K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch:-B-2 **Roll No:**-16010122151

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 xi=yj

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 X_i and Y_j.

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

Analysis of LCS computation:-

```
#include <iostream>
#include <vector>
#include <algorithm>

std::string lcs(const std::string& X, const std::string& Y) {
    int m = X.size();
    int n = Y.size();

    // Create a table to store lengths of LCS of substrings
    // dp[i][j] will store the length of LCS of X[0..i-1] and Y[0..j-1]
    std::vector<std::vector<int>> dp(m + 1, std::vector<int>(n + 1, 0));

    // Building the dp table in a bottom-up manner
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            if (X[i - 1] == Y[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }

    // Reconstruct the LCS itself
    std::string lcs_string;
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs_string = X[i - 1] + lcs_string;
            --i;
            --j;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            --i;
        } else {
            --j;
        }
    }

    return lcs_string;
}
```

```
int main() {
    std::string X, Y;
    std::cout << "Enter the first string: ";
    std::cin >> X;
    std::cout << "Enter the second string: ";
    std::cin >> Y;
    std::string result = lcs(X, Y);
    std::cout << "Longest Common Subsequence: " << result << std::endl;
    return 0;
}
```

Output:-

```
Enter the first string: iambtechstudent
Enter the second string: btech
Longest Common Subsequence: btech


=== Code Execution Successful ===
```

Algorithm:-

16010122151

Ex :- 10

Longest Common Subsequence

Algo LCS

```
LCS(char X[], char Y[], int c[][n], int m, int n)
{
// The two sequence are input in array
X & Y. The length of the longest
common sequence is right in c[m][n]

for(i=0; i<=m; i++)
    c[i][0] = 0;
for(j=0; j<=n; j++)
    c[0][j] = 0;
for(i=1; i<=m; i++)
    for(j=1; j<=n; j++)
        if (X[i] != Y[j])
            c[i][j] = max(c[i-1][j], c[i][j-1]);
        else
            c[i][j] = c[i-1][j-1]+1;

return c[m][n];
}
```

Time ~~& Space~~ Complexity :- O(m*n)

Space Complexity :- O(m*n)

19.04.24

Conclusion:-

The LCS algorithm efficiently determines the longest common subsequence between two sequences, aiding in tasks like string comparison, plagiarism detection, and bioinformatics analysis. By employing dynamic programming techniques, LCS achieves optimal substructure and overlapping subproblems resolution, resulting in a time complexity of O(mn) for sequences of lengths m and n. Its versatility and effectiveness make it a fundamental tool in various fields requiring sequence comparison and pattern recognition.