**Aditya Srivastava  
1910110034**

**DAA Assignment - 1**

**Analysis**

We first take input from “Data.txt” then we will have to take 20 random strings, then substrings from length L – D to L + D for each string. Store substrings of each string in the ArrayList. Now we can say that if M is a substring i.e., of length L and M’ is its neighbor then edit distance that is (M, M’) which should be <= D (given by the user). So, we just have to traverse through all the substrings M from each ArrayList of the string, check all the neighbors M’ for it in the other Array Lists, and store M, if M finds a neighbor in all the strings i.e., the edit distance of M and M’ satisfies the condition provided in the question. Finally, write all the Ms in the “Out.txt”.

**Worst case time complexity:   
(From printAllNeighboursWEditDistance (ArrayList<ArrayList<String>> neighbourSubstring, int D, int indel, int sub, int L))**

**A:** 20 (size of neighbourSubstring.size())  
**N:** number of elements in ArrayList for checking M  
**B:** 20 (size of neighbourSubstring.size())  
**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 (20.N.20.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 in each list that contains list of all the substrings.
   3. For loop for traversing elements in the ArrayList of substring.
   4. Within that first check if it is equal to length L as our M should be of length L if it is then only then there is a for loop to traverse other Array Lists of substrings (other than the one we are checking neighbors from) and nested is a loop to traverse all elements in it.
   5. Calculate edit distance of each substring and if it satisfies the condition then increment count and break from the loop as we have checked that this string contains a neighbour.
   6. Check from count 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.
   7. If you are able to find neighbor in other 19 strings then store M.
   8. After getting all the M 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 between length L-D to L+D and 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. We can use the reflexive property of the edit distance to make it more efficient by storing the index and not traversing it again.