("process complete")}
    case 'e' => {coverageCalculator.getSequenceCoverage(request_file_sam());println("process complete")}
    case 'x' => live=false
    case _ => println("\n >>>  wrong select  <<< \n")
  }

//function which print commands list
  def viewCommand(): Unit = {
    println("Type a letter and press enter\n")
    println("      a    to get inserts length (inserts_length.csv)")
    println("      b    to calculate the mean and standard deviation of the inserts")
    println("      c    to get track with insert with a length exceeding N standard deviations above or below mean
(stdTrack.wig)")
    println("      d    to get physical coverage track(physical coverage.wgi)")
```

```
    println("      e    to get sequence coverage track(sequence coverage)"+ "\n")
    println("      x    to TERMINATE\n")
  }


  def main(args: Array[String]): Unit = {
    println("Welcome to genoma_AnaLyzer\n")
    while (live) {
      viewCommand()
      try{
        matchTest(io.StdIn.readChar())
      }
      catch {
        case e : FileNotFoundException => println("\n >>>  wrong file, repeat  <<< \n")
      }
    }
  }
}
```

StatisticsFuncionContainer.Scala

Tale oggetto contiene le funzioni per estrapolare le insert length, calcolare la media e deviazione standard. Le restanti funzioni servono da supporto per effettuare le operazioni principali.

```
import java.io.{BufferedWriter, FileOutputStream, OutputStreamWriter}
import scala.collection.mutable.ListBuffer
import scala.math.{pow, sqrt}

object StatisticsFunctionsContainer {

//fuction which return a sorted list of number contains in a csv file
  def sortList(csv : csvFile) : ListBuffer[Int] = {
    val list : ListBuffer[Int] = new ListBuffer[Int]()
    for(line <- csv.iterator)
      list += line.toInt
    list.sorted
  }

//fuction wich take a order list and returns an order list without elements out of range
  def cleanData(list: ListBuffer[Int]) : Unit = {
    //calc median and rage of data
    val mediana: Int = list.apply(((list.size)/2))
    val downLimit : Double = mediana/2
    val upLimit : Double = mediana * 1.5
    //find index
    var downIndex = list.indexWhere(n => n>downLimit)
    var upIndex = list.indexWhere(n => n>upLimit)
    //delete unuseful elements
    list.remove(upIndex,list.size-upIndex)
    list.remove(0,downIndex)
  }

//fuction which calculate mean from a list
  private def calc_mean (list: ListBuffer[Int]) : Double = {
    var mean : Double = 0
    for(n <-list)
      mean += n
    mean / list.size
  }

//fuction which calculate standard deviation from a list
  private def calc_devStd(list: ListBuffer[Int],mean : Double) : Double = {
    var x : Double = 0
    for(n <-list)
      x += pow((n-mean),2)
    sqrt(x/list.size)
  }

//fuction which returns  mean, std_dev and number of element out of range from a csv file
  def getMeandStD(csv : csvFile) : Tuple3[Double,Double,Int] = {
    var order_list=sortList(csv)
    val realSize= order_list.size
```

```
      cleanData(order_list)
      val elemeDelete= realSize-order_list.size
      val mean=calc_mean(order_list)
      val std=calc_devStd(order_list,mean)
      Tuple3(mean,std,elemeDelete)
   }

//fuction which returns  mean, std_dev and number of element out of range from a sam file
   def getMeandStD(sam : samFile) : Tuple3[Double,Double,Int] = {
      getInsertsTrack(sam)
      var order_list=sortList(new csvFile("insert_length.csv"))
      val realSize= order_list.size
      cleanData(order_list)
      val elemeDelete= realSize-order_list.size
      val mean=calc_mean(order_list)
      val std=calc_devStd(order_list,mean)
      Tuple3(mean,std,elemeDelete)
   }

//fuction which returns a list of insert length from a sam file
   def getInsertsTrack(sam: samFile) : Unit = {
      val buffer_writer : BufferedWriter= new BufferedWriter(new OutputStreamWriter(new
FileOutputStream("insert_length.csv"),"UTF-8"))
      var tlen : String = ""
      try
      {
         for(line <- sam.iterator.dropWhile(x => x.charAt(0)=='@')){
            tlen=line.split("\\s")(8)
            if(tlen.toInt>0)
               buffer_writer.write(tlen + "\n")
         }
      }
      finally {
         buffer_writer.flush()
         buffer_writer.close()
      }
   }
}
```

CoverageCalculator.Scala

In questo modulo sono esposte le funzioni per calcolare la coverage.

```
import java.io.{BufferedWriter, FileOutputStream, OutputStreamWriter}

object coverageCalculator {

   private var weight : Int =0
   private val table : TableStart= new TableStart()
   private val end_read : TableEnd= new TableEnd()

//fuction which insert a insert in tableStart to calculate physical coverage. Receive in input: sam file line and array of
parameter wich can be empty
   private def tablePC(array: Array[String], para : Array[Double]) : Unit = {
      if (((array(1).toInt & 3) == 3) && (array(8).toInt > 0))
         table.add(array(3).toInt, array(7).toInt)
   }

//fuction which insert a insert in tableStart to calculate sequencel coverage.
   private def tableSC(array: Array[String], para : Array[Double]) : Unit = {
      if (((array(1).toInt & 3) == 3)) {
         if(array(8).toInt > 0)
            table.add(array(3).toInt, array(3).toInt + array(9).length)
         else
            table.add(array(3).toInt-array(9).length, array(3).toInt)
      }
   }

//fuction which insert a insert in tableStart to calculate sequencel coverage
   private def tableSTD(array: Array[String], para : Array[Double]) : Unit = {
      if (((array(1).toInt & 3) == 3) && (array(8).toInt > 0) && array(8).toInt<para(0) && array(8).toInt>para(1))
         table.add(array(3).toInt, array(7).toInt)
```

```
  }

//fuction which create tableStart by using a samfile and a fuction which filter the samfile line
  private def createMPTable(sam:samFile, f:(Array[String],Array[Double]) => Unit, para: Array[Double]
=Array[Double]()) : Unit = {
    for (line <- sam.iterator.dropWhile(x => x.charAt(0) == '@')) {
      val array = line.split("\\s")
      f(array,para)
    }
  }

//fuction which calc the coverage
  private def calculateCoverage(genome : Array[Int]) : Unit = {
    for(index <- 0 until genome.size){
      startController(index)
      genome(index)=weight
      endController(index)
    }
  }

//fuction wich update the weight
  private def startController(index : Int) : Unit = {
    if(table.find(index)){
      var list = table.getEnds(index)
      for(end <- list){
        end_read.add(end)
        weight += 1
      }
    }
  }

//fuction wich update the weight
  private def endController(index : Int) : Unit = {
    if(end_read.lookfor(index)){
      var counter : Int = end_read.getCounter(index)
      weight= weight - counter
      end_read.remove(index)
    }
  }

//fuction wich create a wig tracj
  private def createWigTrack(genome : Array[Int], nameFile: String) : Unit = {
    val buffer_writer : BufferedWriter= new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(nameFile),"UTF-8"))
    buffer_writer.write("fixedStep chrom=genome start=1 step=1 span=1" + "\n")
    try
    {
      for(i <- 0 until genome.size){
        buffer_writer.write(genome(i).toString + "\n")
      }
    }
    finally {
      buffer_writer.flush()
      buffer_writer.close()
    }
  }

//fuction wich cleare the resourced
  private def endProccess() : Unit = {
    table.clear()
    end_read.clear()
    weight=0
  }

//fuction wich create a wig file with physical coverage from a samfile
  def getPhysicalCoverage(sam:samFile) : Unit = {
    val genome : Array[Int] = new Array[Int](sam.genoma_lenght)
    createMPTable(sam,tablePC)
    calculateCoverage(genome)
    createWigTrack(genome,"physical_coverage.wig")
    endProccess()
  }
```

```
//fuction wich create a wig file with sequence coverage from a samfile
   def getSequenceCoverage(sam:samFile) : Unit = {
      val genome : Array[Int] = new Array[Int](sam.genoma_lenght)
      createMPTable(sam,tableSC)
      calculateCoverage(genome)
      createWigTrack(genome,"sequence_coverage.wig")
   }

//fuction wich create a wig file with coverage near the std_dev from a sam file
   def getStdTrack(sam : samFile, n: Int) : Unit = {
      val genome : Array[Int] = new Array[Int](sam.genoma_lenght)
      val tuple=StatisticsFunctionsContainer.getMeandStD(sam)
      val aboveMean : Double=tuple._1+(n*tuple._2)
      val belowMean : Double=tuple._1-(n*tuple._2)
      createMPTable(sam,tableSTD,Array(aboveMean,belowMean))
      calculateCoverage(genome)
      createWigTrack(genome,"stdTrack"+n+".wig")
      endProccess()
   }

//fuction wich create a wig file with coverage near the std_dev from a sam file and parameters
   def getStdTrack(sam : samFile, n: Int, mean : Double, std:Double) : Unit = {
      val genome : Array[Int] = new Array[Int](sam.genoma_lenght)
      val aboveMean : Double=mean+(n*std)
      val belowMean : Double=mean.-(n*std)
      createMPTable(sam,tableSTD,Array(aboveMean,belowMean))
      calculateCoverage(genome)
      createWigTrack(genome,"stdTrack"+n+".wig")
      endProccess()
   }
}
```

tableStart.scala

Classe per gestire gli indici iniziali delle insert usata nel calcolo della coverage

```
import scala.collection.mutable.{HashMap,ListBuffer}

class TableStart {
   private val table : HashMap[Int,ListBuffer[Int]] =  new HashMap[Int,ListBuffer[Int]]()
   def add(start: Int, end :Int) : Unit = {
      if(table.contains(start)){
         table(start) += end
      }
      else{
         var endList=new ListBuffer[Int]
         endList += end
         table += start -> endList
      }
   }
   def find(index : Int) : Boolean = {
      table.contains(index)
   }
   def getEnds(star: Int) : ListBuffer[Int] = {
      table(star)
   }
   def print() : Unit = {
      table.foreach(println)
   }
   def clear() : Unit = {
      table.clear()
   }
}
```

tableEnd.scala

Classe per gestire gli indici finali delle insert usata nel calcolo della coverage

```
import scala.collection.mutable.HashMap
class TableEnd {
```

```
  private val tree : HashMap[Int,Int] = new HashMap[Int,Int] ()
  def add(index_end: Int) : Unit ={
    if(tree.contains(index_end)){
      var counter : Int = tree(index_end)
      tree.remove(index_end)
      tree += index_end -> (counter+1)
    }
    else{
      tree += index_end -> 1
    }
  }
  def remove(index: Int) : Unit = {
    tree.remove(index)
  }
  def lookfor(ele : Int) : Boolean = {
    tree.contains(ele)
  }
  def getCounter(index_end: Int) : Int = {
    tree(index_end)
  }
  def clear() : Unit = {
    tree.clear()
  }
}
```

samFile.scala

Per gestire i file sam in input

```
import java.io.FileNotFoundException

class samFile (val name_file : String) extends Iterable[String]{
  val genoma_name = genomaName()
  val genoma_lenght : Int= genomaLength()
  private def genomaLength () : Int = {
    val it=iterator
    if(it.hasNext){
      val header : String = it.next()
      if(header.startsWith("@") && header.contains("LN"))
            return header.split("LN:")(1).toInt
    }
    return 0
  }
  private def genomaName() : String = {
    val it=iterator
    if(it.hasNext){
      val header : String = it.next()
      if(header.startsWith("@") && header.contains("SN"))
        return header.split("SN:")(1)
    }
    return "no_name"
  }
  override def iterator: Iterator[String] = {
    io.Source.fromFile(name_file).getLines()
  }
}
```

csv.File.scala

Per gestire i file csv in input

```
class csvFile(val name_file : String) extends Iterable[String] {
  override def iterator: Iterator[String] = {
    io.Source.fromFile(name_file).getLines()
  }
}
```