Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058, India
(Autonomous College Affiliated to University of Mumbai)

Experiment No.	5
Aim	To implement dynamic algorithms
Name	Harsh Mukesh Jain
UID No.	2021300048
Class & Division	SE Comps A (C – Batch)
Date of Performance	20-02-2023
Date of Submission	27-02-2023

Theory/Experiment:

Dynamic Programming :

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

The approach of solving problems using dynamic programming algorithm has following steps :

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Longest Common Subsequence :

A longest common subsequence (LCS) is the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from the longest common substring: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences. The problem of computing longest common subsequences is a classic computer science problem, the basis of data comparison programs such as the diff utility, and has applications in computational

linguistics and bioinformatics. It is also widely used by revision control systems such as Git for reconciling multiple changes made to a revision-controlled collection of files.

For example, consider the sequences (ABCD) and (ACBAD). They have 5 length-2 common subsequences: (AB), (AC), (AD), (BD), and (CD); 2 length-3 common subsequences: (ABD) and (ACD); and no longer common subsequences. So (ABD) and (ACD) are their longest common subsequences.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \quad \text{or} \quad j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \quad \text{and} \quad x_i = y_i \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_i \end{cases}$$

For a given string x = (1,0,0,1,0,1,0,1) and y = (0,1,0,1,1,0,1,1,0), determine LCS

x =(1,0,0,1,0,1,0,1) y =(0,1,0,1,1,0,1,1,0)

i	j	0	1	2	3	4	5	6	7	8	9
		y_j	0	1	0	1	1	0	1	1	0
0	X_i	0	0	0	0	0	0	0	0	0	0
1	1	0	↑ 0	↖ (1)	↑ 1	↖ 1	← 1	↖ 1	↖ 0	↖ 1	← 1
2	0	0	↖ 1	↑ 1	↖ (2)	← 2	← 2	↖ 2	← 2	← 2	↖ 2
3	0	0	↖ 1	↑ 1	↖ 2	↑ 2	↑ 2	↖ (3)	← 3	← 3	↖ 3
4	1	0	↑ 1	↖ 2	↑ 2	↖ 3	↖ 3	↑ 3	↖ (4)	↖ 4	← 4
5	0	0	↖ 1	↑ 2	↖ 3	↑ 3	↑ 3	↖ 4	↑ 4	↑ 4	↖ 5
6	1	0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 4	↑ 4	↖ 5	↖ (5)	↑ 5
7	0	0	↖ 1	↑ 2	↖ 3	↑ 4	↑ 4	↖ 5	↑ 5	↑ 5	↖ (6)
8	1	0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 5	↑ 5	↖ 6	↖ 6	↑ 6

Algorithm :

```
LCS-LENGTH(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
                                then c[i, j] = c[i-1, j-1] + 1
                                b[i, j] = ARROW_CORNER
                                else if c[i-1, j] >= c[i, j-1]
                                then c[i, j]=c[i-1, j]
                                b[i, j] = ARROW_UP
                                else c[i, j]=c[i, j-1]
                                b[i, j] = ARROW_LEFT
return c and b

PRINT-LCS(b, X, i, j)
if i=0 or j=0
        then return
if b[i, j] == ARROW_CORNER
        then PRINT-LCS(b, X, i-1, j-1)
        print Xi
elseif b[i, j] == ARROW_UP
        then PRINT-LCS(b, X, i-1, j)
else PRINT-LCS(b, X, i, j-1)
```

Code :

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

int main()
{
        char str1[20], str2[20];
        printf("Enter 1st string:");
        scanf("%s",str1);
        for(int i = strlen(str1); i >= 1; i--) {
           str1[i] = str1[i-1];
        }
        str1[0] = 'x';

        printf("\nEnter 2nd string:");
        scanf("%s",str2);
        for(int i = strlen(str2); i >= 1; i--) {
           str2[i] = str2[i-1];
        }
        str2[0] = 'y';
   int rows = strlen(str1);
   int columns = strlen(str2);
   int dp[rows][columns], direction[rows][columns];
   clock_t start, end;
   double cpu_time_used;
   start = clock();
   // 0 for diagonal arrow, 1 for up and -1 for left arrow
   for(int i = 0 ; i < rows ; i++) {
      for(int j = 0 ; j < columns; j++) {
         dp[i][j] = 0;
      }
   }
   for(int i = 1; i < rows; i++) {
      for(int j = 1; j < columns; j++) {
```

```
            if(str1[i] == str2[j]) {
                dp[i][j] = dp[i-1][j-1] + 1;
                direction[i][j] = 0;

            }
            else if(str1[i] != str2[j]) {
                if(dp[i-1][j] >= dp[i][j-1]) {
                    dp[i][j] = dp[i-1][j];
                    direction[i][j] = 1;
                }
                else {
                    dp[i][j] = dp[i][j-1];
                    direction[i][j] = -1;
                }
            }
        }
    }
    printf("\nAuxillary Table:\n");
    for(int i = 0 ; i < rows ; i++) {
        for(int j = 0 ; j < columns; j++) {
            printf("%d\t",dp[i][j]);
        }
        printf("\n");
    }
    printf("\nArrow Direction Table:\n0 represents diagonal arrow\n1 represents upward\n-
1 represents left direction\n");
    for(int i = 1 ; i < rows ; i++) {
        for(int j = 1 ; j < columns; j++) {
            printf("%d\t",direction[i][j]);
        }
        printf("\n");
    }
    // int ans[(rows < columns)? rows - 1 : columns - 1], k = -1;
    int ans[10], k = -1;
    for(int i = rows - 1 ; i >= 1 ; ) {
        for(int j = columns - 1 ; j >= 1; ) {
            // diagonal arrow
```

```
            if(direction[i][j] == 0) {
                ans[++k] = i;
                i -= 1;
                j -= 1;
            }
            // Up
            else if(direction[i][j] == 1) {
                i--;
            }
            // Left arrow
            else {
                j--;
            }
            if(i < 1 || j < 1) break;
            // printf("\n%d\t%d\t%d\n",i,j,direction[i][j]);
        }
        printf("\n");
    }
    printf("Longest Common Subsequence from X and Y is : ");
    for(int i = k ; i >= 0; i--) {
        printf("%c",str1[ans[i]]);
    }
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("\n\nTime taken for calculating longest common subsequence is: %f
s",cpu_time_used);
    return 0;
}
```

Output :

```
Command Prompt
D:\Desktop>gcc LCS.c

D:\Desktop>a
Enter 1st string:10010101

Enter 2nd string:010110110

Auxillary Table:
0       0       0       0       0       0       0       0       0       0
0       0       1       1       1       1       1       1       1       1
0       1       1       2       2       2       2       2       2       2
0       1       1       2       2       2       3       3       3       3
0       1       2       2       3       3       3       4       4       4
0       1       2       3       3       3       4       4       4       5
0       1       2       3       4       4       4       5       5       5
0       1       2       3       4       4       5       5       5       6
0       1       2       3       4       5       5       6       6       6       6

Arrow Direction Table:
0 represents diagonal arrow
1 represents upward
-1 represents left direction
1       0       -1      0       0       -1      0       0       -1      -1
0       1       0       -1      -1      0       -1      -1      0       -1
0       1       0       1       1       0       -1      -1      0       -1
1       0       1       0       0       1       0       0       -1      -1
0       1       0       1       1       0       1       1       0       -1
1       0       1       0       0       1       0       0       1       1
0       1       0       1       1       0       1       1       0       -1
1       0       1       0       0       1       0       0       1       1

Longest Common Subsequence from X and Y is : 100110

Time taken for calculating longest common subsequence is: 0.045000 s
D:\Desktop>
```

From the table we can deduct that LCS = 6. There are several such sequences, for instance (1,0,0,1,1,0) (0,1,0,1,0,1) and (0,0,1,1,0,1).

Time Complexity: O(m * n) where m and n are the string lengths.
Auxiliary Space: O(m * n) here the recursive stack space is ignored.

Conclusion :

By performing this experiment, I understood the concept of Longest Common Subsequence which is used to find common and longest subsequence between 2 or more sequences. LCS helps in reducing time for comparisons, reduces the required space i.e if 2 strings of length m and n require (m x n) matrix for dp, reduce cache misses.