/\*

\***Program Description**: A program to find the median string and motif from given a set of DNA sequences and the length of the motif using L-mer tree with bypassing.

\* this program reports 5 best median strings and their corresponding motif consensus strings and

positions. In this program a median string is identified based on the given set of input sequences and then the corresponding consensus string is identified.

\***Input:** a set of DNA sequences (arbitrary length each) and motif length (L-mer size); a prokaryotic DNA sequence file “HMP-617.fa”

\***Output:** best 5 median strings (with total\_distances), corresponding motif consensus strings/scores and motif positions (for each motif string)

\***Approach followed:**

\*1) Store all sequences from HMP-617.fa into a list of strings

\*2) starting from the root A, passing into nextvertex function each of those . If there is no match in each of the other sequences, find the most nearest match

\*(since mutation is allowed) and store the hamming distance.

\* 3) Sort the hamming distances list and find the string matches with least hamming distance and print the visited string.

\*4) That is identified as the median.

\*5)align all the median strings and count the number of bases(A,C,G,T). Identify the base with maximum count, that is the character in the motif (for that position)

\* 6) sum all maximum counts to get the consensus score

**SOURCE CODE**

**\*/**

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Arrays;

import java.util.regex.Pattern;

public class Motif {

String fileName = "HMP-617.fa";

try {

// FileReader reads file.

FileReader fileReader =   new FileReader(fileName);

String[] sequences = new String[617];

BufferedReader bufferedReader =  new BufferedReader(fileReader);

String line;

int n =0;

while(((line=bufferedReader.readLine())!= null) && (line.startsWith(">")))

{

String line2 = "";

line2 = line2+line;

sequences[n] = line2;

n++;

 }

 // Always close files.

bufferedReader2.close();

} catch (FileNotFoundException ex1)

{

System.out.println("Unable to open file '" + fileName + "'");

} catch (IOException ex) {

System.out.println("Error reading file '" + fileName + "'");

}

int[] position = new int[1000];

int[] hd = new int[1000];

String median;

String[] alpha = {"A","C","G","T"};

int hamming\_distance[] = new int[];

int d=0;

while(i>0){

if(i<l){

(s,i)=nextVertex(s[],i,6,4);

int optimumScore = allScore(s[],dna[],l) + (t-1) \* l;

if (optimumScore < bestScore)

(s[], i) = Bypass(s[], i, t, n-l+1);

else

(s[],i) = NextVertex(s[],i,t,n-l+1);

int score = allScore(s[], dna,l);

if (score>bestScore){

bestScore = score;

}     }}

Arrays.sort(hamming\_distance);

  hd[0]=hamming\_distance[0];

median=elements[0];

 }        }

public int ByPass(int[] a,int i,int L, int k){

for(int j=0; j<i; j++){

if(a[j] < k){

a[j] = a[j]+1;

return a[i];

a[j] = 0;

}

else

return a[j];

}

}

 }

public static int getHammDis(String sequence1, String sequence2){

  int distance =0;

if(sequence1 == null || sequence2==null)

System.exit(0);

sequence1 = sequence1.toUpperCase();

sequence2 = sequence2.toUpperCase();

if(sequence1.length() != sequence2.length())

{

return -1;

}

for(int i=0;i < sequence1.length();i++)

{

if(sequence1.charAt(i)!=sequence2.charAt(i))

distance++;

}

  return distance;

}

System.out.println("Median String:" + Median+ " (tot\_dist =" +hamming\_distance + " )");

}  
}

public static String getConsensus(int numSeq, int lengthSeq, int motifLength, int[]

consensusPosition, String[] SeqArray)

{

String ConsensusString="";

int i;

int x;

int aCount=0;

int tCount=0;

int gCount=0;

int cCount=0;

int currentPostion;

char currentChar='a';//default character

for (x=0; x<motifLength; x++)

{

aCount=0;//we need to reset all counts before moving

tCount=0;//to the next set of counts

gCount=0;

cCount=0;

for(i=0; i<numSeq;i++)//go with all sequences

{

if(SeqArray[i].charAt(consensusPosition[i]+x)=='a')

{aCount++;

if (aCount>Math.max(tCount, Math.max(gCount,cCount)))

{currentChar='a';}}

if(SeqArray[i].charAt(consensusPosition[i]+x)=='t')

{tCount++;

if (tCount>Math.max(aCount,Math.max(gCount,cCount)))

{currentChar='t';}

}

if(SeqArray[i].charAt(consensusPosition[i]+x)=='g')

{gCount++;

if (gCount>Math.max(tCount,Math.max(aCount,cCount)))s

{currentChar='g';}

}

if(SeqArray[i].charAt(consensusPosition[i]+x)=='c')

{cCount++;

if (cCount>Math.max(tCount,Math.max(gCount,aCount)))

{currentChar='c';}

}

}//Iterates with Sequences Loop Ends

ConsensusString +=currentChar;

}

return ConsensusString;

}

public static void FindMotifPositions(string medianValue, string SequenceValue, ref

List<KeyValuePair<string,int>> MotifPositions)

{

//Variables used in this method.

bool setValue = true;

string v1 = string.Empty;

int v2 = 0;

int k = 0;

int score = 0;

int substring\_position = 0;

//Loop through DNA sequence and get substring of sequence size "6" and find out hamming score.

for (k = 0; k <= (SequenceValue.Length - medianValue.Length); k++)

{

score = 0;

//Get substring of size of l(L=6)

string tempValue = SequenceValue.Substring(k, medianValue.Length);

//Convert median and substring into character array for comparing each character at same position.

char[] median = medianValue.ToArray();

char[] tmpvalue = tempValue.ToArray();

//Loop through character array to find score of mismatch.

for (int l = 0; l < medianValue.Length; l++)

{

if (median[l] != tmpvalue[l])

{

score = score + 1;

}}

int maxscore = score;

//Below written if statement will set minimum hamming score.

if (maxscore != substring\_position)

{

if (setValue)

{

substring\_position = maxscore;

setValue = false;

}

}

//We will check if the new hamming score value is less than the old hamming score value then set new value

//into variables so that that can be update into list.

if (maxscore < substring\_position)

{

substring\_position = maxscore;

v1 = tempValue;

v2 = k;

}

//If hamming score value is zero/same value then we use the below code set last string

as value to below

//variables.

if(maxscore==substring\_position)

{

v1 = tempValue;

v2 = k;

}

}

//Add motif strings and their corresponding score into list.

MotifPositions.Add(new KeyValuePair<string, int>(v1, v2));

System.out.println("Motif Consensus String:"+ v1 + "(Consensus\_Score = "

+ v2 + ")");

System.out.println("Motif Positions/String S=(S1....St):");

}

// This method is used to find out median string and its score. We will add new score to the

existing score if that median has already been placed into dictionary.

public static void TotalDistance(string medianValue, string SequenceValue, ref Dictionary<string, int> searchValue)

{

//Variables to be used in this method

bool setValue = true;

int k = 0

int score = 0;

int substring\_position = 0;

//Loop through DNA sequence till the last substring string has size "6" and find out hamming score.

for (k = 0; k <= (SequenceValue.Length - medianValue.Length); k++)

{

score = 0;

//Get substring of size of l(L=6)

string tempValue = SequenceValue.Substring(k, medianValue.Length);

//Convert median and substring into character array for comparing each character at same position.

char[] median = medianValue.ToArray();

char[] tmpvalue = tempValue.ToArray();

//Loop through character array to find score of mismatch.

for

(int l = 0; l < medianValue.Length; l++)

{

if (median[l] != tmpvalue[l])

{

score = score + 1;

System.out.print(substring\_position.get(l) + "(" + medianValue.get(i) + ")");

}

}

int maxscore = score;

//We will check if the new hamming score value is less than the old hamming score value then set new value

//into variables so that that can be update into list.

if (maxscore != substring\_position)

{

if (setValue)

{

substring\_position = maxscore;

setValue = false;

}

}

//get the score for each substring.

if (maxscore <= substring\_position)

{

substring\_position = maxscore;

setValue = false;

}

}

//We will check for duplicity and if that string is not present then we will add to that list.

if (searchValue.ContainsKey(medianValue))

{

searchValue[medianValue] = searchValue[medianValue] + substring\_position;

}

else

{

//Add substring from sequences and its score.

searchValue.Add(medianValue, substring\_position);

}

}

}

}

/\*

\***Compilation:**

\*1. Used notepad++ for writing the program code.

\*2. In next step, program should be saved in “.java” format.

\*3. Using command prompt in windows, program should be compiled using command line like javac Motif.java

\*4. After running above argument a class file with Motif.java will form this should now run with java Motif, then we get required output.

\***Output:** Submitted through mail.

\*/

291T BIOINFORMATICS COMPUTING

(Assignment 3)

NAME: YELLAPATI,SRUTHI

ID:109153960