Name: Chirag Rana

Class: SE Comps

Batch: C

UID: 2018130043

# **EXPERIMENT – 5**

**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 = <s1, s2, s3, s4, …,sn>.

A sequence Z = <z1, z2, z3, z4, …,zm> 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 ***2m*** 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(n2m)*** time.

## **Dynamic Programming**

Let *X = < x1, x2, x3,…, xm >* and *Y = < y1, y2, y3,…, yn >* 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 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)

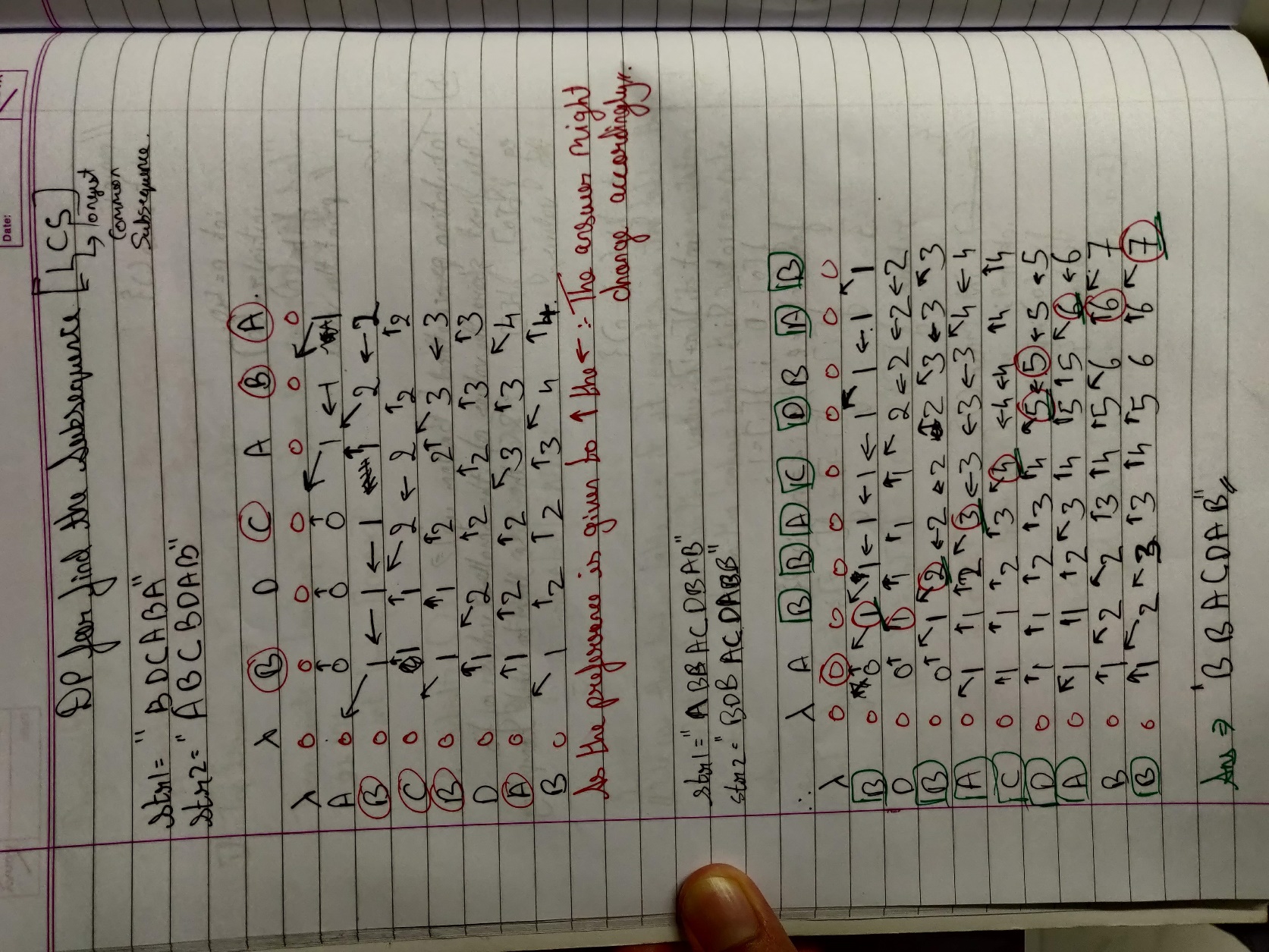
else

Print-LCS(B, X, i, j-1)

**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{

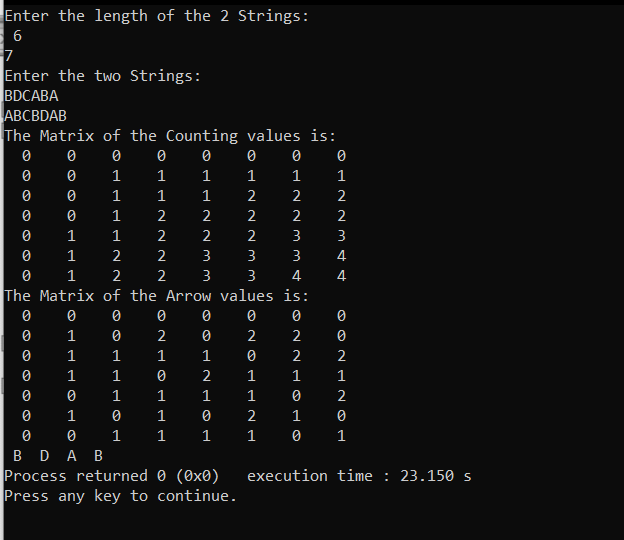
        printLCS(arr1,row,col-1,countarr,arrowarr);

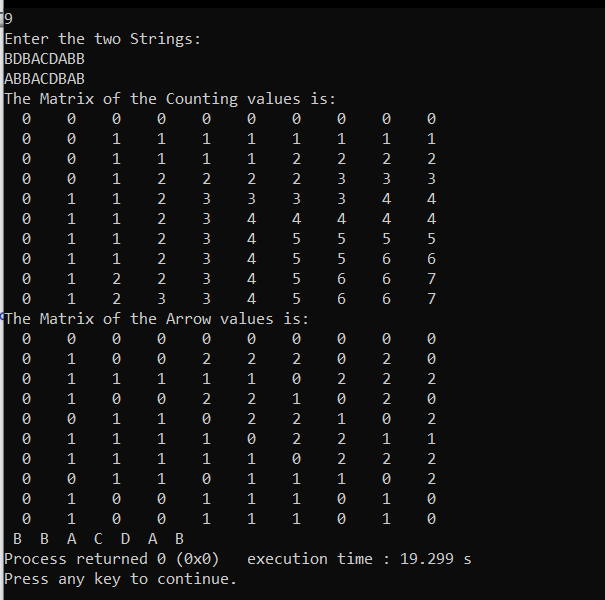
    }

    return ;

}

**OUTPUT:**





**CONCLUSION:**

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