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.	Ashab Uddin	232002274

Lab Date: 26-02-2025

Submission Date: 04-03-2025

Course Teacher's Name: Farjana Akter Jui

Lab Report Status

Marks: ………………………………… Signature: ....

Comments: .... Date: ....

1. INTRODUCTION

This lab is about finding common parts (subsequences) between two strings. The first program shows all the different common subsequences and arranges them from longest to shortest. The second program uses recursion to find out how long the longest common subsequence (LCS) is. Both programs help us understand how to compare strings and use recursion in Java.

2. OBJECTIVE

- To understand the Longest Common Subsequence (LCS) algorithm and how it helps find the length of common parts between strings.

3. PROCEDURE

Task- 1:

function-1: Using Recursion: The findCommonSubseqence method uses recursion to search for all common subsequences by checking matching characters in both strings**.**

function-2: Storing Subsequence: Each found common subsequence is saved in a list, but only if it's not already there.

function-3: Sorting the Results: After collecting all the subsequences, they are sorted from longest to shortest.

Task-2:

The **`lcs`** method works using a recursive process. It checks characters from both strings one by one. If the characters match, it adds 1 to the result and moves to the next pair. If they don't match, the method tries two possible paths — skipping a character from the first string or the second — and continues the process.

The base condition of the recursion is when one of the strings becomes empty. At that point, it returns 0, because there are no more characters to compare.

When the characters don't match, the function chooses the longer result between the two recursive paths. This helps in finding the longest possible common part (subsequence) between the two strings.

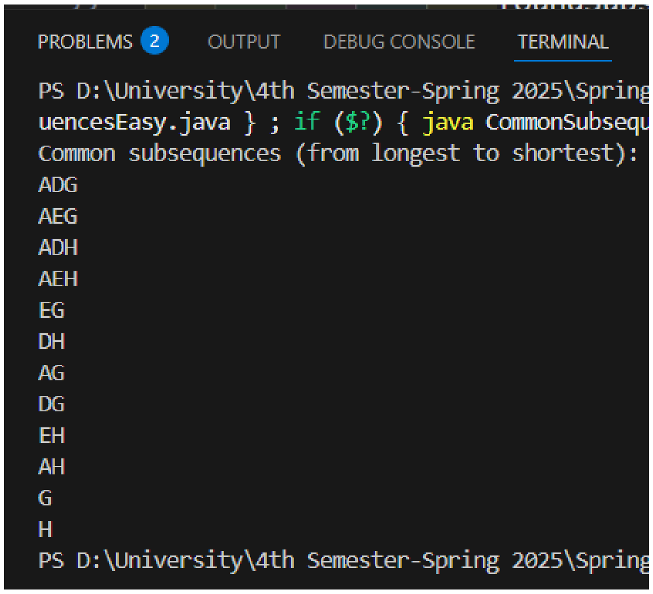
4. IMPLEMENTATION

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

<u>Solution Code:</u>

```
public class CommonSubsequencesEasy {
    static String[] foundSubsequences = new String[1000];
    static int numberOfSubsequences = 0;

    public static void findCommonSubsequences(char[] stringA, char[] stringB, int
indexA, int indexB, String currentSubsequence) {
        if (indexA == 0 || indexB == 0) {
            if (!currentSubsequence.isEmpty()) {
                saveIfNotDuplicate(currentSubsequence);
            }
            return;
        }

        if (stringA[indexA - 1] == stringB[indexB - 1]) {
            findCommonSubsequences(stringA, stringB, indexA - 1, indexB - 1,
stringA[indexA - 1] + currentSubsequence);
        } else {
            findCommonSubsequences(stringA, stringB, indexA - 1, indexB,
currentSubsequence);
            findCommonSubsequences(stringA, stringB, indexA, indexB - 1,
currentSubsequence);
        }
    }
    public static void saveIfNotDuplicate(String newSubsequence) {
        for (int i = 0; i < numberOfSubsequences; i++) {
            if (foundSubsequences[i].equals(newSubsequence)) {
                return;  // Duplicate found, skip it
            }
        }
        foundSubsequences[numberOfSubsequences++] = newSubsequence;
    }
    public static void sortSubsequencesByLength() {
        for (int i = 0; i < numberOfSubsequences - 1; i++) {
            for (int j = i + 1; j < numberOfSubsequences; j++) {
                if (foundSubsequences[i].length() < foundSubsequences[j].length()) {
                    String temp = foundSubsequences[i];
                    foundSubsequences[i] = foundSubsequences[j];
                    foundSubsequences[j] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        String firstText = "ABCDEFGH";
        String secondText = "AEDHG";
        char[] charsOfFirst = firstText.toCharArray();
        char[] charsOfSecond = secondText.toCharArray();

        findCommonSubsequences(charsOfFirst, charsOfSecond, charsOfFirst.length,
charsOfSecond.length, "");

        sortSubsequencesByLength();

        System.out.println("Common subsequences (from longest to shortest):");
        for (int i = 0; i < numberOfSubsequences; i++) {
            System.out.println(foundSubsequences[i]);
        }
    }
}
```

Output:



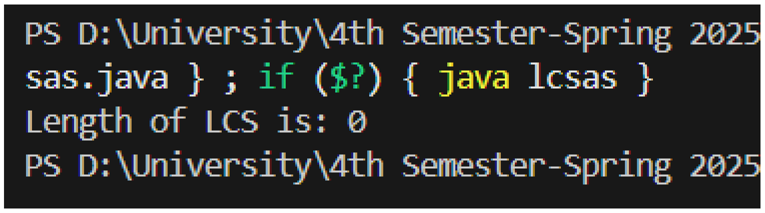
```
PS D:\University\4th Semester-Spring 2025\Spring
uencesEasy.java } ; if ($?) { java CommonSubsequ
Common subsequences (from longest to shortest):
ADG
AEG
ADH
AEH
EG
DH
AG
DG
EH
AH
G
H
PS D:\University\4th Semester-Spring 2025\Spring
```

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

Solution:

```
public class lcsas {

    public static int findLCS(char[] str1, char[] str2, int len1, int len2) {

        if (len1 == 0 || len2 == 0) {
            return 0;
        }
        if (str1[len1 - 1] == str2[len2 - 1]) {
            return 1 + findLCS(str1, str2, len1 - 1, len2 - 1);
        } else {
            return max(
                findLCS(str1, str2, len1, len2 - 1),
                findLCS(str1, str2, len1 - 1, len2)
            );
        }
    }

    public static int max(int a, int b) {
        return (a > b) ? a : b;
    }

    public static void main(String[] args) {
        String text1 = "ABCDEFGH";
        String text2 = "abcdefgh";

        char[] chars1 = text1.toCharArray();
        char[] chars2 = text2.toCharArray();

        int length1 = chars1.length;
        int length2 = chars2.length;

        int result = findLCS(chars1, chars2, length1, length2);
        System.out.println("Length of LCS is: " + result);
    }
}
```

Output:



```
PS D:\University\4th Semester-Spring 2025
sas.java } ; if ($?) { java lcsas }
Length of LCS is: 0
PS D:\University\4th Semester-Spring 2025
```

4. DISCUSSION

The first program explores all possible unique common subsequences through recursion and organizes them by their length in descending order, showcasing the variety of potential matches between two strings. This method identifies every possible subsequence, providing a comprehensive view of how the strings relate to one another.

The second program, on the other hand, calculates only the length of the Longest Common Subsequence (LCS) using a basic recursive method. While it’s more focused than the first, it’s not optimized for large inputs because it doesn’t use dynamic programming. Together, these programs show the difference between finding all subsequences and just calculating the length of the longest one.