Name: Chirag Rana
Class: SE Comps
Batch: C
UID: 2018130043

EXPERIMENT – 7

AIM: To perform experiments based on Longest Common Subsequence(C, B) using Dynamic Programing.

THEORY:

The longest common subsequence problem is finding the longest sequence which exists in both the given strings.

Subsequence

Let us consider a sequence $S = <s_1, s_2, s_3, s_4, \ldots, s_n>$.

A sequence $Z = <z_1, z_2, z_3, z_4, \ldots, z_m>$ 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.

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.

Naïve Method

Let ***X*** be a sequence of length ***m*** and ***Y*** a sequence of length ***n***. Check for every subsequence of ***X*** whether it is a subsequence of ***Y***, and return the longest common subsequence found.

There are 2^m subsequences of ***X***. Testing sequences whether or not it is a subsequence of ***Y*** takes ***O(n)*** time. Thus, the naïve algorithm would take $O(n2^m)$ time.

Dynamic Programming

Let $X = < x_1, x_2, x_3, \ldots, x_m >$ and $Y = < y_1, y_2, y_3, \ldots, y_n >$ be the sequences. To compute the length of an element the following algorithm is used.

In this procedure, table ***C[m, n]*** is computed in row major order and another table ***B[m,n]*** is computed to construct optimal solution.

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 x_i = y_j
         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
```

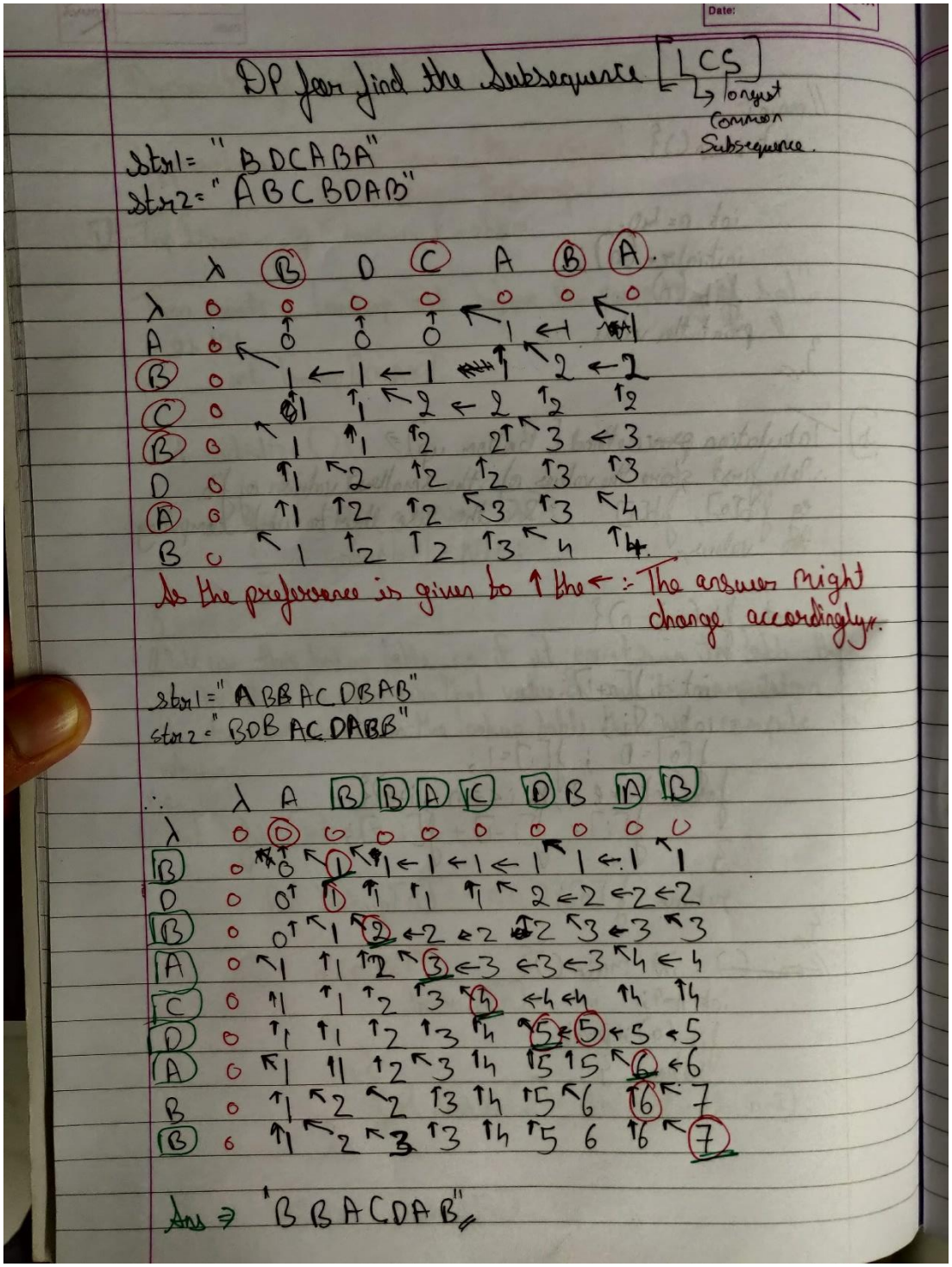
Algorithm: Print-LCS (B, X, i, j)

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

TIME COMPLEXITY:

To populate the table, the outer **for** loop iterates ***m*** times and the inner **for** loop iterates ***n*** times. Hence, the complexity of the algorithm is *O(m, n)*, where ***m*** and ***n*** are the length of two strings.

DESIGN AND IMPLEMENTATION:



CODE:

```
#include<stdio.h>
#include<stdlib.h>
void LCS(char arr1[],int len1,char arr2[],int len2);
void printLCS(char arr1[],int row,int col,int countarr[][col],int arrowarr[][col]);

int main(){
    int len1,len2;
```

```
    char arr1[100];
    char arr2[100];

    printf("Enter the length of the 2 Strings:\n ");
    scanf("%d",&len1);
    scanf("%d",&len2);
    printf("Enter the two Strings:\n");
    scanf("%s",&arr1);
    scanf("%s",&arr2);

    LCS(arr1,len1,arr2,len2);

    return 0;
}
void LCS(char arr1[],int len1,char arr2[],int len2){
    int countarr[50][50];
    int arrowarr[50][50];
    for (int i = 0; i <= len1; i++){
        for (int j = 0; j <= len2; j++){
            countarr[i][j] = 0;
            arrowarr[i][j] = 0;
        }
    }

    for (int i = 1; i <= len1; i++){
        for (int j = 1; j <= len2; j++){
            if (arr1[i-1] == arr2[j-1]){
                countarr[i][j] = countarr[i-1][j-1] + 1;
                arrowarr[i][j] = 0;
            }
            else if( countarr[i-1][j] >= countarr[i][j-1]){
                countarr[i][j] = countarr[i-1][j];
                arrowarr[i][j] = 1;
            }else{
                countarr[i][j] = countarr[i][j-1];
                arrowarr[i][j] = 2;
            }
        }
    }

    printf("The Matrix of the Counting values is:\n");

    for(int i = 0; i <= len1; i++){
        for (int j = 0; j <= len2; j++){
            printf("  %d  ",countarr[i][j]);
        }
        printf("\n");
    }

    printf("The Matrix of the Arrow values is:\n");
```

```
    for (int i = 0; i <= len1; i++){
        for (int j = 0; j <= len2; j++){
            printf("  %d  ",arrowarr[i][j]);
        }
        printf("\n");
    }

    printLCS(arr1,len1+1,len2+1,countarr,arrowarr);
    return ;
}

void printLCS(char arr1[],int row,int col,int countarr[][50], int arrowarr[][50]){
    if (row == 0 || col == 0) {return;}
    if( arrowarr[row][col] == 0){
        printLCS(arr1,row-1,col-1,countarr,arrowarr);
        printf(" %c ",arr1[row-1]);
    }else if (arrowarr[row][col] == 1){
        printLCS(arr1,row-1,col,countarr,arrowarr);
    }else{
        printLCS(arr1,row,col-1,countarr,arrowarr);
    }
    return ;
}
```

OUTPUT:

```
Enter the length of the 2 Strings:
 6
7
Enter the two Strings:
BDCABA
ABCBDAB
The Matrix of the Counting values is:
  0    0    0    0    0    0    0    0
  0    0    1    1    1    1    1    1
  0    0    1    1    1    2    2    2
  0    0    1    2    2    2    2    2
  0    1    1    2    2    2    3    3
  0    1    2    2    3    3    3    4
  0    1    2    2    3    3    4    4
The Matrix of the Arrow values is:
  0    0    0    0    0    0    0    0
  0    1    0    2    0    2    2    0
  0    1    1    1    1    0    2    2
  0    1    1    0    2    1    1    1
  0    0    1    1    1    1    0    2
  0    1    0    1    0    2    1    0
  0    0    1    1    1    1    0    1
 B  D  A  B
Process returned 0 (0x0)   execution time : 23.150 s
Press any key to continue.
```

```
9
Enter the two Strings:
BDBACDABB
ABBACDBAB
The Matrix of the Counting values is:
  0    0    0    0    0    0    0    0    0    0
  0    0    1    1    1    1    1    1    1    1
  0    0    1    1    1    1    2    2    2    2
  0    0    1    2    2    2    2    3    3    3
  0    1    1    2    3    3    3    3    4    4
  0    1    1    2    3    4    4    4    4    4
  0    1    1    2    3    4    5    5    5    5
  0    1    1    2    3    4    5    5    6    6
  0    1    2    2    3    4    5    6    6    7
  0    1    2    3    3    4    5    6    6    7
The Matrix of the Arrow values is:
  0    0    0    0    0    0    0    0    0    0
  0    1    0    0    2    2    2    0    2    0
  0    1    1    1    1    1    0    2    2    2
  0    1    0    0    2    2    1    0    2    0
  0    0    1    1    0    2    2    1    0    2
  0    1    1    1    1    0    2    2    1    1
  0    1    1    1    1    1    0    2    2    2
  0    0    1    1    0    1    1    1    0    2
  0    1    0    0    1    1    1    0    1    0
  0    1    0    0    1    1    1    0    1    0
 B  B  A  C  D  A  B
Process returned 0 (0x0)   execution time : 19.299 s
Press any key to continue.
```

CONCLUSION:

1. I implemented Longest Common Subsequence and understood the feature of dynamic Programming.