DAA Assignment

MOTIF FINDING

The problem statement:

Motifs are a set of repeating patterns of nucleic acid in DNA that might significantly affect certain traits.

Given an array of strings representing DNA strands, Motifs are to be found using an edit-distance based algorithm.

Algorithm

Motif Search

```
L = length of the motif
D = maximum edit distance for relationship
Indel = weight for insert or delete operation in finding minimum edit distance
Subs = weight for substituting while finding minimum edit distance

Loop 'i' through all strand:
    Loop 'j' through each element of stand[i]:
        Create a substring of strand[i] of size L starting from index j
        Loop 'x' through all strands except for strand[i]:
            Boolean flag = false
            Loop 'y' through each element of strand[x] while flag is false:
                //(setting a lower and upper bound for length of string outside which relationship can be formed with 0 probability )
                Loop 'k' from lower limit to upper limit:
                //(This loop is for looping through all possible substrings of varied lengths of the strand starting from index y
                    Sample = substring of strand[x] from index y to y+k
                    Editdistance = minedit(motif,sample)
                    if (editdistance <= D):
                        Flag = true
                        Count++;
                        break;
if( count == 19):
    Print( motif )
```

Edit Distance

- Create a 2D array mindis[][] with dimensions as (length + 1) of the two strings for which edit distance is to be found (l1 for string1 && l2 for string2)
- For i in l1:
    For j in l2:
        If (i== 0)
            Mindis[i][j] = j
        Else if (j == 0)
            Mindis[i][j] = i
        Else

$$E[i,j] = \min \begin{cases} E[i-1,j] + indel \\ E[i,j-1] + indel \\ E[i-1,j-1] + \begin{cases} 0, if\ X[i] = Y[j] \\ sub, if\ X[i] \neq Y[j] \end{cases} \end{cases}$$

Analysis of Algorithm

- Minedit has a complexity of n2
- This is a brute force algorithm which runs through all possible substrings in each DNA strand.
- The algorithm runs x*y times for picking each substring
- For checking for relation, the algorithm runs for (x-1)*y*(z) where z is the number of substrings that can be formed starting from a single index.

The complexity of the algorithm would be (x*y) * (x-1)*y*z * n*n

Time complexity:- **O(n^7)**

This algorithm can be made to run faster in the DP part in the mini part.
An if statement can be added in the algorithm where
if(mini > dis)
The return mini
This way extra iterations can be prevented thereby reducing the runtime.

Better algorithm to solve this problem:

Generalized Suffix Tree

There is an approach where a Generalized Suffix Tree is built for solving this.
The complexity for building the generalised suffix tree is O(len(S1) + len(S2) + … + len(SN)) using an algorithm called Ukkonen algorithm.

This algorithm reduces search space and time but there are a few mismatches in this

Planted Motif Search

This algorithm divides all the strings into l-mers (substring of length l)