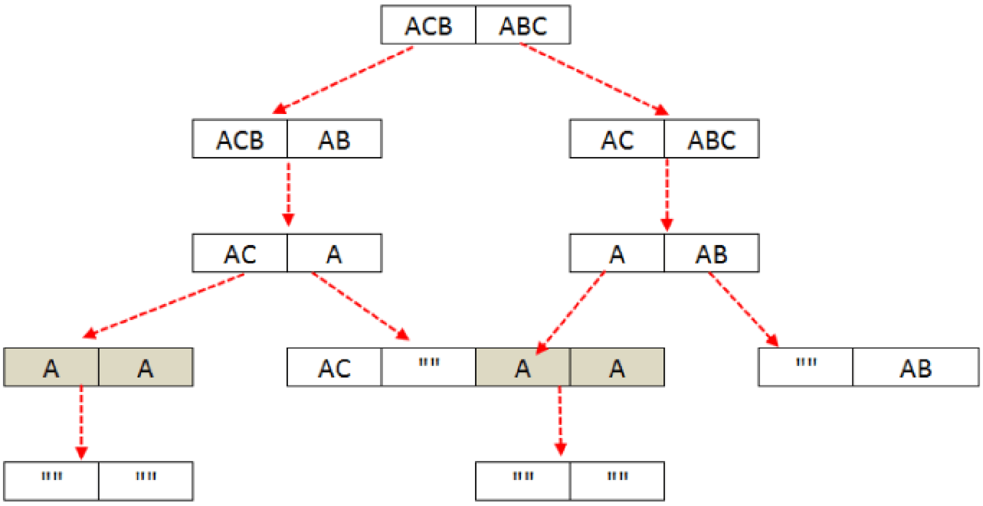
Experiment No : 7

Name: Shawn Louis

Roll No: 31

Batch: B

Problem Statement:	**To implement Longest Common Subsequence Algorithm** 1) Using Dynamic Programming 2) Show the length of the longest subsequence and also print the subsequence.
Objective:	• To be able to implement a problem using dynamic programming
Expected Outcome:	• **Ability to understand a given problem statement and build logic as per dynamic programming.** • **Ability to write effficient code.**
Theory:	**Subsequence:** Let us consider a sequence S = <s_1, s_2, s_3, s_4, …,s_n>. A sequence Z = <z_1, z_2, z_3, z_4, …,z_m> over S is called a subsequence of S, if and only if it can be derived from S deletion of some elements. **Common Subsequence:** Suppose, ***X*** and ***Y*** are two sequences over a finite set of elements. We can say that ***Z*** is a common subsequence of ***X*** and ***Y***, if ***Z*** is a subsequence of both ***X*** and ***Y***. **Longest Common Subsequence:** If a set of sequences are given, the longest common subsequence problem is to find a common subsequence of all the sequences that is of maximal length. **Problem Statement:** Given two string sequences, write an algorithm to find the length of longest subsequence present in both of them. The longest common subsequence problem is finding the longest sequence which exists in both the given strings.



Algorithm:

		A	B	C	D	A
	0	0	0	0	0	0
A	0	$\boxed{1}$	1	1	1	1
C	0	1	1	$\boxed{2}$	2	2
B	0	1	2	2	2	2
D	0	1	2	2	$\boxed{3}$	3
E	0	1	2	2	3	3
A	0	1	2	2	3	$\boxed{4}$

LCS - "ACDA"

Algorithm: LCS-Length-Table-Formulation (X, Y)

```
m := length(X)
n := length(Y)
for i = 1 to m do
  C[i, 0] := 0
for j = 1 to n do
  C[0, j] := 0
for i = 1 to m do
  for j = 1 to n do
    if xi = yj
      C[i, j] := C[i - 1, j - 1] + 1
      B[i, j] := 'D'
    else
      if C[i -1, j] ≥ C[i, j -1]
        C[i, j] := C[i - 1, j] + 1
        B[i, j] := 'U'
      else
      C[i, j] := C[i, j - 1]
      B[i, j] := 'L'
return C and B
```

Algorithm: Print-LCS (B, X, i, j)

```
if i = 0 and j = 0
  return
if B[i, j] = 'D'
  Print-LCS(B, X, i-1, j-1)
  Print(xi)
else if B[i, j] = 'U'
  Print-LCS(B, X, i-1, j)
else
  Print-LCS(B, X, i, j-1)
```

Program Code:

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

int max(int a, int b);

char *s1, *s2;  //for dynamic alloc
int i, j, m, n, length;
int table[10][10] = {0};
```

```
char sequence[10];

void LCS();
void print_table();
void sequenceFinder();

int main()
{
  clrscr();

  //Getting first string
  printf("\nEnter size of string 1 : ");
  scanf("%d", &m);

  //dynamic allocation of memory for string
  s1 = (char*)malloc((m+1) * sizeof(char));

  printf("Enter string 1: ");
  scanf("%s", s1);

  //Getting second string
  printf("\nEnter size of string 2 : ");
  scanf("%d", &n);

  //dynamic allocation of memory for string
  s2 = (char*)malloc((n+1) * sizeof(char));

  printf("Enter string 2: ");
  scanf("%s", s2);

  //Displaying both strings
  printf("\nString 1 : %s\n", s1);
  printf("String 2 : %s\n", s2);

  //Calling to dynamically programmed LCS function
  LCS();
  //printing the DP table
  print_table();

  /*Length of longest subsequence is stored
     in the last element of table*/
  length = table[m][n];
  printf("\nLength of LCS is %d", length);

  if(length == 0)
    printf("\nNo LCS found");
  else
  {
    //traces the longest common subsequence
```

```
    sequenceFinder();
    printf("\nThe LCS       : %s", sequence);
  }

  getch();
  return 0;
}

void LCS()
{
  for (i=0; i<=m; i++)
    for (j=0; j<=n; j++)
    {
      if (i == 0 || j == 0)
       table[i][j] = 0;

      //when char of both strings match
      else if (s1[i-1] == s2[j-1])
       //assign incremented value of upper-left diagonal elem
        table[i][j] = table[i-1][j-1] + 1;

         //if char of both strings dont match
      else
       //assign max value of upper or left element
        //Ternary operator used to assign max
       table[i][j] = (table[i-1][j] > table[i][j-1]) ?
table[i-1][j] : table[i][j-1];
    }

}

void sequenceFinder()
{
  sequence[length] = '\0';
  i = m;
  j = n;
  while (i > 0 && j > 0)

      // If current character in s1 and s2 are same
      // current character is part of LCS
      if (s1[i-1] == s2[j-1])
      {
        // Put current character in result
        sequence[length-1] = s1[i-1];
       i--; j--; length--;
      }

      // If not same, then find the larger of two and
      // go in the direction of larger value
```

```
      else if (table[i-1][j] > table[i][j-1])
        i--;

      else
        j--;
}

void print_table()
{
  printf("\n\n");
  printf("DP table : \n");
  for(i = 0; i < m+1; i++)
    for(j = 0; j < n+1; j++)
    {
      printf("%d\t", table[i][j]);
      if((j+1) % (n+1) == 0)
                    printf("\n");
    }
}
```

Output Snapshot:

```
Enter size of string 1 : 2
Enter string 1         : bd

Enter size of string 2 : 4
Enter string 2         : abcd

String 1 : bd
String 2 : abcd


DP table :
0       0       0       0       0
0       0       1       1       1
0       0       1       1       2

Length of LCS : 2
The LCS       : bd
```

```
Enter size of string 1 : 5
Enter string 1         : stone

Enter size of string 2 : 7
Enter string 2         : longest

String 1 : stone
String 2 : longest


DP table :
0       0       0       0       0       0       0       0
0       0       0       0       0       0       1       1
0       0       0       0       0       0       1       2
0       0       1       1       1       1       1       2
0       0       1       2       2       2       2       2
0       0       1       2       2       3       3       3

Length of LCS : 3
The LCS       : one
```

```
Enter size of string 1 : 2
Enter string 1         : df

Enter size of string 2 : 5
Enter string 2         : yuopo

String 1 : df
String 2 : yuopo


DP table :
0       0       0       0       0       0
0       0       0       0       0       0
0       0       0       0       0       0

Length of LCS : 0
No LCS found_
```

Application	The longest common subsequence problem is a classic computer science problem, the basis of data comparison programs such as the diff-utility, and has applications in bioinformatics. It is also widely used by revision control systems, such as SVN and Git, for reconciling multiple changes made to a revision-controlled collection of files.
Outcome:	Successfully implemented Least Common Subsequence Problem using Dynamic Programming in C.