K. J. Somaiya College of Engineering, Mumbai-77

(A Constituent College of Somaiya Vidyavihar University)

Department of Computer Engineering

Batch: A3	**Roll No.: 16010121045**

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, Saratai 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 = y j
               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 xi
    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 *Xi* 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

Q> STATIONWAGON
INNOVATION

		0	1	2	3	4	5	6	7	8	9	10	11	12
			S	T	A	T	I	O	N	W	A	G	O	N
0		0	0	0	0	0	0	0	0	0	0	0	0	0
1	I	0	0↑	0↑	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←
2	N	0	0↑	0↑	0↑	0↑	1↑	1↑	2↖	2←	2←	2←	2←	2↖
3	N	0	0↑	0↑	0↑	0↑	1↑	1↑	2↖	2←	2←	2←	2←	3↖
4	O	0	0↑	0↑	0↑	0↑	1↑	2↖	2←	2←	2←	2←	3↖	3↑
5	V	0	0↑	0↑	0↑	0↑	1↑	2↑	2←	2←	2←	2←	3↑	3↑
6	A	0	0↑	0↑	1↖	1←	1←	2↑	2←	2←	3↖	3←	3←	3↑
7	T	0	0↑	1↖	1←	2↖	2←	2↑	2↑	2↑	3↑	3↑	3↑	3↑
8	I	0	0↑	1↑	1↑	2↑	3↖	3←	3←	3←	3←	3←	3←	3←
9	O	0	0↑	1↑	1↑	2↑	3↑	4↖	4←	4←	4←	4←	4↖	4←
10	N	0	0↑	1↑	1↑	2↑	3↑	4↑	5↖	5←	5←	5←	5←	5↖

LCS = ATION

Analysis of LCS computation

Time Complexity: O(2 ^ N), i.e., exponential as we generate and compare all the subsequences of both the strings

Note: The total number of subsequences of a string is O(2 ^ N).

Space Complexity: O(1) as no extra space is being used.

Where 'N' is the length of the shortest of the two strings.

Code:

```
#include <bits/stdc++.h>
using namespace std;

string lcs(string X, string Y, int m, int n)
{
    int L[m + 1][n + 1];
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    int len = L[m][n];
    string lcs(len, ' ');
    int i = m, j = n;
    while (i > 0 && j > 0)
    {
        if (X[i - 1] == Y[j - 1])
        {
            lcs[--len] = X[i - 1];
            i--;
            j--;
        }
        else if (L[i - 1][j] > L[i][j - 1])
            i--;
        else
            j--;
    }
    return lcs;
}
```

```
int main()
{
    string X = "AGGTAB";
    string Y = "GXTXAYB";
    int m = X.length();
    int n = Y.length();
    string ans = lcs(X, Y, m, n);
    cout << "LCS is " << ans << endl;
    cout << "Length is " << ans.length() << endl;
    return 0;
}
```

Output:

```
❯ cd "/Users/pargatsinghdhanjal/Desktop

LCS is GTAB
Length is 4
```

Algorithm:

X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
If X[i] = Y[j]
LCS[i][j] = 1 + LCS[i-1, j-1]
Point an arrow to LCS[i][j]
Else
LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

CONCLUSION:

Hence, we can conclude that through this experiment we learnt and implemented the Longest Common Subsequence String Matching Algorithm.