**Batch: B-1              Roll No.: 16010122104**

**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* = *x*1 *, ..., xm*  and *Y* = *y*1 *, ... , 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 *Xi* 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 xi <> yj

**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*)*

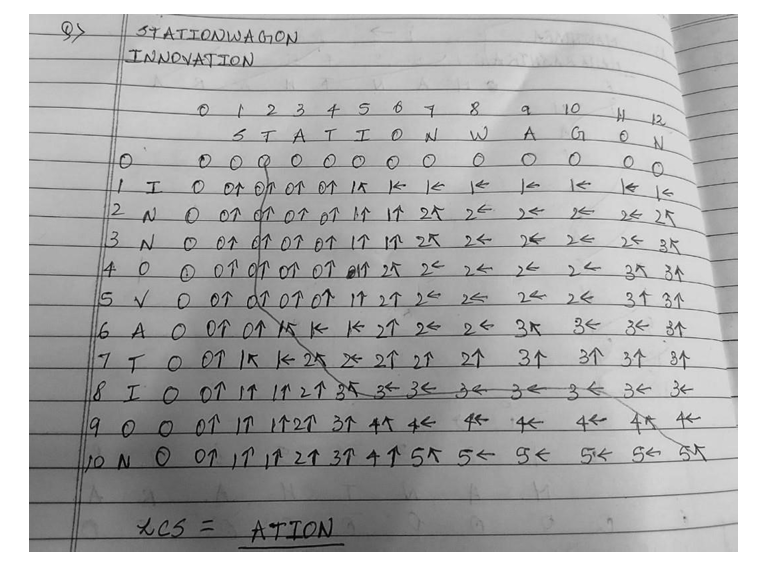
Initial call is PRINT-LCS*(b, X , m, n)*.

*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**



**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:**

def lcs(X, Y, m, n):

L = [[None]\*(n+1) for a in

range(m+1)] for i in range(m+1):

for j in range(n+1): if (i == 0

or j == 0): L[i][j] = 0

elif (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]) l = L[m][n] LCS = [None] \* (l) a = m

b = n while (a > 0 and b > 0):

if (X[a - 1] == Y[b - 1]):

LCS[l - 1] = X[a - 1] a = a -

1 b = b - 1 l = l -

1 elif (L[a - 1][b] > L[a][b -

1]):

a = a - 1

else:

b = b - 1;

print("LCS is ")

print(LCS) return

L[m][n]

X = input("enter String 1:")

Y = input("enter String 2:")

m =

len(X) n

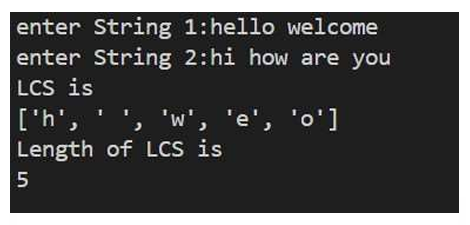
= len(Y)

lc = lcs(X, Y, m, n)

print ( "Length of LCS is " )

print ( ) lc

**Output:**



**Algorithm:**

1. Suppose X and Y are the two given sequences
2. Initialize a table of LCS having a dimension of X.length \* Y.length
3. XX.label = X
4. YY.label = Y
5. LCS[0][] = 0
6. LCS[][0] = 0
7. Loop starts from the LCS[1][1]
8. Now we will compare X[i] and Y[j]
9. if X[i] is equal to Y[j] then
10. LCS[i][j] = 1 + LCS[i-1][j-1]
11. Point an arrow LCS[i][j]
12. Else
13. LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])