Dr BR Ambedkar National Institute of Technology
Jalandhar-144011, Punjab, India.

Department of Computer Science & Engineering

Submitted to :

Dr Nonita Sharma

Submitted by:

Godala Soma Sandeep Reddy	**Roll no.20103059**
Gourav Gujariya	**Roll no.20103060**
Lovepreet Kumar	**Roll no.20103084**

Section A(group 3)

Table of contents

- **Problem statement:**
- Given the two string sequences, find the length of the longest common subsequence present in both of the strings and the subsequence itself.
- A longest common subsequence is a sequence that appears in the same relative order but is not necessarily contiguous; it should be a subset of the other string. For **example,** “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”.
- **Input:**
- case 1-
    - S1 = "ababaa"
    - S2 = "baba"
- **Output:**"baba"
    - length:4
- case 2-
    - S1 = "bcdaacd"
    - S2 = "acdbac"
- **Output:**"cdac"
    - Length:4

Approach:

- To find the longest common subsequence of the given string the following approaches could be taken:

1. a brute force where we need to first know the number of possible different subsequences of a string with length n, i.e., find the number of subsequences with lengths ranging from 1,2,..n-1.from the theory of permutation and combination that several combinations with 1 element are nC1. Many combinations with 2 elements are nC2 and so forth and so on. We know that nC0 + nC1 + nC2 + … nCn = 2n. So a string of length n has 2n-1 different possible subsequences since we do not consider the subsequence with length 0. This implies that the time complexity of the brute force approach will be O(n * 2n).
2. By using Dynamic programing

- The solution to this classic problem of dynamic programming possesses important properties of a dynamic programming problem

1. *Optimal Substructure:*
   a. Let the first string sequences and second string sequence be X[0..m-1] and Y[0..n-1] of lengths m and n respectively. And let L(X[0..m-1], Y[0..n-1]) be the length of LCS of the 2 sequences X and Y.
   b. Algorithm approach will be Following the recursive definition of L(X[0..m-1], Y[0..n-1]).If last characters of both sequences match (or X[m-1] == Y[n-1]) then L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2]) If last characters of both sequences do not match (or X[m-1] != Y[n-1]) then L(X[0..m-1], Y[0..n-1]) = MAX (L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2])).
   c. Consider the input strings "ABCDGH" and "AEDFHR. The last characters do not match the strings. So the length of LCS can be written as:L("ABCDGH", "AEDFHR") = MAX (L("ABCDG", "AEDFH**R**"), L("ABCDG**H**", "AEDFH"))
   d. So the LCS problem has optimal substructure property as the main problem can be solved using dynamic programming property of solving and using the solution of subproblems.
2. *Overlapping Subproblems:* overlapping of subproblems is seen while solving the problem and by using recursion we can see that many of the subproblems are being solved a number of times so by using a method of dynamic programming as memorization or tabulation we could reduce the time complexity

Pseudocode:

Let's take two strings for subsequence x and y of length m,n respectively and create a vector for memorizing the values which have been calculated of size n+1 with value -1.
Function lcs which will iterate through the string and update the vector

```
Fun lcs(char x,char y,int m,int n,vector){
        if(m==0 or n==0)
                Return 0;
        if(X[m-1]==Y[n-1])
                Return vector[m][n]=1+lcs(X,y,m-1,n-1,dp);
        If (dp[m][n]!=-1){
                Return vector[m][n];}
        Return vector[m][n] = max(lcs(X, Y, m, n - 1, dp), lcs(X, Y, m - 1, n, dp));
```

Code:

```python
import streamlit as st

def lcs_algo(S1, S2, m, n):
  L = [[0 for x in range(n + 1)] for x in range(m + 1)]

  # Building the mtrix in bottom-up way
  for i in range(m + 1):
      for j in range(n + 1):
          if i == 0 or j == 0:
              L[i][j] = 0
          elif S1[i - 1] == S2[j - 1]:
              L[i][j] = L[i - 1][j - 1] + 1
          else:
              L[i][j] = max(L[i - 1][j], L[i][j - 1])

  index = L[m][n]
```

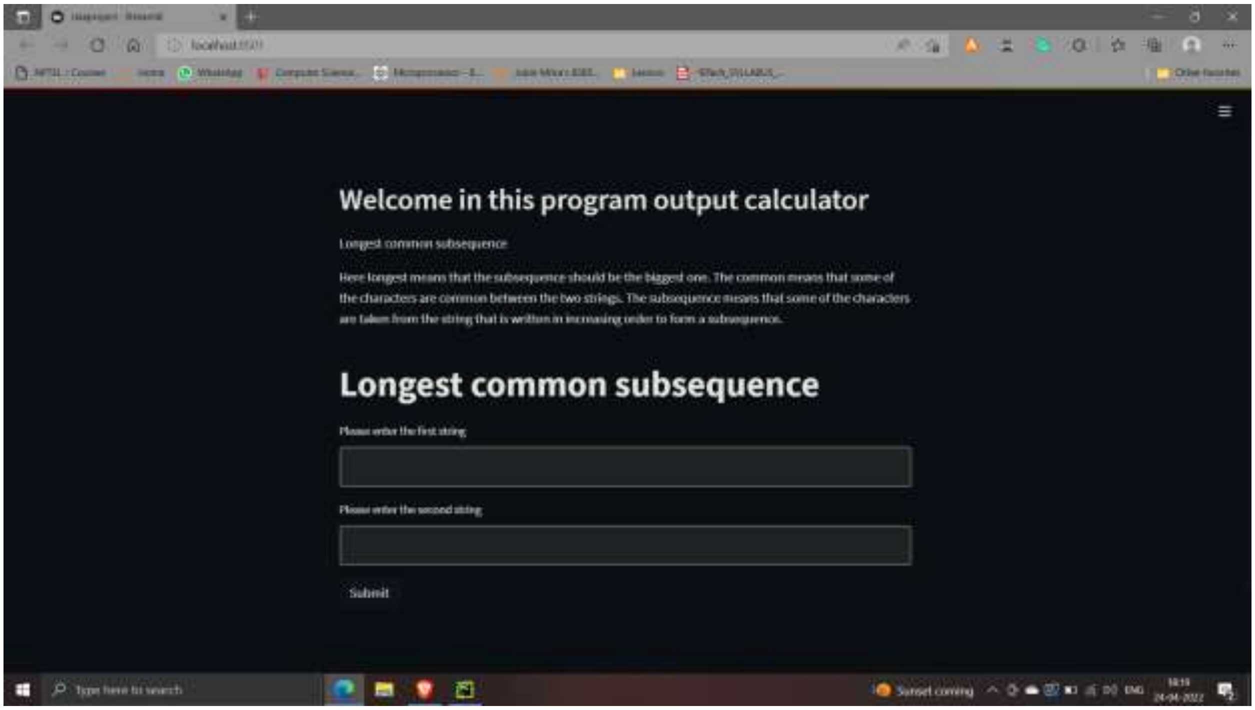
```python
  lcs_algo = [""] * (index + 1)
  lcs_algo[index] = ""

  i = m
  j = n
  while i > 0 and j > 0:

      if S1[i - 1] == S2[j - 1]:
          lcs_algo[index - 1] = S1[i - 1]
          i -= 1
          j -= 1
          index -= 1

      elif L[i - 1][j] > L[i][j - 1]:
          i -= 1
      else:
          j -= 1

  # Printing the sub sequences
  s = "".join(lcs_algo)
  st.success("LCS = " + s)
  st.success("LCS length = {}.".format(len(s)))


def LCS():
  st.title('Longest common subsequence')
  s1 = st.text_input("Please enter the first string")
  #   st.write(type(s1),s1)
  s2 = st.text_input("Please enter the second string")
  #   st.write(type(s2),s2)
  submit = st.button('Submit')
  if submit:
      m = len(s1)
      n = len(s2)
```

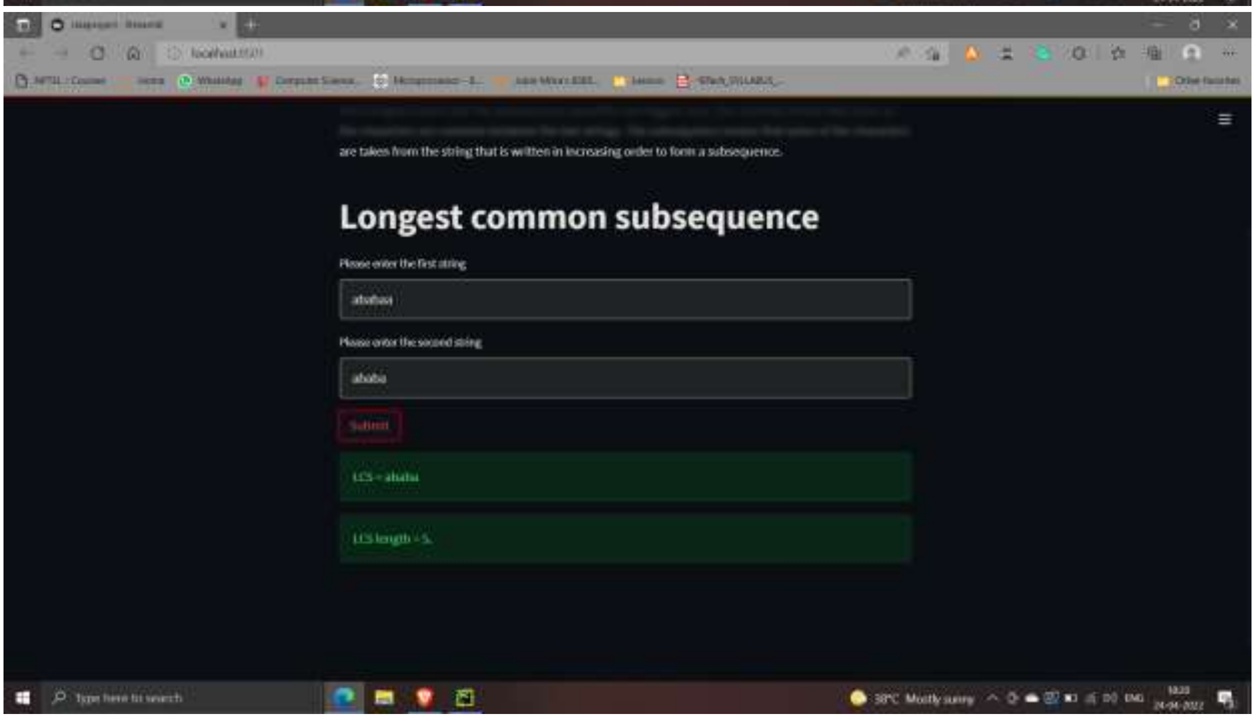
```
        lcs_algo(s1, s2, m, n)


# end of function LCS


# Driver Code
if __name__ == "__main__":
    st.header('Welcome in this program output calculator')
    st.write('Longest common subsequence')
    st.write('Here longest means that the subsequence should be the biggest one.'
             ' The common means that some of the characters are common between the two strings. '
             'The subsequence means that some of the characters are taken from the string that is '
             'written in increasing order to form a subsequence.'
             )
    LCS()
```

Some important steps for implementing this program :

1. Install streamlit using (!pip install streamlit)
2. Install python 3 or higher version
3. Run code in pycharm and run by the help of terminal
4. paste(streamlit run ‘the file location’)

Output :





Time complexity:

The overall time complexity of the program is O(m,n)
We observe that our algorithm takes time equal to the length of the strings we entered so the overall time complexity will be O(m*n)