Longest common Sub-Sequence

Problem

Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “ac”, “abf”, “bef”, “acg”, “’dfg”, … etc are subsequences of “abcdefg”.

`Input`: 2 strings s1 and s2.

`Output`: Longest Subsequence present in both

Solution

Defining the subproblems (Giving the nodes of the DAG)

We are given 2 strings $s1[1,\cdots,m]$ and $s2[1,\cdots n]$ and our goal here is to find the size of longest common subsequence.

What kind of subproblems we need?

- It should have the DAG properties
- it should go part of the way toward solving the whole problem

Intuitively, we can see that checking the Longest common Subsequence between some prefixes of the entire strings appears to be good subproblems.

So for some prefix. So for some prefix of the first string $s1[1,\cdots,i]$ and some prefix of the second string $s2[1,\cdots,j]$. We call it the subproblem LCS(i,j) .Our goal is to find **LCS(m,n)**.

Getting recursive definition and Defining Edges of the DAG

Recursive Definition

There are 2 possibilities for the last char of the 2 strings

1. If $s1[i] == s1[j]$, Then $LCS[i][j] = 1 + LCS[i-1][j-1]$
2. If $s1[i] == s1[j]$, Then $LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])$

Edges

Underlying DAG here will be:

- Edge between $(i-1,j) \rightarrow (i,j)$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$. Weight of edge = 0.
- Edge between $(i,j-1) \rightarrow (i,j)$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$. Weight of edge = 0.
- There will Edge between $(i-1,j-1) \rightarrow (i,j)$ If s1[i] == s2[j] . Weight of edge = 1.

Here the solution will be Longest Path in a DAG.

Order of solving problems

We maintain a 2-D DP table here

We can fill the first row and first column using just the base case

$$LCS(0,j) = 0 \ \ and \ \ LCS(i,0) = 0$$

The Longest common substring between 0 length string and another string is 0.

After filling first row and first column. We can clearly notice that if we fill either row-wise or column-wise then we will always have all the entries (left, top, diagonal) required to compute **LCS(i,j)** entry.

Time complexity Analysis

Since we have to fill the table of mn elements.

Each filling takes constant time.

Final time complexity = O(n^2).

Space Complexity Analysis

It takes table of order of n^2. That we need.

Final space complexity = O(n^2)

Example RUN

Step1

		C	B	D	A
	0	0	0	0	0
A	0				
C	0				
A	0				
D	0				
B	0				

Step-2

		C	B	D	A
	0	0	0	0	0
A	0	0	0	0	1
C	0				
A	0				
D	0				
B	0				

Then continuing in the similar way

		C	B	D	A
	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2