GREEN UNIVERSITY OF BANGLADESH

Department of Computer Science and Engineering (CSE)

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

Lab Report No: 04

Experiment Name: Implemention of Longest Common Subsequence (LCS) Algorithm

Course Title : Algorithm lab
Course Code: CSE 208
Section: D8

Student Details:

Name	ID
Majharul Islam	2320022256

Submission Date. : 07.05.2025
Course Teacher's Name: Feroza Naznin.

[For Teachers use only: Don't Write Anything inside this box]

Assignment Report Status	
Marks: ……	Signature:.....
Comments:.....	Date:.....

1. TITLE OF THE LAB REPORT EXPERIMENT

Implemention of Longest Common Subsequence (LCS) Algorithm

2. OBJECTIVES

The objective of this experiment is to implement a Java program that prints all common subsequences between two strings in descending order of their lengths. The problem enhances our understanding of dynamic programming, recursion, and string manipulation in Java.

3. PROCEDURE

We approach this problem by:

- **Understanding Subsequence**: A common subsequence is a sequence that appears in both input strings, not necessarily contiguously.
- **Generating Subsequences**: Use recursive backtracking to generate all subsequences of both strings.
- **Storing Matches**: Store all common subsequences in a set.
- **Sorting**: Use a list to sort these common subsequences in descending order based on their length.

Pseudocode Summary:

1. Generate all subsequences of string A.
2. Generate all subsequences of string B.
3. Find the intersection of the two subsequence sets.
4. Sort the resulting list by length in descending order.
5. Print the result.

4. IMPLEMENTATION

```java
import java.util.*;

public class CommonSubsequences{

    public static void generateSubsequences(String str, int index, String current, Set<String> result) {
        if (index == str.length()) {
            if (!current.isEmpty()) result.add(current);
            return;
        }
        generateSubsequences(str, index + 1, current + str.charAt(index), result);

        generateSubsequences(str, index + 1, current, result);
    }

    public static void main(String[] args) {
        String s1 = "ABCDEFGH";
        String s2 = "abcdefgh";

        Set<String> subseq1 = new HashSet<>();
        Set<String> subseq2 = new HashSet<>();

        generateSubsequences(s1, 0, "", subseq1);
        generateSubsequences(s2, 0, "", subseq2);
        Set<String> common = new HashSet<>(subseq1);
        common.retainAll(subseq2);

        List<String> sortedList = new ArrayList<>(common);
        sortedList.sort((a, b) -> b.length() - a.length());

        System.out.println("Common subsequences in descending order of length:");
        for (String s : sortedList) {
            System.out.println(s);
        }

        if (sortedList.isEmpty()) {
            System.out.println("No common subsequences found (case-sensitive match).");
        }
    }
}
```

---

5. TEST RESULT & OUTPUT

Test Case: "ABCDEFGH" and "abcdefgh"

Output:

No common subsequences found (case-sensitive match).

Additional Test Case: "ABCD" and "ABDC"

Output Example:

ABD
ABC
AB
AD
AC
BD
BC
CD
A
B
C
D

6. ANALYSIS AND DISCUSSION

- What went well?
  - Correct generation of subsequences using recursion.
  - Accurate filtering and sorting logic.
- Trouble spots?
  - Performance bottleneck for larger strings due to exponential growth of subsequences.
  - Ensuring no duplicates in final output.
- Most difficult part?
  - Designing an efficient and correct recursive subsequence generator
- What did you like?
  - Working with set operations and custom sorting in Java.
- What did you learn?
  - How to apply recursion and backtracking for combinatorial problems.
  - Understanding the role of data structures (Set, List) in filtering and ordering.
- Objective Mapping:
  - The aim of printing all common subsequences in descending order of length was successfully achieved and validated.

7. SUMMARY

This lab report demonstrated the process of identifying and printing all common subsequences between two given strings using Java. By generating all subsequences recursively and comparing them using set operations, we achieved accurate results even in the case of no common subsequences due to case sensitivity. The experiment enhanced our problem-solving skills and understanding of recursion, string manipulation, and data structure usage in Java. It also emphasized performance considerations in combinatorial problems.