**Aditya Srivastava  
1910110034**

**DAA Assignment - 1**

**Analysis**

We will have to first take 20 random strings and then create a substring for each string of length L. Store substrings of each string in the ArrayList. Now we can say that if M is a substring and M’ is its neighbor then edit distance that is (M, M’) and its reflexive (M’, M) will give the same distance as well. The edit distance is transitive in nature as well. So, we just have to traverse through all the substrings M of the first string, get all the neighbors M’ for it, and print M and all the M’ if M finds a neighbor in all the strings.

Now let us see in detail how the reflexive and transitive properties of edit distance will be able to get all the substrings.

**Reflexive:**

Let’s say if we are checking a substring of 1st string and then we check for neighbors in the 4th string if one substring satisfies the condition of (M, M’) <= D then when we go to the 4th string and for that specific substring we check for neighbors then that same substring of 1st string will still satisfy the condition as neighbor.

**Transitive:**

If we get substrings from 2 strings then we get it from 3rd as well. Taking an example like assume that you find the neighbor in the 4th string and 5th string for M then if we choose that substring in the 4th string so as to find its neighbor then we can say for sure that you will find a neighbor in the 5th. We know it because:

Substring of 1st string has a neighbor in 4th string. ---(1)

Substring of 1st string has a neighbor in 5th string. ---(2)

Then from the transitive property,

Substring of 4th String (same as in (1)) has a neighbor in 5th string (that will be the same as in (2))

Forming from this ideology we are able to find all the substrings in a lot less time as we have to traverse through the first ArrayList find all its neighbors and if it satisfies the condition then just print all of them.

So, we get all substrings that are needed.

**Worst case time complexity:**

**N:** number of elements in 1st ArrayList  
**M:** 19 (size of neighbourSubstring.size()-1)  
**J:** Number of elements in the current ArrayList  
**Edit Distance:** O(ab), where a -> characters in 1st substring, b -> characters in 2nd substring  
**Time Complexity:** O (N.19.J. ab) -> O(NJab)

**Algorithm**

1. **public static void main (String args []) throws IOException**
   1. Initialize file pointer for the input file.
   2. For loop which runs 20 times and calls function to create a random string and after creating the string calls the function to create the substrings of the string of length L.
   3. Store it in an ArrayList. Then store that ArrayList in a ArrayList which holds the Array Lists of substrings of each string.
   4. After the whole loop completes, call function to print all the substrings that satisfy the conditions.
2. **private static void printAllNeighboursWEditDistance (ArrayList<ArrayList<String>> neighbourSubstring, int D, int indel, int sub) throws IOException**
   1. Initialize file pointer for output file.
   2. For loop for traversing elements in the First ArrayList of substring.
   3. Within that there is a for loop to traverse other Array Lists of substrings and nested is a loop to traverse all elements in it.
   4. Calculate edit distance of each substring and if it satisfies the condition then store its index.
   5. Check if in any string you are not able to find any neighbor then there is no need to look for neighbors in the strings ahead just start looking neighbors in another substring from 1st array.
   6. If you are able to find neighbor in other 19 strings then add all the neighbors into an ArrayList.
   7. After getting all the neighbors print the non-duplicate ones.
3. **private static int calculateEditDistance (String substring, String neighbourSubstring, int indel, int sub)**
   1. Initialize a memorization array for DP.
   2. Nested for loops that gets each character of the two strings and check what type of operation needs to be performed on them.
   3. Get the minimum cost required to convert substring to neighbourSubstring.
   4. Return the minimum cost from the array.
4. **private static ArrayList<String> createAllSubstrings (String dNASequence, int L)**
   1. A nested for loop which creates substring from the provide sequence
   2. Check if it is equal to length L then store it in the ArrayList.
   3. Return the ArrayList with all the substring of the given string.
5. **private static String createRandomString (char [] types)**
   1. For loop which runs for 600 times generate a random integer from 0 to 3 and keep adding it to the string by getting the character from types.
   2. Returns the final random string of length 600

**Improvement**

Currently program write in the file multiple times hence increasing the execution time. We can create a StringBuilder where we would just append all the output in it and then after every algorithm executes then just write it in the file. This will reduce the execution time considerably.