BIG DATA PROGRAMMING

EXERCISE 1

Approach:

The approach to solving this problem of finding the longest common subsequence is to make use of a recursive function i.e. a function that calls itself. This method is commonly used when we need to generate a sequence ex. Fibonacci etc.

Here we give two text inputs to the function that are needed for finding the LCS between them, namely String 1 and String 2. Check if both strings are null, then return " ". After this, firstly we compare the first characters of both the strings, if equal it could be the beginning of a sequence, so we keep that character and call the same function again giving the remaining characters as the input of both the strings. This keeps happening till the strings have similar characters in the beginning and once it encounters a different character, then the character of the first string is kept common and we recurse the function with the remaining character of the second string removing one from the beginning each time until it encounters the same. If it does, then this is added on. This keeps happening until the longest common subsequence is identified.

Pseudocode:

```
Step 1: START
Step 2: Define function LCS with 2 inputs LCS(String1,String2)
Step 3: Check if String 1 and String 2 are not null
Step 4: Split 1st character of both strings as a, b and remaining as s1,s2
Step 5: Compare a==b
Step 6: If a == b, Return a + Go back to Step 2 with LCS(s1,s2)
Step 7: Else Return maximum of Step 2 with LCS(String1,s2) and LCS(s1,String2)
Step 8: STOP
```

EXERCISE 2

Approach:

The approach to solving this problem of matching one strings with another that has wild characters is to make use of a recursive function again.

Here we give two text inputs to the function that are needed, one is the base string to be matched and a matching string that consists of wild characters. Firstly, we check if after the '*' there are characters in the first string. This is done by checking the lengths of the strings. Next, if the matching string contains a '.' that signifies any single character, or if the current characters of both the strings match, then recurse the match function with the remaining characters of both the strings. There exists two probabilities if there is a '*', to consider the current character of the first string or to ignore it. This condition must be checked and accordingly recursed else to return false. Now this recursing will stop and return true only when both the strings length has been reduced to 0.

Pseudocode:

Step 1: START

Step 2: Define function LCS with 2 inputs Match(String1,String2)

Step 3: Check if String 1 and String 2 are not of length 0 else return **TRUE**

Step 4: Check if length of String 2> 1 AND String2[0]=='*' AND length of String1 == 0 then return **FALSE**

Step 5: Check if length of String 2 > 1 and String2[0]=='.' OR Length of both strings ! == 0 AND String1[0] == String2[0] only then return **Match** of Remaining character of both String1 , String 2

Step 6: Check if Length of String2 !=0 AND String [0] =='*' the **Match** (String1, remaining of String2) or **Match** (remaining of String1, String 2) **else** return **FALSE**

Step 8: STOP

GITHUB LINK: https://github.com/anish29292/BigDataProgramming

Matrikulation : 11012097 (Anish Umesh)