EAST WEST UNIVERSITY

Lexicographically Smallest and Longest Common Subsequence

Course Name: Algorithms
Course: CSE 246
Section: 05
Semester: Fall2022
Group No: 05
Project Problem ID: 17

Submitted By:

Student1:Miftahul Kamal Jannat	2020-1-60-231
Student2: Sadia Islam Maria	2020-1-60-276
Student3:MD.Yeasin Arafath Emon	2020-2-60-182

Submitted To:

Redwan Ahmed Rizvee
Lecturer, Department of Computer Science and Engineering East West University

Submission Date: 28 December, 2022

1.Problem Statement:

In the given project we need to find out the length of the longest common subsequence and also need to find the lexicographically smallest lcs observed in two given string containing any ASCII characters.

There will be two given string like str1=ABCDEF and str2=DEFABC and the output will 3 which is the length of the lcs and ABC is the lexicographically smallest Lcs.

2.Algorithm Discussion:

- Let's start with finding the length of the sub-sequence.

Consider the input sequences be s1[0..i-1] and s2[0..j-1] of lengths i and j respectively. And L(s1[0..i-1], s2[0..j-1]) be the length of LCS of the two sequences s1 and s2.

LCS will check the smallest LCS from Two given string s1 and s2.lexSmallest function take two string and one of the string give smallest LCS.At first n+1 and m+1 shaped 2D array Dp created where m,n are the length of the s1 and s2.Then it will repeat the character and check ,then it will fill up the Dp array.

There can be two cases in LCS. If the last character is match or the last character do not match.

1. If the last character match , then increment the length of LCS by 1 and process s1[m-1] and s2[n-1].

2. If the last character do not match , then then the length will be maximum value b/w s1[m-1][n] and s2[n][n-1].

```
// Java program to find length and lexicographically smallest LCS of two strings in
package lexsmallestlcs;
import java.util.Scanner;
public class LexSmallestLCS {
    // Function to find the length of the longest common subsequence of
    // sequences `s1[0...m-1]` and `s2[0...n-1]
    public static String lexSmallestLCS(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();

// dp matrix to store result of sub calls for lcs
        int[][] dp = new int[m + 1][n + 1];

    // Building the mtrix in bottom-up way
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                // if the last character of `s1` and `s2` matches
                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {

                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    // otherwise, if the last character of `s1` and `s2` don't match
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }

        // Following code is used to print LCS
        int lcsLength = dp[m][n];
```

After the Dp array is filled, the method calculates the length of LCS by taking the value in the last cell of dp. It then creates a new string and iterates backwards through the dp array starting from the last cell to create the LCS string.
At each step, checks whether the current subscripts of s1 and s2 are equal. If they are, it appends the string character from s1 and decrements both i and j to move to the next cell in the dp array. If they are not equal, it moves to the dp cell with the highest value, either on the top or on the left.
Finally, the method reverses the string and returns the LCS string by calling its toString method.

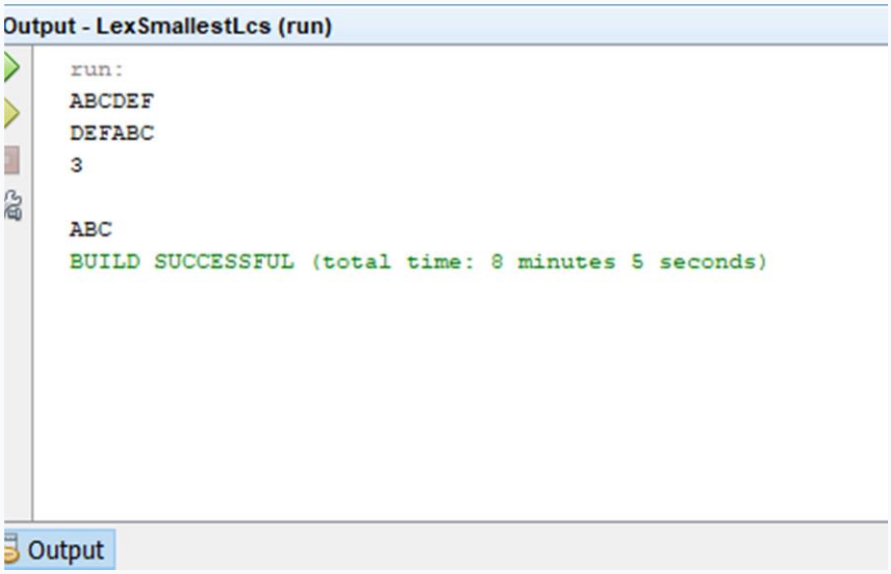
The main method reads in two strings from the user, calls the lexSmallestLCS method with the strings, and prints LCS and the length of LCS.

```
    // Create a StringBuilder object
     // using StringBuilder() constructor
     StringBuilder sb = new StringBuilder();

    // one by one store characters in lcs[]
     int i = m;
     int j = n;


     while (lcsLength > 0) {
         // If current character in s1[] and s2 are same, then
   // current character is part of LCS
         if (s1.charAt(i - 1) == s2.charAt(j - 1)) {

             sb.append(s1.charAt(i - 1));// Put current character in result
             i--;                        // reduce values of i, j and index
             j--;
             lcsLength--;
         } else {
             // If not same, then find the larger of two and
             // go in the direction of larger value
             if (dp[i - 1][j] > dp[i][j - 1]) {
                 i--;
             } else {
                 j--;
             }
         }
     }

     // reverse the string
     // print string
     return sb.reverse().toString();
 }

 public static void main(String[] args) {
     Scanner in = new Scanner(System.in);
     String s1 = in.nextLine();
     String s2 = in.nextLine();
     // Printing the sub sequence length
     System.out.println(lexSmallestLCS(s2, s1).length());
     System.out.println(" ");
     // Printing the sub sequence
     System.out.println(lexSmallestLCS(s2, s1));
 }
```

6. discussing input and output format of the code (using test cases):

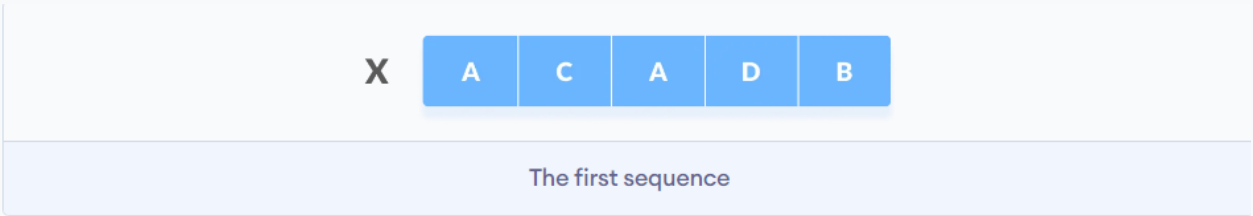
Input: Two given string ABCDEF and DEFABC.

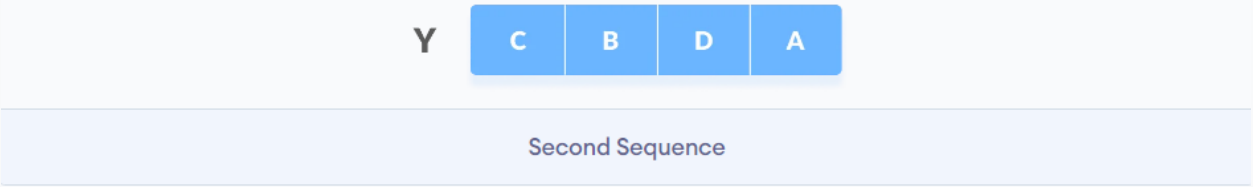
Output: At first Create a table of dimension `n+1*m+1` where n and m are the lengths of `s1` and `s2` respectively. The first row and the first column are filled with zeros. Fill each cell of the table using the following logic. If the character correspoding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell. Else take the maximum value from the previous column and previous row element for filling the current cell. Point an arrow to the cell with maximum value. If they are equal, point to any of them. repeated until the table is filled. The value in the last row and the last column is the length of the longest common subsequence.



Simulation:

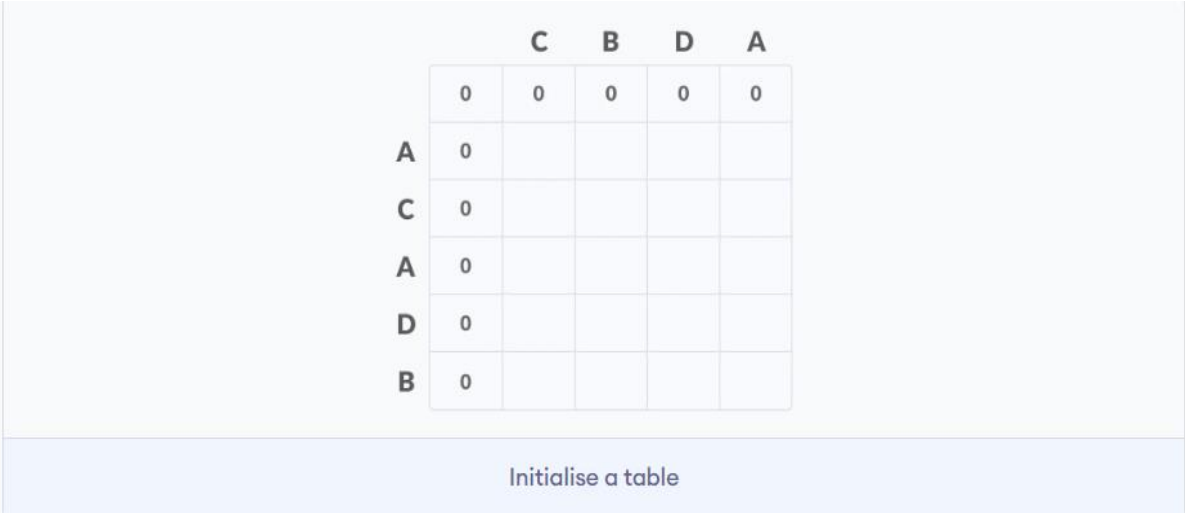
Let us take two sequences:



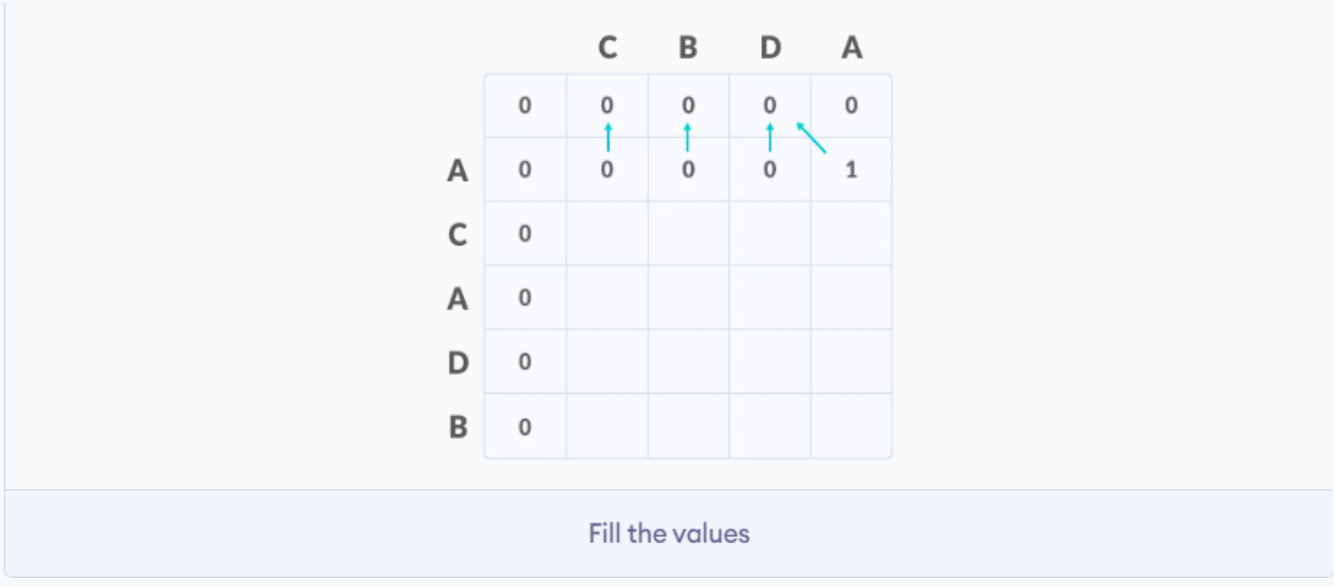


The following steps are followed for finding the longest common subsequence.

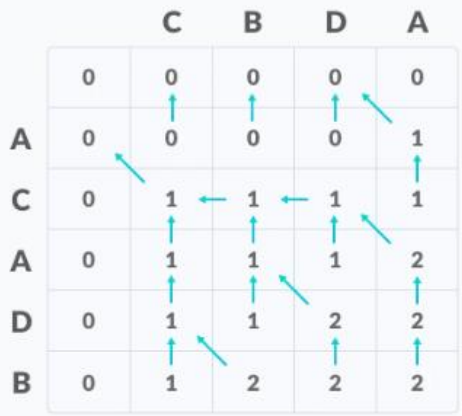
1. Create a table of dimension `n+1*m+1` where n and m are the lengths of `X` and `Y` respectively. The first row and the first column are filled with zeros.



2. Fill each cell of the table using the following logic.
3. If the character correspoding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell.
4. Else take the maximum value from the previous column and previous row element for filling the current cell. Point an arrow to the cell with maximum value. If they are equal, point to any of them.

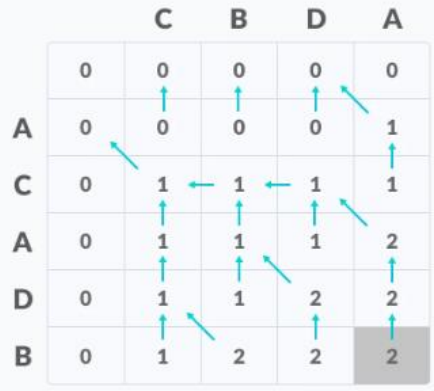


5.Step 2 is repeated until the table is filled.



Fill all the values

6.The value in the last row and the last column is the length of the longest common subsequence.



The bottom right corner is the length of the LCS

3. Complexity Analysis:

Time Complexity:

- Worst case time complexity: O(m*n)
- Average case time complexity: O(m*n)
- Best case time complexity: O(m*n)
- Space complexity: O(m*n)

Since we are using two for loops for both the strings ,therefore the time complexity of finding the longest common subsequence using dynamic programming approach is O(m * n) where n and m are the lengths of the strings.

Space Complexity:

O(m*n) since we are using m*n size array.

4. Implementation:

Proper commenting of the code is given below:

```java
// Java program to find length and lexicographically smallest LCS of two strings in
package lexsmallestlcs;
import java.util.Scanner;
public class LexSmallestLCS {
    // Function to find the length of the longest common subsequence of
    // sequences `s1[0...m-1]` and `s2[0...n-1]
    public static String lexSmallestLCS(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();

// dp matrix to store result of sub calls for lcs
        int[][] dp = new int[m + 1][n + 1];

     // Building the mtrix in bottom-up way
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                // if the last character of `s1` and `s2` matches
                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {

                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    // otherwise, if the last character of `s1` and `s2` don't match
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }

        // Following code is used to print LCS
        int lcsLength = dp[m][n];
```

```java
     // Create a StringBuilder object
      // using StringBuilder() constructor
      StringBuilder sb = new StringBuilder();

     // one by one store characters in lcs[]
      int i = m;
      int j = n;


      while (lcsLength > 0) {
          // If current character in s1[] and s2 are same, then
    // current character is part of LCS
          if (s1.charAt(i - 1) == s2.charAt(j - 1)) {

              sb.append(s1.charAt(i - 1));// Put current character in result
              i--;                        // reduce values of i, j and index
              j--;
              lcsLength--;
          } else {
              // If not same, then find the larger of two and
              // go in the direction of larger value
              if (dp[i - 1][j] > dp[i][j - 1]) {
                  i--;
              } else {
                  j--;
              }
          }
      }

      // reverse the string
      // print string
      return sb.reverse().toString();
  }

  public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      String s1 = in.nextLine();
      String s2 = in.nextLine();
      // Printing the sub sequence length
      System.out.println(lexSmallestLCS(s2, s1).length());
      System.out.println(" ");
      // Printing the sub sequence
      System.out.println(lexSmallestLCS(s2, s1));
  }
```

In this algorithm, we have used function to find the length of the longest common subsequence of and created an Dp matrix to store result of sub calls for LCS. Building the matrix in bottom-up way to check if the last character of `s1` and `s2` matches or not.

Constructs a string builder with no characters in it and an initial capacity of 16 characters. It is used to append the specified string with this string.

If current character in s1[] and s2 are same, then current character is part of LCS If not same, then find the larger of two and go in the direction of larger value.

Reverse the string and print string.

4. Applications (Bonus)

Lexicographical order is nothing but the dictionary order or preferably the order in which words appear in the dictionary. For example, let's take three strings, "short", "shorthand" and "small". In the dictionary, "short" comes before "shorthand" and "shorthand" comes before "small". This is lexicographical order.