NPTEL
NPTEL ONLINE CERTIFICATION COURSE

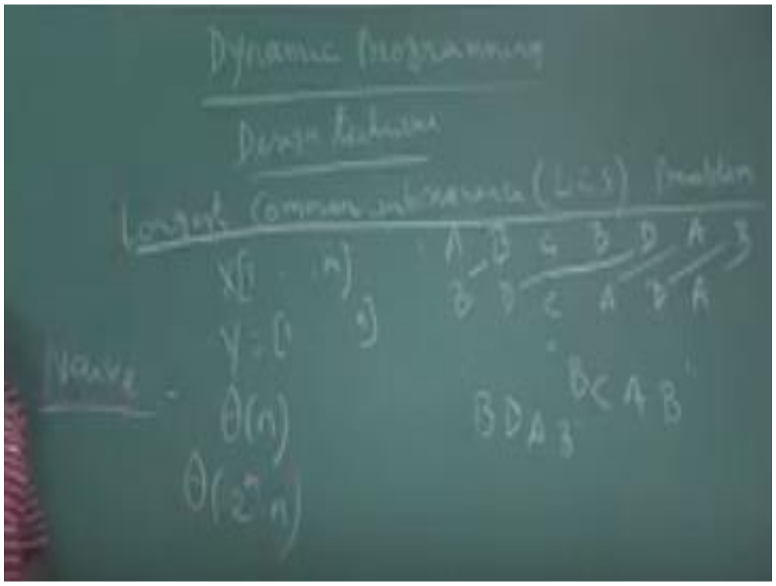
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 15: Dynamic Programming

Okay, so we talk about divide in dynamic programming so it is a design technique like divide and conquer, design technique similar to divide and conquer.

(Refer Slide Time: 00:30)



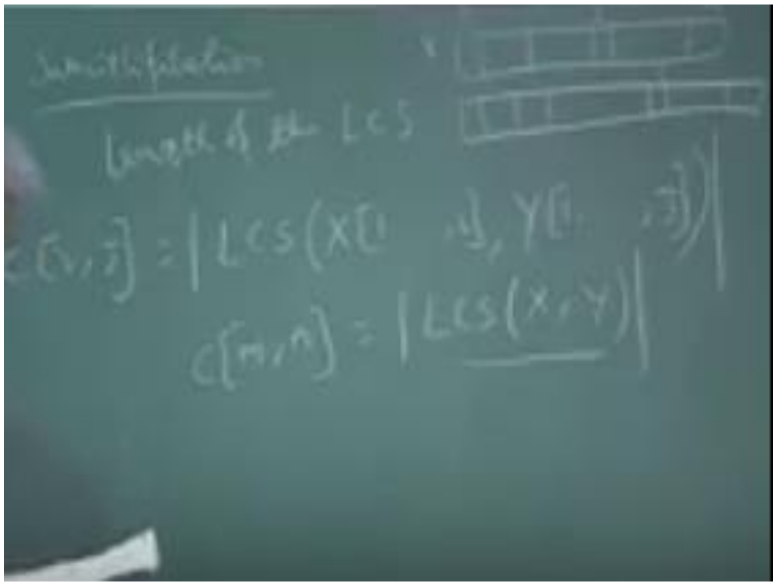
But we will see by technique one problem. One example which is called longest common subsequent problem, longest common subsequent which is also called LCS problem, so in this problem we have given two sequence X[l = m] and Y = [l n] and we have to find the longest common subsequence, for example suppose x is say A, B, C, B, D, A, B, and Y is say B, D, C, A, B, A, suppose these two for the given sequence X and Y and we have to find out the longest common subsequence between them.

So how we can get a longest common subsequence, so then so B, C, A, B, this is a common subsequence of length four so any, we can verify we cannot get any subsequence of length five, so length four is the longest length of the longest common subsequence, so can we have any other longest, any other subsequence of length four? So we can take B yeah, so we can take B, D, A, B, so that could be many, so it is not any but we want to get the problem is to get the longest common subsequence, that is our problem of LCS.

So what is the approach what is the main approach we can think, so we can choose a subsequence from this sequence and we can check whether this is a, we can choose a subsequence from this, we can check whether this is a, this sequence is a subsequence of this or not. So for that if we take a sequence from subsequence from here to check whether this is a subsequence of this it will take order of n because length we have to just scanning the washed case.

And there are how many subsequence we can have, so this is basically power step so totally per n, m, so it is basically n x totally per (2^n – n) so this is the exponential, so this approach is not good because it is not helping our time so we want to see further we can get a better algorithm to finding the longest common subsequence.

(Refer Slide Time: 04:18)



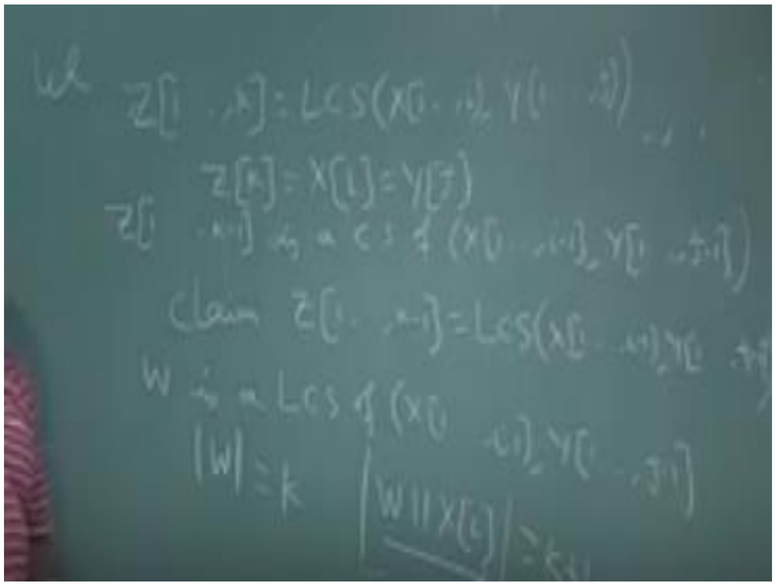
So for that we simplify this problem in a way that, so we look at the length of the longest common subsequence. Simplification, we look at the length of the longest common subsequence and that length is unique, in our example it was four so we want to find first we have a simpler version of this problem instead of finding the longest common subsequence first, we first find the what is the length of the longest common subsequence and then after the after getting the length we will try to get the a longest common subsequence, okay.

So we denote this length by, say we denote a notation L of yeah we just denoted by C [i, j], C [i, j] is basically length of the longest common subsequence of the sequence up to I, this is the prefix this is the Y prefix, okay, so we have the X sequence and we have a Y sequence, so this is from one to n and the Y is this is X this is Y and the y is from one to n, now if you take i over here.

We take a prefix of this we take a sequence up to i and we can take a sequence up to g for y and we consider the longest common subsequence between these two and this length of the longest common subsequence is denoted by C [i, j], C prefix i, j. Okay now if we can find out C [i, j], for all i, j then we have done why because we are looking for the length of the longest common subsequence for full sequence X and Y.

So basically C[m, n] will be verse that, this is the length of the longest common subsequence for the X and Y, C[m, n] so if we can get all the C [i, j], is so from there we can just get the C[m, n] and that will give us the longest common subsequence, length of the longest common subsequence okay. So now we will use a here to n for to have recurse, any recursive formula to find the length of the longest common subsequence.

(Refer Slide Time: 07:19)



Let $Z[1 \ldots k] = LCS(X[1 \ldots i], Y[1 \ldots j])$

$Z[k] = X[i] = Y[j]$

$Z[1 \ldots k-1]$ is a CS of $(X[1 \ldots i-1], Y[1 \ldots j-1])$

Claim $Z[1 \ldots k-1] = LCS(X[1 \ldots i-1], Y[1 \ldots j-1])$

W is a LCS of $(X[1 \ldots i-1], Y[1 \ldots j-1])$

$|W| \geq k$ $\quad |W \| X[i]| \geq k+1$

Okay so that recursive formula is telling this appear here so C [i, j], is of the form C[i- 1, j – 1] + 1 if X[i] = Y[j] otherwise if x [i] not equal to i[j] then it is maximum of c[i-1, j], C [I, j-1] this is if x [i] is not equal to Y[j] otherwise, okay. So this is the recursive call for finding the longest common subsequence, length of the longest common subsequence.

So we need to proof this so we just to try to see the first part and then the second part will be the similar, so to proof first part we assume this case where x [i] is equal to Y[j], so this is our X sequence and this is our y sequence and this is i this is j, now these two values are same, okay.

Now let, let, what is this, okay so let Z be the longest common subsequence of (x[1 – i], y[1 – j]), okay now, and we are assuming the length of the longest common subsequence is K.

So basically this is so now we claim, now we claim that this last one has to be the common value of this and this, why because otherwise if this is if z [k] is the little below this then we can add these two and we can have a one more length k + 1 which conduct it the length of the longest common subsequence is K. So the last alphabet of this longest common subsequence has to be this common value.

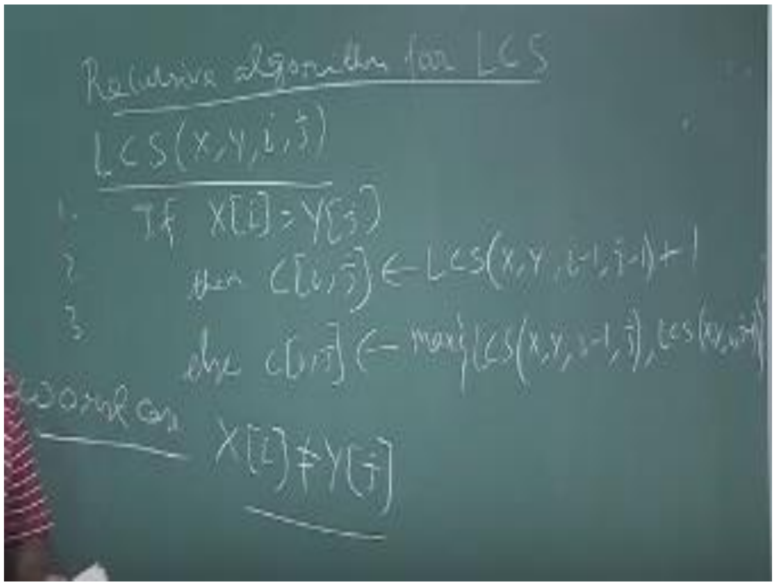
Okay, now if we take Z – 1 then this is a common subsequence, common subsequence of (X[1 – i-1], Y[j-1]) okay. Now we have to see whether this is the longest common subsequence of this or not, so that is our claim our claim is this Z is the longest common subsequence of okay, so this will prove by using the contradiction method.

Cotton pasting, suppose it is not suppose this is not to the longest common subsequence of these two, so we have a longest common subsequence whose length is more than this, so suppose that is W, W is a longest common subsequence of, so whose length must be more than this so it must be at least K. Okay, now we can just take W and this, this is cotton pasting.

So this and this one is same as Y of J so these will be a longest common subsequence for this (X[1 – i-1], Y[j-1]), but length of this is K+1 which contradict the fact that the length of the longest common subsequence of this is K, so this claim is true, so this is longest common subsequence of or this, so that means this is the length of the longest common subsequence this is k which is basically (k- 1) + 1, so this is basically this just now we have seen this is the length of the longest common subsequence, this +1.

So this is the first part of the two of the theorem and the second part also similar where we if they are not equal, if this two value are not equal then we take the maximum of this one and this is less. So that is the recursive formula, now we use this formula to have a algorithm for finding the length of the longest common subsequence of X and X prefix up to i and Y prefix up to j.

(Refer Slide Time: 13:38)



Recursive algorithm for LCS

LCS(x, y, i, j)

```
1.  If X[i] = Y[j]
2.      then C[i, j] ← LCS(x, y, i-1, j-1) + 1
3.      else C[i, j] ← max{LCS(x, y, i-1, j), LCS(x, y, i
```

worst case $X[i] \neq Y[j]$

So this is the recursive algorithm for LCS, okay so we have find the LCS of so if X[i] = Y[j] then C [i, j] = LCS of (x, y- i-1, j-1) +1 else C [i, j] is max of LCS of I mean length of the longest common subsequence actually, these are i[j-1] one is this is less okay so this is the recursive call for finding the length of the longest common subsequence of X and Y.

Now how I find is the washed case, washed case is X[i] is not equal to Y[j] and in that case we have to take two calls we each in discuss one less that is the washed case okay, so now we want to see the washed case recurrence for this, okay.

(Refer Slide Time: 15:58)

So the recursion tree for the washed case we take say n = 3, n = 4 so we want to find out the longest, so this is the washed case. So for these we have to calculate this is wall less so 2- 4 and 3-3, and here 1-4 then 2-3 so here 2-3, 3-2 like this, then here 1-3 then here 2-2, 1-3, 2-2 like this so if you see this part is same as this part, so this is the extra work we are doing, this is the overlapping, this is the repetition of the sub problems, so when we recursively call it, if you do not store the value then we have to unnecessary have to do the many computation, this is the repeated calculation we are doing.

So this is one of the hallmark for the dynamic problem. If we see in our problem we have such kind of repetition our mind will be think that okay, so this is the indication that we should go for a dynamic programming problem. This is all hallmark for dynamic programming doing problem. So let us write the hallmark, so hallmark 1 is basically, this is hallmark 2, hallmark 1 is basically telling optimal sub structure, this hallmark for dynamic programming problem.

So when you think that okay, we, maybe we have to go for a dynamic programming technique. So these two is the indication, so first indication is optimal sub structure. So this is telling us an optimal solution or solution instant to a problem containing optimal solution to the sub problems, okay. So this is the first hallmark for dynamic programming problem, so an optimal solution of a problem, so this is telling us if we have a Z=LCS(x, y), now for the full sequence.

Now if you take any subset of any prefix of Z now that we have already seen that will give us a LCS, longest common subsequence of the prefix of corresponding (x, y). So that is the optimal sub structure, so if we have the solution of the whole problem then that will contain the solution of the sub problems okay.

And this is the basically hallmark to, so it is telling us overlapping sub problems okay. So it is telling us a recursive solution contain a small number of distinct sub problems repeated many times, repeated many times, okay. So this is the overlapping sub problems for example we are we are repeating this calculation so that is one indication that we should think now we have to go for the dynamic property problem, so to avoid that what we do, we memorize the value, once we calculated this C, 2c so we will store it somewhere.

So that for this branch V1, V2 calculate again so this is called memorization or memorization so we will memorize this value in order to avoid the reputation calculation, so let us have that modified version of that code this recursive code. So this is the this is the memorize used in algorithm, this is the recursive algorithm so once we compute some hallow will store it will to avoid the same computation in the future.

So that's why so the code is similar to earlier all the thing so we will only do the calculation if the C[I, j] nil otherwise if C[I, j] is already calculated we will not do it again, so that change we have to do if C[i, j] NIL or it is have some value, then we do the same thing do if when the remaining part of the code is here, then C[I , j] that is recurrence is basically LCS[x] sorry [x, y, (i-1), (j-i) + 1] else c[I, j] is basically maximum of this part is same.

This is the coming from the that recurrence one index is less comma LCS of okay, so this is the way we have heard the reputed computation so this will handle using the using in a table we compute this in a bottom of way so this is the dynamic program for this dynamic programming approach for solving this problem so the idea is to compute the table bottom of way so we will compute the table bottom of way so that we will avoid the computation let us take the same sequence so our sequence was so let us have this table so the sequence has A, B, C, B, D, A, B. This is the x and the y is B, D, C, A, B, A.

B, D, C, A, B, A Okay so, so we will just okay so this the x sequence this is the y sequence now we initial so this initialized by all 0 now this is 0 because this is, this sequence with nil so this there, there is no common such name this is basically C[i, j] and we will use that formula to find this value now we check this and this so these are two the different.

So this will be 0 according to the second formulae maximum of this two, so now we come here so this sequence, this sequence so these two are same, so what we do we take this plus 1. Similarly these two maximum of this two 1, 1 so this is basically common so this is like this and we have 1,1,1.

So similarly if do this here, so this is 0, this is not common maximum of this two this is maximum of this two 1, this is also 1 and now we have a common over here so this will come 2, so 2,2 so let me just finish this so we are using the recursion formulae every time, so is d, so this is basically 0 then this is basically 1 maximum of this two then this will become 2, then this is 2 and this will become 2,2,2 and then we have 1,1,2,2,2 now we have a common over here.

So it will be 3 and then maximum of this 2 is 3 like this is 0, no this will be 1 because maximum of this 2 this should be 1 and then we have a common this will be 2,2, then again we have a common this plus 1,3 we are using the formulae 3 then 3 again then 3 and this will be now 4, okay and this is 0, this is again 1, 2 , 2, 3, 3 and this is comma and this plus 1 4 and this is maximum 4 okay so this is the table 4 cij now so this is the cij is basically C(m,n) is basically 4 okay so this why you are getting the Ci this is the dynamic programming method.

So this is we are not calculating we are not repeating the calculation which we have already did so now from here how we can get the decimal length of the longest common subsequence how we can get the longest common subsequence we will follow this path we will follow this link and we will get the longest common subsequence so this is because of this is because of A so A and then if we take this, this is B and then if you take these this is C.

And if you take this, this is B so this is a longest common subsequence okay so we first find the length and from there we get the longest common subsequence so what is the time complexity for this, this basically to compute this table then this is order of ending to end okay so this is the picnic of dynamic programming technique now we will use this technique in the graph algorithm, thank you.