LONGEST COMMON SUBSEQUENCE(LCS)

# Algorithm:

**1.** Build L[m+1][n+1], where L[i][j] contains length of LCS of s1[0…i-1] and s2[0…j-1]

**a.** Simply run two loops, outer loop variable i, inner loop variable j  
**b.** For each character in s1(outer loop) select all the characters in s2(inner loop)  
**c.** If  i == 0 and j == 0, then make L[i][j] =0 //Base case  
**d.** If s1[i-1] == s2[j-1], then make L[i][j] = L[i-1][j-1] + 1  
**e.** else, L[i][j] = max(L[i-1][j], L[i][j-1])  
**2.** As we can see L[m][n] contains the length of the longest common subsequence. create a character array LCS[] to print the longest common subsequence

**3.** Traverse the array L[m][n]

**a.** if s1[i-1] == s2[j-1], then include this character in LCS[]  
**b.** else, compare values of L[i-1][j] and L[i][j-1] and go in direction of greater value.

# Correctness of Algorithm:

**• Proof of correctness:**

***- Induction Hypothesis:*** the algorithm computes L[i, j] correctly for all (i, j) < (m, n).(L[i, j] is computed before L[m, n])

***- Base case:*** L[0, j] = L[i, 0] = 0. Correct because empty sequence has no LCS.

***- Inductive Step:*** Assume IH is true. When computing L[m, n], the transition considers 3 cases. By IH, L[m− 1, n − 1], L[m − 1, n], L[m, n − 1] are computed correctly. Hence, the algorithm makes the correct decision and m[u, v] is computed correctly.

Let us take one example:

**String 1: ABCD**

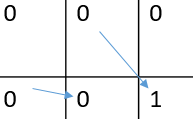
**String 2: BD**

here we call rows as i and columns as j of the matrix.

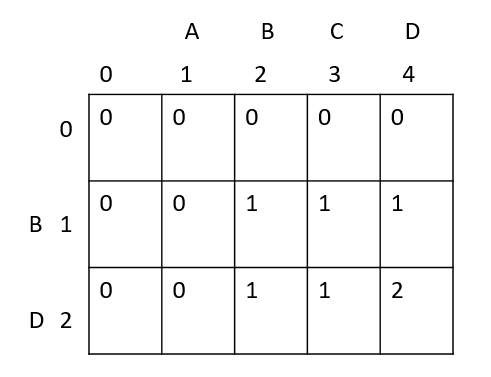
**Step 1**: Initially We have the 0th row and column 0.

**Step 2** : Now we start at i=1 and compare it with j=1 so here B and A are not equal so 2nd condition is applied and max from upper or left side is selected here its 0 so put it in the matrix location LCS[1,1].

**Step 3**:Same way compare the i=1 with j=2 here both are equal so condition one is applied and 1+diagonal element is filled in matrix.

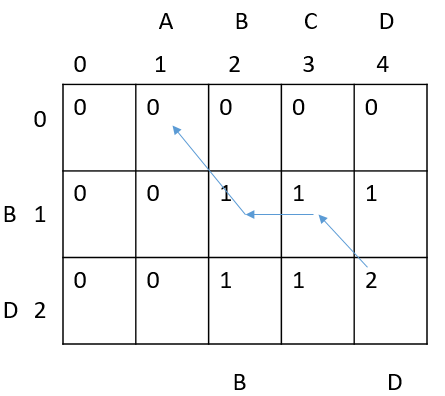


**Step 4**: same way fill the whole matrix which will look like:



Here from last element LCS[2,4] we can have the length of the longest subsequence.

**To find subsequence:**

****

Here we start from last node and go to previous node from which the current value came from after applying condition till the 0th row it is our path.

Here last element is LCS[2,4]=2 which came from diagonal element LCS[1,3]=1 cause the i and j are equal so reach there and again find the elements came from here it came from left side element LCS[1,2] so reach there now it came from diagonal 0th row element 0 so stop.

Now in our path whenever there is upward side arrow consider that for our subsequence (start from left to right) so here first upward arrow came from LCS[1,2] so add B in our subsequence same way also add D so finally our longest common subsequence is BD.

# Complexity of Algorithm:

Algorithm runs in O(mn) time, as double-nested for-loop takes quadratic time and within the for-loop, if statement will be executed exactly mn times. Input and output array requires m+n+(m+1)(n+1) location max function takes constant time. Thus algorithm requires O(mn) time and O(mn) space.