Batch: C2	**Roll No.: 16010122257**
Experiment No. 10	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Longest Common Subsequence String Matching Algorithm

Objective: To compute longest common subsequence for the given two strings.

CO to be achieved:

CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyze and solve problems for different string matching algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:
Given 2 sequences, $X = x1 , ..., xm$ and $Y = y1 , ... , yn$, find a subsequence common to both whose length is longest. A subsequence doesn't have to be consecutive, but it has to be in order.

New Concepts to be learned:
String matching algorithm, Dynamic programming approach for LCS, Applications of LCS.

Recursive **Formulation:**

Define $c[i, j]$ = length of LCS of X_i and Y_j.
Final answer will be computed with $c[m, n]$.

$c[i, j] = 0$
if $i=0$ or $j=0$.
$c[i, j] = c[i - 1, j - 1] + 1$
if $i,j>0$ and $x_i=y_j$

$c[i, j] = \max(c[i - 1, j], c[i, j - 1])$
if $i, j > 0$ and $x_i <> y_j$

Algorithm: Longest Common Subsequence

Compute length of optimal solution-

```
LCS-LENGTH (X, Y, m, n)
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 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] ← "≈"
               else if c[i − 1, j] ≥ c[i, j − 1]
                    then c[i, j] ← c[i − 1, j]
                         b[i, j] ← "↑"
                    else c[i, j] ← c[i, j − 1]
                         b[i, j] ← "←"
return c and b
```

Print the solution-

```
PRINT-LCS(b, X, i, j)
  if i = 0 or j = 0
     then return
```

if $b[i, j]$ = "≈"

then PRINT-LCS$(b, X, i-1, j-1)$

print x_i

elseif $b[i, j]$ = "↑"

then PRINT-LCS$(b, X, i-1, j)$

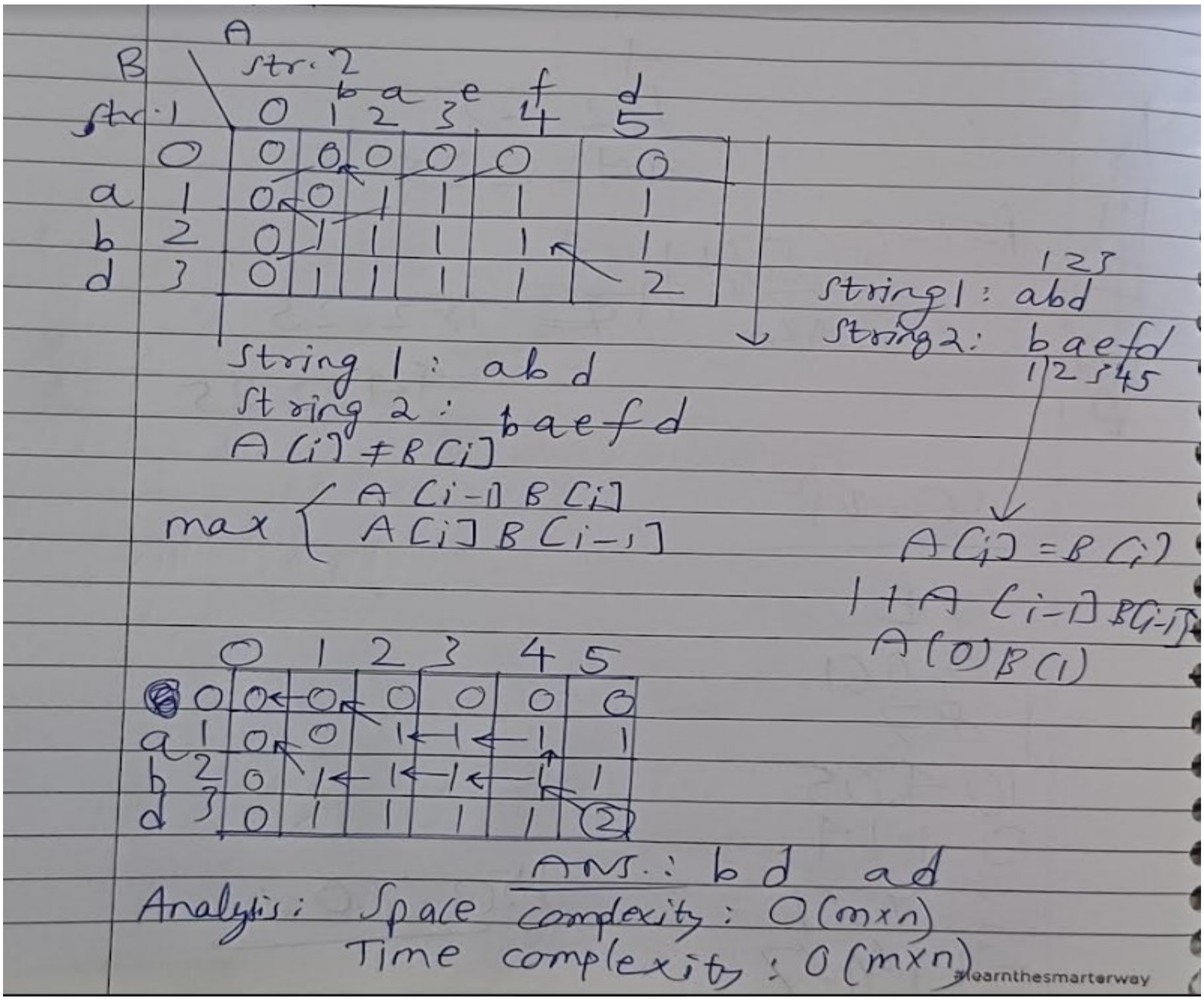
else PRINT-LCS$(b, X, i, j-1)$

Initial call is PRINT-LCS(b, X, m, n).
$b[i, j]$ points to table entry whose subproblem we used in solving LCS of X_i and Y_j.

When $b[i, j]$ = ≈, we have extended LCS by one character. So longest com- mon subsequence = entries with ≈ in them.

Example: LCS computation

Analysis of LCS computation



CODE:

Start here X LCS.cpp X

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main() {
    string str1, str2;
    cout << "Enter the first string: ";
    cin >> str1;
    cout << "Enter the second string: ";
    cin >> str2;

    vector<char> output;
    vector<vector<int>> arr(str1.length() + 1, vector<int>(str2.length() + 1, 0));

    for (int i = 0; i < str1.length(); i++) {
        for (int j = 0; j < str2.length(); j++) {
            if (str1[i] == str2[j]) {
                arr[i + 1][j + 1] = arr[i][j] + 1;
            } else {
                if (arr[i][j + 1] <= arr[i + 1][j]) {
                    arr[i + 1][j + 1] = arr[i + 1][j];
                }
                if (arr[i][j + 1] > arr[i + 1][j]) {
                    arr[i + 1][j + 1] = arr[i][j + 1];
                }
            }
        }
    }

    int p = str1.length() - 1;
    int q = str2.length() - 1;

    while (p >= 0 && q >= 0) {
        if (str1[p] == str2[q]) {
            output.push_back(str1[p]);
            p--;
            q--;
        } else {
            if (arr[p][q - 1] <= arr[p - 1][q]) {
                p--;
            } else if (arr[p][q - 1] > arr[p - 1][q]) {
```

```cpp
                q--;
            }
        }
    }

    // Printing the LCS array
    for (int i = 0; i < arr.size(); i++) {
        for (int j = 0; j < arr[i].size(); j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    // Printing the common string
    cout << "Common string is: ";
    for (int j = output.size() - 1; j >= 0; j--) {
        cout << output[j];
    }
    cout << endl;

    return 0;
}
```

Output:

```
Enter the first string: abcdefa
Enter the second string: acd
0 0 0 0
0 1 1 1
0 1 1 1
0 1 2 2
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
Common string is: acd
```

```
C:\Users\DELL\Desktop\LCS.exe
Enter the first string: abcdefa
Enter the second string: tacd
0 0 0 0 0
0 0 1 1 1
0 0 1 1 1
0 0 1 2 2
0 0 1 2 3
0 0 1 2 3
0 0 1 2 3
0 0 1 2 3
Common string is: acd

Process returned 0 (0x0)   execution time : 17.958 s
Press any key to continue.
```

CONCLUSION:

Thus,we've implemented Longest Common Sub sequence(LCS) using in dynamic programming.We've implemented this algorithm using the table method.It has applications in computational linguistics and bioinformatics.