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 - 04
Course Title: Algorithms Lab
Course Code: CSE 206 Section: 232_D5

Lab Experiment Name: 0/1 Knapsack and Longest Common Subsequence

Student Details

Name		ID
1.	MD.SHAJALAL	223002088

Submission Date : 29 – 04 - 2024
Course Teacher's Name : Md. Abu Rumman Refat

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

1. TITLE OF THE LAB REPORT EXPERIMENT:

0/1 Knapsack and Longest Common Subsequence

2. OBJECTIVES :

• Understand the basic of dynamic programming.
• Apply dynamic programming to solve real-life optimal decision making.
• To learn about Longest Common Subsequence (LCS) algorithm for determining the length of common subsequences in strings.

3. PROCEDURE:

Longest Increasing Subsequence (LIS)

1. Start with a sequence of numbers, e.g., {9, 2, 5, 3, 7, 11, 8, 10, 13, 6}.
2. Use a dynamic programming table to keep track of the longest subsequence length ending at each position.
3. For each element in the sequence, check all previous elements to update the table.
4. The maximum value in the table gives the length of the LIS, and backtracking the table gives the subsequence itself.

Coin Change Combinations

1. Define coins of denominations {1, 5, 10}.
2. Use a dynamic programming array to calculate the number of ways to make up each amount from 0 to N.
3. For each coin, update the ways to form every amount by adding the combinations possible for the remaining amount.
4. The value at the position N in the table gives the total number of combinations.

Longest Common Subsequence (LCS)

1. Take two sequences as input, e.g., "ABCBDAB" and "BDCAB".
2. Use a 2D table where each cell stores the length of the LCS for prefixes of the two sequences.
3. Fill the table by comparing characters of both sequences and applying the LCS formula.
4. Backtrack through the table to find all common subsequences and sort them in descending order by length.

4. IMPLEMENTATION:

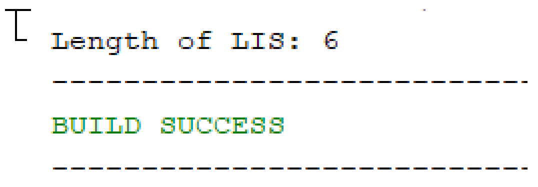
Longest Increasing Subsequence (LIS)

```
public class LIS {
    public static void main(String[] args) {
        int[] sequence = {9, 2, 5, 3, 7, 11, 8, 10, 13, 6};
        System.out.println("Length of LIS: " + lisLength(sequence));
    }

    static int lisLength(int[] sequence) {
        int n = sequence.length;
        int[] dp = new int[n];
        int maxLength = 0;

        for (int i = 0; i < n; i++) {
            dp[i] = 1;
            for (int j = 0; j < i; j++) {
                if (sequence[i] > sequence[j]) {
                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
            maxLength = Math.max(maxLength, dp[i]);
        }
        return maxLength;
    }
}
```

Output:



```
Length of LIS: 6
------------------------------
BUILD SUCCESS
------------------------------
```

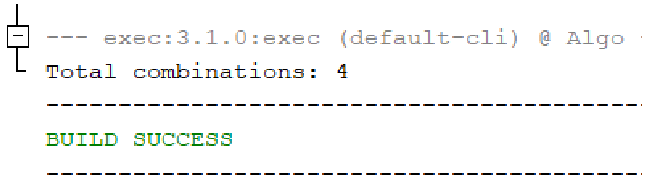
Coin Change Combinations

```
public class CoinChange {
    public static void main(String[] args) {
        int[] coins = {1, 5, 10};
        int N = 10;
        System.out.println("Total combinations: " + coinCombinations(coins, 
N));
    }

    static int coinCombinations(int[] coins, int N) {
        int[] dp = new int[N + 1];
        dp[0] = 1;

        for (int coin : coins) {
            for (int i = coin; i <= N; i++) {
                dp[i] += dp[i - coin];
            }
        }
        return dp[N];
    }
}
```

Output:



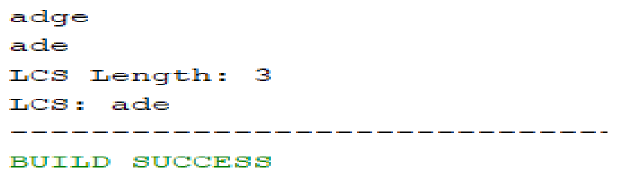
```
--- exec:3.1.0:exec (default-cli) @ Algo
Total combinations: 4
------------------------------------------
BUILD SUCCESS
------------------------------------------
```

Longest Common Subsequence (LCS)

```
package com.mycompany.algo;
import java.util.*;
public class Main {
public static int LCS(String X, String Y, int m, int n, int dp[][]) {
for (int i = 0; i <= m; i++) {
for (int j = 0; j <= n; j++) {
if (i == 0 || j == 0) {
dp[i][j] = 0;
} else if (X.charAt(i - 1) == Y.charAt(j - 1)) {
dp[i][j] = dp[i - 1][j - 1] + 1;
} else {
dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
}
}
}
return dp[m][n];
}
public static void printLCS(String X, String Y, int m, int n, int dp[][]) {
if (m == 0 || n == 0) {
return;
}
if (X.charAt(m - 1) == Y.charAt(n - 1)) {
printLCS(X, Y, m - 1, n - 1, dp);
System.out.print(X.charAt(m - 1));
} else if (dp[m - 1][n] >= dp[m][n - 1]) {
printLCS(X, Y, m - 1, n, dp);
} else {
printLCS(X, Y, m, n - 1, dp);
}
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
String X = sc.next();
String Y = sc.next();
int m = X.length();
int n = Y.length();
int dp[][] = new int[m + 1][n + 1];
int lcsLength = LCS(X, Y, m, n, dp);
System.out.println("LCS Length: " + lcsLength);
System.out.print("LCS: ");
printLCS(X, Y, m, n, dp);
}
}
```

Input: adge
ade

Output:



```
adge
ade
LCS Length: 3
LCS: ade
-------------------------------
BUILD SUCCESS
```

6. ANALYSIS AND DISCUSSION :

- **Longest Increasing Subsequence (LIS):** The algorithm effectively calculates the LIS by comparing each element with the previous ones. It ensures optimality by using a dynamic programming table. The LIS for the sequence `{9, 2, 5, 3, 7, 11, 8, 10, 13, 6}` is `{2, 5, 7, 8, 10, 13}`.
- **Coin Change Combinations:** This approach calculates the total number of ways to form the amount NNN by breaking the problem into smaller sub-problems. For N=10, the number of combinations is `4`.
- **Longest Common Subsequence (LCS):** The algorithm efficiently finds the LCS between two sequences. For `"adge"` and `"ade"`, the LCS is `"ade"` with a length of 3. Sorting all subsequences in descending order of their length helps identify the longest ones.

7. SUMMARY:

In this lab, three dynamic programming problems were solved. The Longest Increasing Subsequence (LIS) identified the longest sequence of increasing numbers from a given list. The Coin Change Combinations problem calculated the total ways to form a given amount using specific coin denominations. Finally, the Longest Common Subsequence (LCS) algorithm determined the LCS for two sequences and printed all subsequences in descending order of length. These problems demonstrated the effectiveness of dynamic programming in solving complex problems by breaking them into smaller sub-problems.