International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume1, Issue 3 (June '2015)

# ROLE OF JAVA CLASSES TO SOLVE DNA SEQUENCING AND MOTIF FINDING PROBLEMS IN BIO INFORMATICS

**K.Senthil Kumaran,**
Lecturer,
Department of Computer Science,
Hindustan College of Arts and Science, Chennai, India

**A.Vijaya Kumar,**
Assistant Professor,
Department of Computer Science,
Hindustan College of Arts and Science, Chennai, India

**P.Jayaseelan,**
Assistant Professor,
Department of Computer Science,
Hindustan College of Arts and Science, Chennai, India

**G.Mohendran,**
Assistant Professor,
Dept of Computer Science,
T.J Institute of Technology,,Chennai,India

**Abstract:** In java the classes Pattern and Matcher are useful for solving DNA sequencing and Motif Finding Problems. Pattern is a compiled representation of a regular expression. Matcher is an engine that performs match operations on a character sequence like DNA sequence by interpreting a Pattern. The Bioinformatics problems like DNA sequencing, DNA Mapping, Predicting Genes, Comparing Genes and Genome Rearrangements need enormous support from programming language constructs for searching billions of DNA chemical nucleotide bases (A,T,G,C). In this regard, System defined Java classes from java.util.regex java package will play an important role. Their capabilities will be analyzed in this research paper by doing dry lab experiments.

***Keyword:Java,Pattern,Matcher,DNA,Sequencing,Bioinformatics***

## I . INTRODUCTION

Java supports variety of Classes to implement search operations in DNA sequences. From Java™ 2 Platform Std. Ed. v1.6, we can write industry standard, robust, platform independent java programs to solve DNA sequencing problems. The two classes Pattern and Matcher are very useful for implementing searching operation in a string of ATGC DNA sequence. We know that Java classes are classified into packages as genes are packed in to chromosomes. Here we will concentrate on the above mentioned Java classes and their member variables (Fields) , Member functions. The package in which these classes Pattern and Matcher classes are stored is java.util.regex java package.

## II . PATTERN

public final class **Pattern**
extends Object
implements Serializable

This class is a compiled representation of a regular expression. A regular expression is specified as a string of DNA chemical nucleotide bases (A,T,G,C). It must first be compiled into an instance of this class. The resulting pattern can then be used to create a Matcher object that can match arbitrary ATGC DNA sequence against the regular expression. All of the state involved in performing a match resides in the matcher; so many matchers can share the same pattern. A typical invocation sequence is thus

```
Pattern p = Pattern.compile("A*G");

Matcher m = p.matcher("AAAAAAAG");

boolean b = m.matches();
```

In the above example the Boolean variable b will receive a value true if and only if , the given input search sequence "AAAAAAAG" is confined into the compiled regular expression "A*G". The regular expression "A*G" denotes that the Character A will be repeated zero or more time and the G will come only one time. Since the given input string is "AAAAAAAG", so the Boolean variable b will get a value true. The Boolean variable b will receive a value false if and only if, the given input search sequence is not confined into the given regular expression. A matches method is defined by this class as a convenience when a regular expression is used just once. This method compiles an expression and matches an input sequence against it in a single invocation. The statement

```
boolean b = Pattern.matches("A*G", "AAAAAAAG");
```

is equivalent to the three statements above, though for repeated matches it is less efficient since it does not allow the compiled pattern to be reused. In this example, the first

 

parameter for matches method is the regular expression and the second parameter is the given input search string.

## III .MATCHER

public final class **Matcher**
extends Object

This Class is an engine that performs match operations on a ATGC DNA character sequence by interpreting a Pattern object. A matcher is created from a pattern by invoking the pattern's matcher method. Once it is created, a matcher can be used to perform three different kinds of match operations:

- The matches( ) method attempts to match the given ATGC DNA character sequence against the pattern.
- The lookingAt( ) method attempts to match the given ATGC DNA character sequence, starting at the beginning, against the pattern.
- The find( ) method scans the given ATGC DNA character sequence looking for the next subsequence that matches the pattern.

Each of these methods returns a Boolean value indicating success[true] or failure[false]. More information about a successful match can be obtained by enquiring the state of the matcher. This class also defines methods for replacing matched DNA subsequences with new strings whose contents can, if desired, be computed from the match result. The appendReplacement( ) and appendTail( ) methods can be used in repeatedly in order to collect the result into an existing string buffer, or the more convenient replaceAll( ) method can be used to create a string in which every matching subsequence in the DNA input sequence is replaced. The explicit state of a matcher includes the start and end indices of the most recent successful match. It also includes the start and end indices of the DNA input subsequence captured by each capturing group in the pattern as well as a total count of such subsequences. As a convenience, methods are also provided for returning these captured subsequences in string form. The explicit state of a matcher is initially undefined. So attempting to enquiry any part of it before a successful match will cause an IllegalStateException to be thrown. The explicit state of a matcher is recomputed by every match operation. The implicit state of a matcher includes the input character sequence as well as the append position, which is initially zero and is updated by the appendReplacement( ) method. A matcher may be reset explicitly by invoking its reset() method or, if a new input sequence is desired, its reset(CharSequence) method. Resetting a matcher discards its explicit state information and sets the append position to zero. Instances of this class are not safe for use by multiple concurrent threads.

## IV . DRY LAB EXPERIMENT 1

The following Java program is a simple example for initiating to write DNA sequencing programs to solve Motif Finding Problems. We can use any kind of algorithms like Exhaustive Search Method, Randomized algorithms and Greedy Methods so on. But we need a programming language support for initializing the regular expression and searching DNA sequence. In Java we can create an object for the regular expression using the class java.util.regex.Pattern. After creating an object of Pattern class, We have to create a Matcher class object from a pattern by invoking the pattern's matcher method. In the Next step the matches method attempts to match the given ATGC DNA character sequence against the pattern.

```
package javaapplication9;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
//  @author K.Senthil Kumaran
public class Main {

   public static void main(String[] args) {
      String serch    = "AAAAAAAG";
      String regularexp = "A*G";
      Pattern pattern = Pattern.compile(regularexp);
      Matcher matcher = pattern.matcher(serch);
      boolean matches = matcher.matches();
      System.out.println("Is   "+serch+"   matching   "+"with
"+regularexp+" ?");
      if(matches==true)
      System.out.println("Result : Yes, It is matching");
      else
       System.out.println("Result : No, It is not matching");
   }
}
```

Result:

```
run:
Is AAAAAAAG matching with A*G ?
Result : Yes, It is matching
BUILD SUCCESSFUL (total time: 0 seconds)
```

In this program the search string is stored in a String variable serch=”AAAAAAA”. The regular expression is stored in a String variable regularexp = "A*G". The Pattern class object is created with the regularexp as a parameter using a pattern class member function compile() as follows Pattern pattern = Pattern.compile(regularexp). Pattern class supports matcher( ) function ,using this Matcher object will be created. In the matcher( ) function searching string serch is used as a parameter. It is expressed in java statement as below. Matcher matcher = pattern.matcher(serch).

Finally using the matches( ) function available in the Matcher class , We can find whether the search string is confined to the given regular expression as follows. boolean matches = matcher.matches( ). This method matches( ) will return the Boolean value either true or false based on the successes or failure of the search operation respectively. In the above example the result is true because the searching sequence "AAAAAAA" is matching with the regular expression = "A*G".

**V . DRY LAB EXPERIMENT 2**

The Matcher class also supports lookingAt() method and it is working like matches() method with few differences. ThelookingAt() method is used to matches the regular expression against the beginning of the text, but matches() function matches the regular expression against the entire text. In other way , We can say if the regular expression matches the beginning of a text but not the whole text than lookingAt() method will return true, whereas matches() will return false. The following program is the example for applying the lookingAt() method from Matcher class.

```
package javaapplication10;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
//  @author K.Senthil Kumaran
 public class Main {
   public static void main(String[] args) {
          String serch    = "GACAACAAT";
      String regularexp = "GA*";
      Pattern pattern = Pattern.compile(regularexp);
      Matcher matcher = pattern.matcher(serch);
      //using matches method( )
     boolean matches = matcher.matches();
      System.out.println("Is "+serch+" matching "+"fully with
"+regularexp+" ?[using matches method]");
      if(matches==true)
      System.out.println("Result : Yes, It is matching");
      else
       System.out.println("Result : No, It is not matching");
      // using lookingAt method( )
      boolean look=matcher.lookingAt();
       System.out.println("Is "+serch+" matching "+"begining
only  with "+regularexp+" ?[using lookingAt method]");
      if(look==true)
      System.out.println("Result : Yes, It is matching");
      else
       System.out.println("Result : No, It is not matching");
   }
}
```

Result:
run:
Is GACAACAAT matching fully with GA* ?[using matches method]
Result : No, It is not matching
Is GACAACAAT matching begining only with GA* ?[using lookingAt method]
Result : Yes, It is matching
BUILD SUCCESSFUL (total time: 0 seconds)

In the above example the searching string "GACAACAAT" is only matching with the beginning of the regular expression "GA*" , but not completely. So lookingAt( ) method will return true, whereas matches( ) will return false. This lookingAt( ) is very useful while comparing a particular human gene against other reptiles and mammals genes.

**V . DRY LAB EXPERIMENT 3**

In the below Java coding, We are trying to utilize the Matcher class methods like find( ),start( ),and end( ). The find( ) method attempts to find the next subsequence of the input sequence that matches the pattern. It returns true, if and only if, a subsequence of the input sequence matches this matcher's pattern. The start() method returns the start index of the previous match. The end() method returns the offset after the last character matched.

```
package javaapplication11;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
//  @author K.Senthil Kumaran
 public class Main {
    public static void main(String[] args) {
      String serch    = "GACAACGAATGAAGTGA";
      String regularexp = "GA";
      Pattern pattern = Pattern.compile(regularexp);
      Matcher matcher = pattern.matcher(serch);
       int ct = 0;
      while(matcher.find()) {
         ct++;
         System.out.println("found Occurrence: " + ct + " : "
               + matcher.start() + " - " + matcher.end());
        }
}
}
```

Result:
run:
found Occurrence: 1 : 0 - 2
found Occurrence: 2 : 6 - 8
found Occurrence: 3 : 10 - 12
found Occurrence: 4 : 15 - 17
BUILD SUCCESSFUL (total time: 0 seconds)

In the above example "GACAACGAATGAAGTGA" is a searching sequence that is matched against the regular expression "GA". The find( )method is repeatedly executed in a while loop as long as it is returning a Boolean value true. First time when find( ) method is executed it returns the value true and the first occurrence of the starting index of GA is 0 that is returned by start( )method. The end() method returns

the offset after the last character matched. That is 2. In the second time the find( ) method is successful so it returns true and the second occurrence of the input subsequence is at index 6 and 7. So start( )method returns 6 and end( ) method returns 8. This process will be repeated two more time and we can identify that there are totally 4 occurrence of the subsequence "GA" in the given searching string. This kind of intermediate matching operations are very useful in Motif finding problems. Fruit Flies have a small set of immunity genes that are normally dormant in the fly genome. These genes are switched on when the organism is infected. When these genes are switched on, they produce proteins that destroy the pathogen, by the way curing the infection. The fruit flies are infected with a bacterium by an experiment. Now we measure which genes are switched on as an immune response. Many immunity genes in the fruit fly genome have strings that are reminiscent of AATCTCGGGGATTTCC, located upstream of the genes start. These short strings are called as NF-kB binding sites. Motif finding is the problem of discovering such motifs without any prior knowledge of how motifs look.

## VI. APPLICATION

1. DNA sequencing
2. DNA Mapping
3. Predicting Genes
4. Comparing Genes
5. Genome Rearrangements and so on

## VII. FUTURE SCOPE

This research paper is just a end of the beginning of designing and developing a online software packages like DDBJ(DNA Data bank of Japan), NCBI(National Center for Biotechnology Information) , EMBL(European Molecular Biology laboratory), BLAST and FASTA.

## VIII.CONCLUSION

This research paper is the part of the DOE-NIH U.S. Human Genome Project formally began October 1, 1990. The Human Genome Project (HGP) refers to the international 13-year effort, formally begun in October 1990 and completed in 2003, to discover all the estimated 20,000-25,000 human genes and make them accessible for further biological study. Another project goal was to determine the complete sequence of the 3 billion DNA subunits (bases in the human genome). As part of the HGP, parallel studies were carried out on selected model organisms such as the bacterium *E. coli* and the mouse to help develop the technology and interpret human gene function.

## REFERENCES

[1]. T.Brown . Genomes. John Wiley and Sons, New York 2002.
[2]. D.E.Knuth. The Art of Computer Programming, Addision-Wesley,1998.
[3]. A.Aho, J.Hopcroft, and J.Ullman. Data Structures and Algorithms. Addision- Wesley,Boston,1983.
[4]. O.Gotoh. An improved algorithm for matching biological sequences. Journal of Molecular Biology.
[5]. W.Gates and C.Papadimitriou. Bounds for sorting by prefix reversals. Discrete Mathematics,27:45-57,1979.
[6]. Neil C. Jones and Pavel A.Pevzner An Introduction to Bioinformatics Algorithms , Ane Books New Delhi-110 002.
[7]. John E. Hopcroff and Jeffery D.Ullman Introduction To Automata Theory, Languages, And Computation, Narosa Publishing House New Delhi.
[8]. National Center for Biotechnology information website: www.ncbi.nih.gov/index.html
[9]. DNA Data bank of Japan website: www.ddbj.nig.ac.jp/Welcome-j.html
[10]. European Molecular Biology Laboratory website: www.embl.org

AUTHOR'S PROFILE

K.Senthil Kumaran M.Sc.M.Phil is working as a Lecturer in Department of Computer Science, Hindustan College of Arts and Science, Chennai, India. His research areas are Bioinformatics computing and Ethical Hacking. He is having 12+ years of teaching experience in various colleges in Tamilnadu.

A. Vijaya Kumar, M.C.A., M. Phil., M.Tech., working as a Asst. Professor in Department of Computer Science at Hindustan College of Arts and Science, Chennai, India. Having 13 Years of Teaching experiences. An Empathic Student Counselor and Trainer. A research scholar from Bharathiar University, Coimbatore.

P.Jayaseelan M.C.A.,NET., is working as a Assistant professor In Department of Computer Science, Hindustan College of Arts and Science, Chennai, India. His research areas are Soft Computing and Image Processing. He is having 5+ years of teaching experience in various colleges in Tamilnadu.

G.Mohendran, MCA,M.E., has 7 years of experience in Teaching and his research areas are MANET&VANET is currently working as Asst. Professor in T.J institute of Technology, Chennai, India.