****

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE)**

**Faculty of Sciences and Engineering**

**Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)**

**Lab Report NO 03**

**Course Title: Algorithm Lab**

**Course Code: CSE 208**

**Section: D9**

**Lab Experiment Name:** Implementation of Longest Common Subsequence (LCS) Algorithm.

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Rukonuzzaman Topu | 232002280 |

**Lab Date: 10/04/25**

**Submission Date: 16/04/25**

**Course Teacher’s Name: Farjana Akter Jui**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature: .....................**  **Comments: .............................................. Date: ..............................** |

**1. INTRODUCTION**

This lab focuses on identifying common subsequences between two strings. The first program is designed to generate all distinct common subsequences and display them in order from longest to shortest. The second program calculates the length of the Longest Common Subsequence (LCS) using a recursive method. Together, these programs illustrate the use of recursion and string matching techniques in Java.

**2. OBJECTIVES**

The primary objectives of this lab report are as follows:

* To learn about Longest Common Subsequence (LCS) algorithm for determining the length of common subsequences in strings.

**3. PROCEDURE**

**Code 1:**

* Recursive Traversal: The find CommonSubseqence method navigates through both strings recursively to identify all matching subsequences.
* Unique Storage: Each identified common subsequence is stored in a result array only if it hasn't been added before.
* Descending Arrangement: Once all subsequences are gathered, they are sorted from longest to shortest based on their length.

**Code 2:**

* Recursive Evaluation: The lcs function determines the length of the longest common subsequence by recursively comparing characters and evaluating both options when there's no match.
* Termination Condition: If one of the strings reaches the end, the function returns 0.
* Length Optimization: When characters differ, the function returns the greater value between the two recursive paths to ensure the longest possible subsequence is considered.

**4. IMPLEMENTATION**

Task 1: Print all the common subsequences according to the descending order of the lengths for two given sequences.

Solution:

public class commonSubsequences {

    static String[] results = new String[1000];

    static int count = 0;

    public static void findCommonSubseq(char X[], char Y[], int m, int n, String current) {

        if (m == 0 || n == 0) {

            if (!current.equals("")) {

                addIfNotExists(current);

            }

            return;

        }

        if (X[m - 1] == Y[n - 1]) {

            findCommonSubseq(X, Y, m - 1, n - 1, X[m - 1] + current);

        } else {

            findCommonSubseq(X, Y, m - 1, n, current);

            findCommonSubseq(X, Y, m, n - 1, current);

        }

    }

    public static void addIfNotExists(String s) {

        for (int i = 0; i < count; i++) {

            if (results[i].equals(s)) {

                return;

            }

        }

        results[count++] = s;

    }

    public static void sort() {

        for (int i = 0; i < count - 1; i++) {

            for (int j = i + 1; j < count; j++) {

                if (results[i].length() < results[j].length()) {

                    String temp = results[i];

                    results[i] = results[j];

                    results[j] = temp;

                }

            }

        }

    }

    public static void main(String[] args) {

        String X = "XYZK";

        String Y = "YZKX";

        char arr1[] = X.toCharArray();

        char arr2[] = Y.toCharArray();

        int len1 = arr1.length;

        int len2 = arr2.length;

        findCommonSubseq(arr1, arr2, len1, len2, "");

        sort();

        System.out.println("Common subsequences (sorted by descending length):");

        for (int i = 0; i < count; i++) {

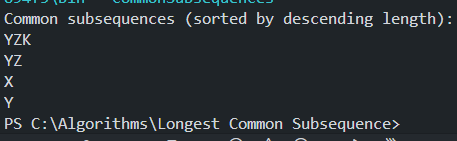
            System.out.println(results[i]);

        }

    }

}

Output:



Task 2: What will be the LCS for the sequences “ABCDEFGH" & “abcdefgh" ?

Solution:

import java.util.\*;

import java.lang.\*;

public class Topu{

    public static int lcs (char x[],char y[],int m,int n){

        if(m==0||n==0){

            return 0;

        }

        if(x[m-1]==y[n-1]){

            return 1+lcs(x,y,m-1,n-1);

        }else {

            return max(lcs(x,y,m,n-1),lcs(x,y,m-1,n));

        }

    }

    static int max(int l1, int l2) {

        return (l1 > l2) ? l1 : l2;

    }

public static void main(String arg[]){

Topu lcs=new Topu();

String x="ABCDEFGH";

String y="abcdefgh";

char arr1[]=x.toCharArray();

char arr2[]=y.toCharArray();

int len1=arr1.length;

int len2=arr2.length;

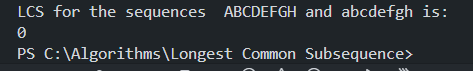
System.out.println("LCS for the sequences  ABCDEFGH and abcdefgh is:");

System.out.print(lcs(arr1, arr2, len1, len2));

}

}

Output:

****

**4. DISCUSSION**

The first program finds all different common subsequences between two strings using recursion and sorts them by length. It shows many ways the strings can match. The second program only finds the length of the Longest Common Subsequence using a basic recursive method. Both programs show how to compare strings, but they are slow for big inputs because they don’t use faster methods like dynamic programming. Together, they help us understand the difference between finding all matches and finding just the longest one.