VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore ▪ Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

Name:	Saad Wasi Ahmed Siddiqui	**Slot:**	L13+L14
Professor:	Dr. Indira B	**Course Code:**	BCSE204P
Class Nbr:	CH2025260101604	**Semester:**	Fall Semester 2025-26
Reg. Nbr:	23BRS1113	**Course Title:**	Design and Analysis of Algorithms Lab

Lab Assessment 4 - Dynamic Programming

Aim

The aim of this program is to find the **Longest Common Subsequence (LCS)** between two given strings using the **Dynamic Programming** approach. The program will output:

1. The length of the LCS.
2. The LCS string itself.

Algorithm

1. Read two strings X and Y.
2. Let `m` = length of X, n = length of Y.
3. Create a 2D integer array `dp[m+1][n+1]` to store lengths of LCS for substrings.
4. Initialize the first row and column to 0.
5. For each `i` from 1 to m:
    - For each `j` from 1 to n:
        - If `X[i-1] == Y[j-1]`, set `dp[i][j] = dp[i-1][j-1] + 1`.

        - Else, set `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`.
6. `dp[m][n]` will contain the length of the LCS.
7. Trace back from `dp[m][n]` to build the LCS string:
    - If `X[i-1] == Y[j-1]`, add it to LCS and move diagonally.
    - Else move in the direction of the larger value.
8. Reverse the built string (because it is constructed backwards).
9. Output the LCS length and string.

Code

```
#include <stdio.h>
#include <string.h>

#define MAX 1000

int max(int a, int b) {
    if(a>b)
    return a;
    else
    return b;
}

void lcs(char X[], char Y[]) {
    int m = strlen(X);
    int n = strlen(Y);
    int dp[MAX][MAX];


    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[i - 1] == Y[j - 1])
```

```
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }


    int len = dp[m][n];
    char lcs_str[len + 1];
    lcs_str[len] = '\0';


    int i = m, j = n, index = len - 1;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs_str[index] = X[i - 1];
            i--;
            j--;
            index--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }

    printf("Length of LCS: %d\n", len);
    printf("LCS string: %s\n", lcs_str);
}

int main() {
    char X[MAX], Y[MAX];

    printf("Enter first string: ");
    scanf("%s", X);

    printf("Enter second string: ");
    scanf("%s", Y);

    lcs(X, Y);
```

```
    return 0;
}
```

Sample Input Output

```
Enter first string: designandanalysisofalgorithm
Enter second string: signalog
Length of LCS: 8
LCS string: signalog
```

Time Complexity Analysis

- **DP table filling:**
  There are `m` rows and n columns, and each cell computation takes O(1) time.
  → O(m × n)

- **LCS reconstruction:**
  At most `m + n` steps in the traceback.
  → O(m + n)

Overall Time Complexity: O(m × n)
Space Complexity: O(m × n) (can be reduced to O(min(m, n)) if only length is needed).