ASSINGMENT 1

# BIG DATA PROGRAMMING

RISHABH Garg (SRH Hochschule Heidelberg)

# Table of Contents

# Problem 1

## Finding the Longest Common Subsequence of Two Strings

GitHub Link: https://github.com/rishabh191292/Big-Data-Programming.git

## Approach

Before starting the actual implementation of the above problem. The concept was thoroughly studied. For better understanding of the problem in hand, various open source platforms present on the internet were referred. After carefully studying these open sources which were mostly based on the dynamic algorithm structure using recursion as the logic to implement the problem of Longest subsequence problem. The approach used by me is not based on recursive method rather than simply traversing the two strings and checking all the characters of 2nd string with the 1st string. The matching characters are subsequently appended to a new string variable. This new string what we get is the Longest subsequence present. The above problem is implemented in the c++ language. The code is not the best of the implementation and might break for very long strings and with strings having multiple longest subsequences. Basic c++ libraries such as iostream & string are used in the program for solving the above problem.

## Algorithm/PSEUDO CODE

Step 1: Two strings are taken as input from the user and are stored as str1 & str2.

Step 2: The length of both the strings are computed using an inbuilt function present in the string library called as strlen(). The Lenghts of both the strings are computed and saved in strlen1 & strlen2.

Step 3: An explicit function by the name of abc() has been made which computes the longest subsequence present. This function has been called from our main() and string 1, string 2, length of string 1 & length of string 2 has been passed as arguments

Step 4: If any of the passed strings str1 & str2 are null then the program ends as it is an undesirable condition.

Step 5: Two for loops are created, the outer for loop runs till the length of string1 and the inner for loop runs till the length of the string2. The starting index of the second loop is not from 0 rather than a local variable position1 initialized -1. The value of the position1 would be computed and stored as soon as there would be a match.

Step 6: All the characters in both the string are checked for possible match using an if condition. If there is a hit then the matched character is appended to a new string variable created locally. and the value of position1 is updated by adding 1 to the inner loop operator's current value (position1 = j+1). After a successful match the current iteration is stopped by using break statement in order to start a new iteration.

Step 7: The above procedure is repeated if in case the length of our string 1 is less than the 2nd string i.e string2. All the conditions remain same only thing which has been changed is the outer loop runs till the end of string2 and inner loop runs till the end of the string1, another integer variable position2 which stores the new initializing

for the 2nd loop every time a match has been found. All the matched characters are stored in a new string variable lcs2.

Step 8: For longest substring, the sizes of both lcs1 & lcs2 are compared and correspondingly the Longest of the two is printed.

## Problem 2

Write a Function that Takes two Strings S1 & S2 as its input and returns a Boolean denoting whether S1 matches S2.

### Approach

The approach for solving the above problem is same as stated above for the problem. The expected output accuracy of about 50% - 60%. The problem again is implemented on c++. The problem is divided into three functions. The first function is computing for the values where the pattern string has alphabets as well as stars only. The second function is computing for the values where the pattern string has alphabets as well as dots only. The third function is computing for the values where the pattern string has alphabets as well as stars & dots.

### Algorithm/Pseudo Code

**matchstar(text[], pattern[], textlength, patternlength)**
{
Initialize & declare integer values: i , j , boolen,counter = 0;
if(textLenght is zero)||(patternLength == 0)
  THEN :Print Invalid.
Else if(if the pattern string has only a star)
  THEN:Print no match possible
Else if(the first character in the pattern string has a * and also has other alphabets)
  For: till the end of the pattern string
    Pattern[i-1] = pattern[i]  // deleting the character *
    If: length of pattern string <length of text string
     Then: For( run a loop till [patternLength -1] )
        if(characters in the pattern string matches characters in text str)
          Update the boolen variable to 1
        Else: update the boolen value as 0
    Else
      For: run the loop till the end of input string
        If: check all the characters in pattern matches the input string
          Then: Update boolen to 1;
        Else: update the Boolen 0;

    Else

For: run it till the end of input string
For: run it till the end of pattern string
If: pattern string has a * [star character]
If: all text string characters matches the character just before the * in the pattern string
Then: Update counter as 1
Else if: if counter has a value 1 & pattern[j+1] is not null
Then: Update boolen as 1
Else
Replace star character in the pattern string with previous character
Else if(match all the characters in the pattern string with input text string)
Then: Update the boolen value to 1
Else Boolen value is updated to zero
If:boolen value has is 1
Then: print string match
Else: printNo match
}

**Matchdot(text[],pattern[],textlength, patternlength)**

{
Local Variables are created such as I,j,pos=0, boolen=0;
If(pattern string is of the form "." & input text string s1 is of length 1 only)
{
Print Perfect match
}
Run a for loop: for(till the end of text length)
{
If(the characters not matching in the input text string and at the same positions if there are dots in the pattern )
{
Replace all these characters by the input text string characters for these positions
}
}
If(length of text string is greater than length of pattern string)
{
For(run a loop till the end of the pattern length)
{
If(check if all the characters in the pattern string matches the s1)
{
Update the value of boolen to 1
}
Else
{
Boolen remains 0;
}
}

```
}
Else
{
        For(run a loop till the end of input text string)
        {
                If(check if all the characters in the pattern string matches the s1)
                {
                        Update the value of boolen to 1
                }
                Else
                {
                        Boolen remains 0;
                }
        }
}
If(value of boolen is 1) : Perfect match
Else: No match
```

**Matchdotstar(text[],pattern[],textLength,PatternLength)**

```
{
The local variables I,j,boolen and counter are declared and initialised to 0
If(pattern string is of the form ".*"): it is a perfect match
For: till the end of the pattern length
If: (pattern string does not have any character & the first character of the pattern string is a.
   star)
        then: No match can be found
For: till the end of input text string
    For: till the end of pattern string
        If: pattern has a "*"
          If: previous character is same as all the character in the input text string
                i=i+1    // Increase the outer loop counter by 1
          else:
                j=j+1    // increase the inner loop counter by 1
        else if: (if the pattern string has a "." && characters in the text string does not match
                the characters in the pattern string)
                then: replace the characters in the pattern string with the input text string
                        characters
        else if: match the characters in the pattern string with the input text string
                then : if all the characters match then update the counter as 1
        else: counter value remains 0 only

if:value of the counter is 1
  then: print perfect match
else: no match found
}

Main()
```

{
Declare char arrays text[20] and pattern array [20]
Declare integer values choice, patternLengths2, inputLengths1
Ask for the user input to put in choice of pattern string he/she wishes to input

1. Pattern string with only stars
2. Pattern string with only dots
3. Pattern string with a combination of dots and stars

Ask the user to input the length of the string s1
Ask the user to input the length of the pattern string s2
If: choice entered by the user is 1
Then: function call matchstar(text[20],pattern[20],inputlengths1,patternlengths2)
Else if: choice entered by the user is 2
Then: function call matchdot(text[20],pattern[20],inputlengths1,patternlengths2)
Else if: choice entered by the user is 3
Then: matchdotstar(text[20],pattern[20],inputlengths1,patternlengths2)
Else: print wrong choice

Return 0
}