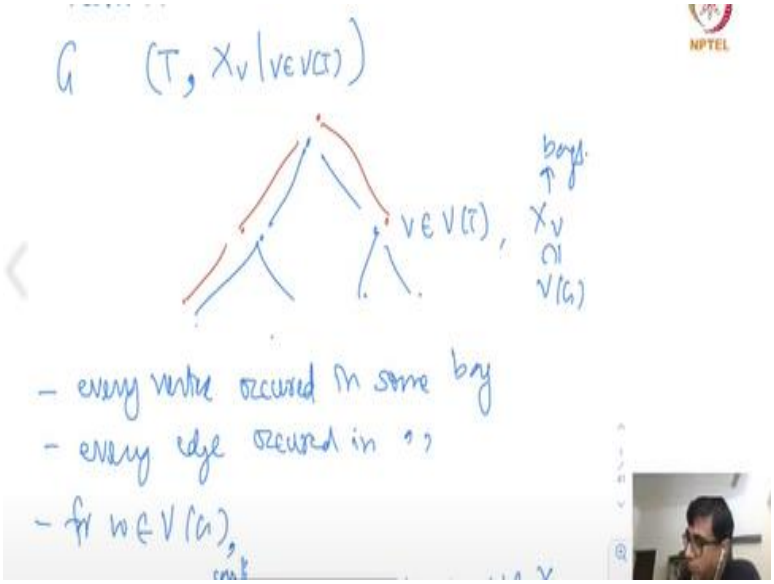
Parameterized Algorithms
Prof. Neeldhara Misra
Prof. Saket Saurabh
The Institute of Mathematical Science
Indian Institute of Technology, Gandhinagar

Lecture - 25
Nice Tree Decomposition and Algorithm for Max Weight Independent Set

Welcome to the week six of the course and today we will be talking about and then in this week we will be talking about more things about tree width. So, just to initiate our discussion let us recall the definition of tree width.

(Refer Slide Time: 00:33)

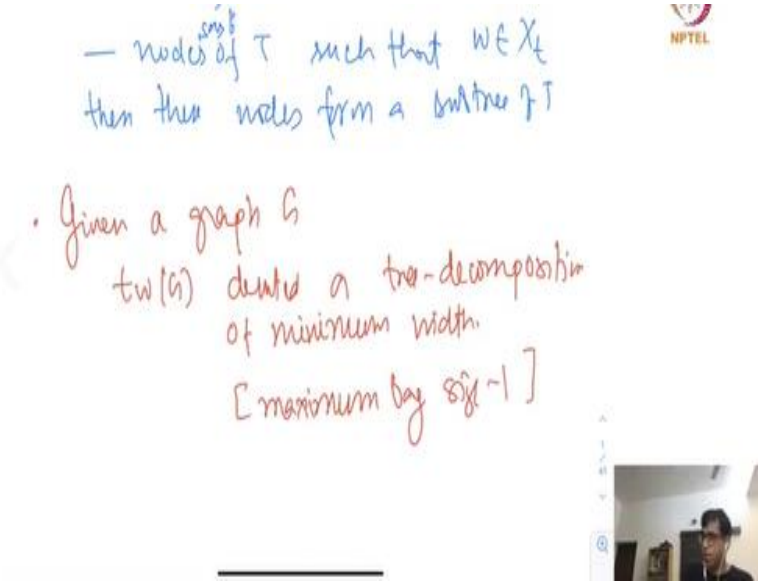


So, what was the tree width? So, given a graph G, tree width of a graph is nothing but a decomposition. So, basically, given a graph G we associated a tree and for every vertex V in every vertex V in V T, we had some X V which is a subset of vertex set of graph. Now this tree satisfied some properties. First, every vertex occurred in some bag and recall these were called bags. Every edge occurred in some bag and for every say w in V G, what do we say?

Look at all the nodes, look at all the nodes of T such that w belongs to all the nodes say t, w belongs to X t, then these nodes form a sub tree of t. Meaning look at all the bags, where the vertices of like look at w some fix some vertex in your graph and look at all the backs in which

they appear. Then this is going to form a sub tree. In other words, the tree induced on these set of vertices are going to be connected. So, these were the three properties.

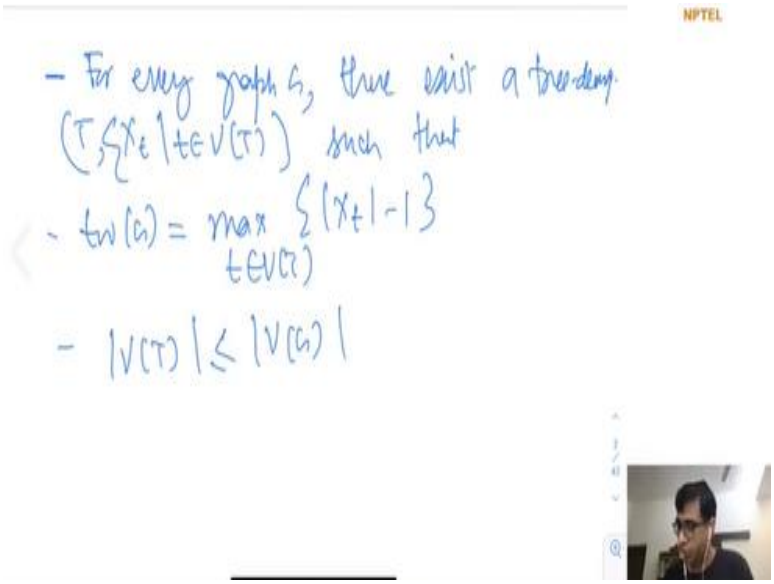
(Refer Slide Time: 02:42)



That we saw, and we also saw that it has a very we also saw that a graph G given a graph G. If you recall correctly tree width of G denoted a tree decomposition of minimum width. And how did we define the width of a tree decomposition? The width of a tree decomposition was a maximum bag size minus 1. So, that was maximum bag size minus 1, this was the differential. And so basically what is the tree with the graph? You look at all the tree decomposition that is possible of a graph.

And you take the one where the maximum bag size is minimized, and that is it. So, this is what we did last time.

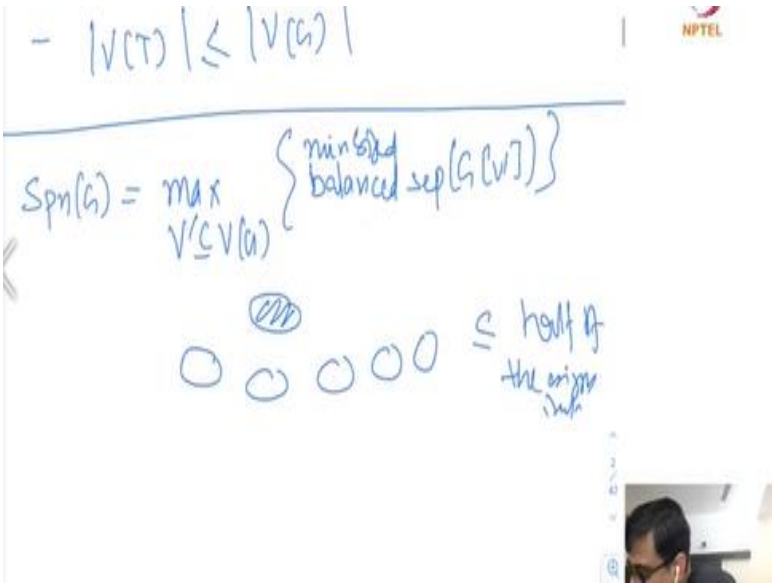
(Refer Slide Time: 03:51)



And so, what we saw is that, so what we saw some nice properties of these like they had some very nice separation property and everything but others property nice property we saw from the structural like for every graph G there exist a tree decomposition T X. Let us call it X t, just lets X t, such that tree width of G is equal to max t in V T, X t - 1. This is that, and what are the property? Vertex in T is less than in vertex object.

So, we can actually come up with a tree decomposition where the number of vertices in the tree itself is less than equal to vertexes of my original graph. And the way we achieved this was by this notion of what we call simple tree composition which basically meant that there is no bag which is contained inside another bag. So, now before we go further and so in this.

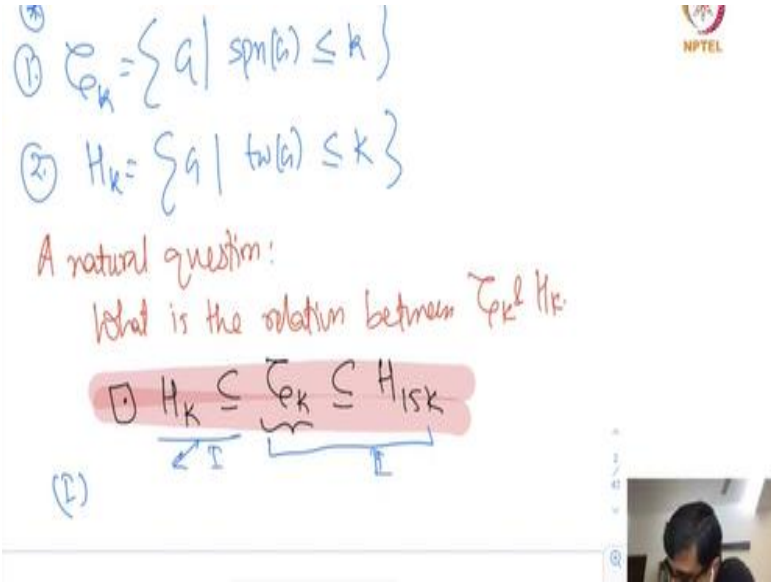
(Refer Slide Time: 05:14)



So, let us come back and define some properties or let us relate the way we started. So, first of all we define the notion of separation of a graph. Remember, so what is the separation of a graph G? So, separation of a graph G was basically the basically it was max over. What was the separation of a graph? It was max over, say V prime subset of V G. And what is called, what we call? Balanced separation of graph induced.

So, basically what it meant is that what is a minimum sized subset, whose removal has a property that every connector component is at most half of the original this type. And we said look at for every induced sub graph find what is the minimum size balance separator and if you look at the maximum over all induced subgraph that will be the separation of a graph.

(Refer Slide Time: 06:37)



And then we define this notion of what was this? All those graphs which separation number was k and now. So, this is our first kind of family, and we so this is how we started our discussion about tree decomposition tree width is that H. I will look at psi k, then it is basically all those graphs for which every induced subgraph we can find a balance separator. And that is what motivated us to define the notion of tree decomposition.

And let us call it, I mean let us call it H k is all those graph whose tree width is at most k. So, a natural question is let us say, a natural question what is the relation between and H k? So, the known relation is that, it was a very good so, what is not hard or what is easy to show is the following. That what is easy to show is that is H k is contained inside this. So, see if a graph has tree width at most k, then it is a separation number is at most k.

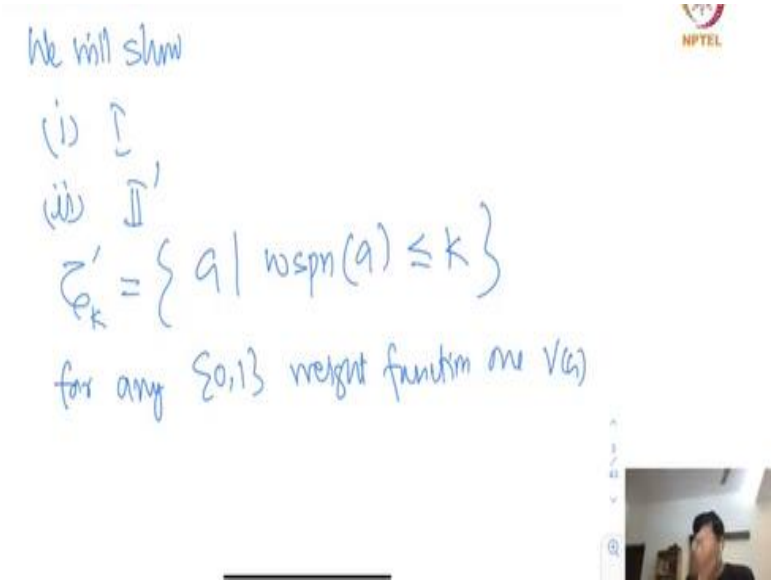
And we will see public actually, we have seen this proof but it is not very hard we will see at some point of time also. But what was interesting is that what was open is that, is it also true that if a graph has separation number at most k, then its tree width is also bounded? May be not bounded by k, but may be bounded by some other function of k. And in 2019 or 20 it was shown, that this is bounded by H 15 k.

So, if a graph is separation number at most k, then the tree width of that graph is bounded by at most 15 k, so this is a very nice combinatorial property. So, in some sense in some sense, what is

shows is that the graph of separation number at most k and tree with are essentially a similar essentially the same object in certain senses. So, we will not be able to like it is beyond the, what you call this course to show this equivalence like this.

So, what we will show first is this will show that is one thing. We will show so let us call this one and let us call this two.

(Refer Slide Time: 10:17)

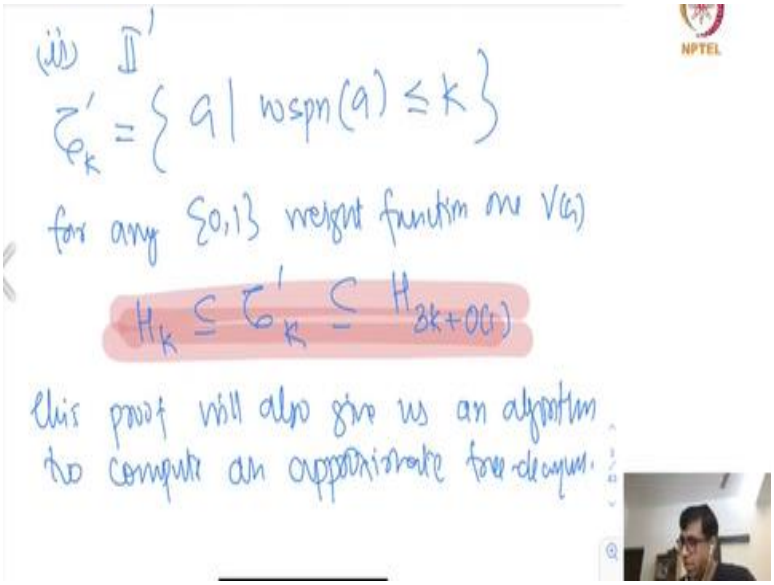


So, we will show. So, what we will show is first one we will definitely so, it is very easy we will be able to show it. And secondly what we will show is not two but two prime. We will show a two prime. What I mean by two prime? So, what we will show is that so, if I define H, so let us define so what I will define is the following. Let us define graph is that let us say separation number let us call it w separation number is at most k.

So, what will be this graph if, let us say so we will define a new family of graph which will have slightly weighted separation number of G is at most k. Now what will this we mean is that the property is that like for any 0, 1 weight function over V G. We will define the notion of weighted separator it means; I would like to find the separator of side at most for every induced sub graph there is a separator of size at most cases that if you look at the weight of every connected component.

That is upper bounded by half the total weight. So, now notice that if I put all the weights equal to 1 for every vertex this is same as the family of separation, but this is slightly more we are demanding much more from this graph class. In the sense that that well this is not only true about the weight function which assigns one to every vertex. But this should be assigned to every weight function that assigns 0 and 1.

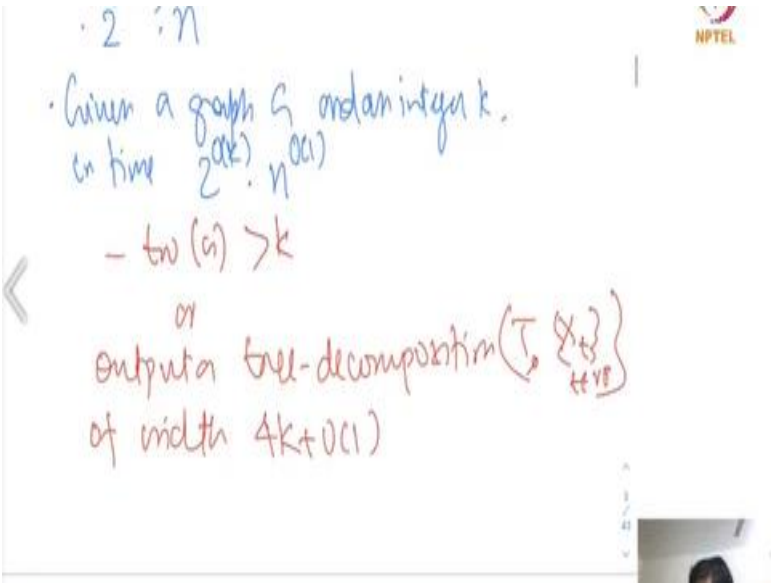
(Refer Slide Time: 12:42)



So, what we will be actually able to show that H k is contained inside this, and this is contained inside H in fact what is known is that, this is contained inside. So, we will be able to show at least we will show something like 3k plus some order one. So, this is what we will show, and this will form and this proof will give us, this proof will also give us an algorithm to compute an approximate tree decomposition.

So, the proof also will have an algorithmic property and in fact that algorithm will run in time 2 to the power big O of K, n to the power big of 1.

(Refer Slide Time: 14:12)



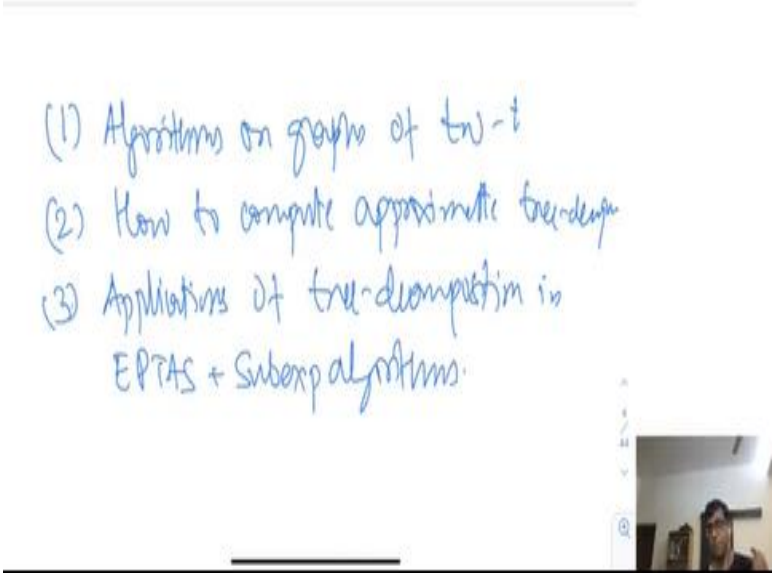
So, in other words what we will do? In this much time, so our problem so given a graph G and an integer k, in time 2 to the power big of K, n to the power big o of 1. We will either say that the tree width of G is more than k or output a tree decomposition T, or output tree decomposition T, X t, t in v of t, of width. So, I think it will be like it should be something like 4k plus order one, but we will see I think this is wrong.

So, basically what all that all I want to tell you that this idea of coming up with a separation number like graph which is separable, graph which is recursively separable is also extremely useful property, useful for graph algorithms. So, we have already seen its applications via we made n to the power big o of K algorithm for max independent set and some other things now after we have defined. So, now because of all this if I wanted to show that if my problem is like.

For example, whether max independent set is FPT on graph class V k, what can I do, is that I could pro, I could compute a tree decomposition of some order k and if on that graph of boundary tree width if we have an algorithm to compute max independent set effectively or efficiently, then that two together combined will give us the desired FPT algorithm for max independent set parameterized by the separation number.

So, from now onwards it will (()) (16:21) separation number and we will only talk about tree decomposition because these two objects or these two properties are linearly related to each other. So, in our first lecture, so this this week is divided into three parts.

(Refer Slide Time: 16:38)



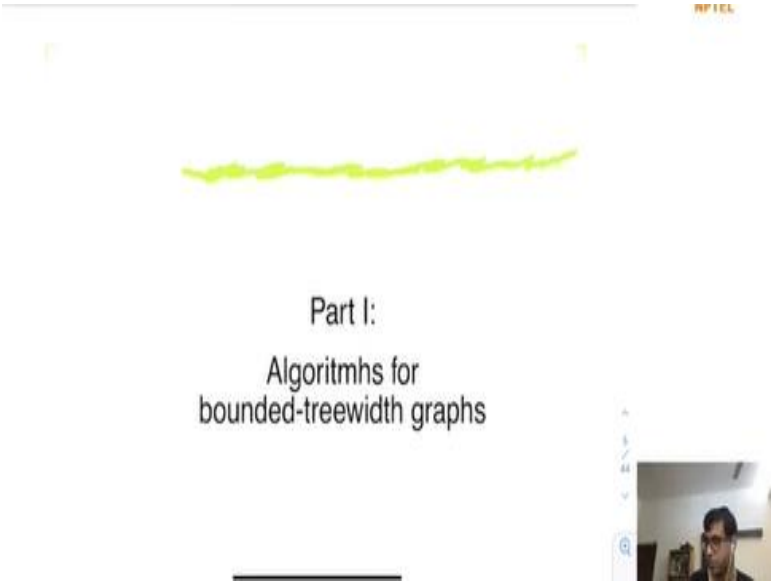
So, first is algorithms on graphs of tree width t. How to design those algorithms t p algorithm? Second how to compute approximately decomposition competition? And finally, applications of tree decomposition in polynomial time approximation scheme in fact EPTAS + subexp algorithms. We already have talked about how this can be used to design FPGA algorithm. At the end of our first week.

When we talked about how a tree width can be used to get designer of them for vertex cover, for feedback vertex set, a long cycle or like k path or some of those things. So, we will, so but in this week, we will see some other application especially for polynomial time approximation schemes on graph classes such as planar graphs and also to design sub exponential algorithms people define what sub exponential algorithm would mean.

You must have seen already about what the sub exponential algorithms are, in the fourth week when you have talked about randomized algorithms and you would have learned the methods of chromatic coding to design sub exponential algorithm for weighted feedback arc set on

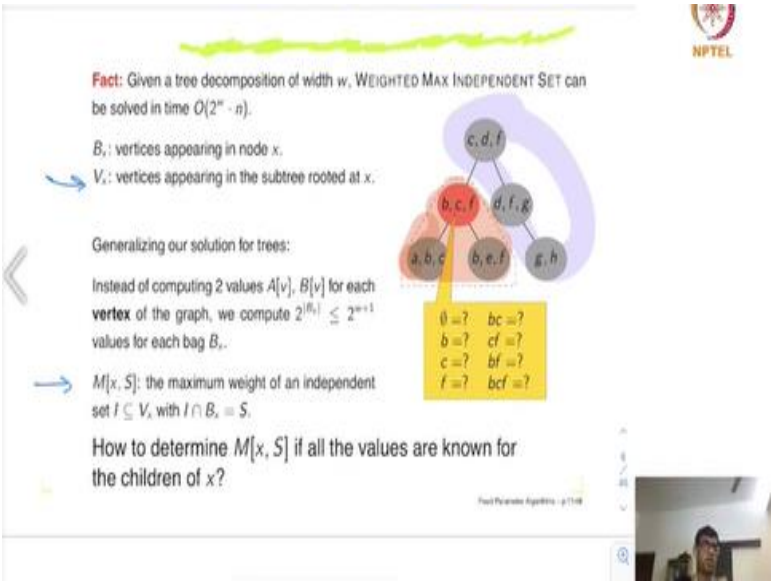
tournaments. So, with these in mind let us move towards designing some dynamic programming algorithms on graphs about it really.

(Refer Slide Time: 18:47)



So, for these parts, I will use slides because otherwise writing it term becomes very difficult and I wanted to give you some weight. But I will give you intuitions about everything.

(Refer Slide Time: 19:01)



So, now we will first do what is called Max Independent Set and tree decomposition. So, this is our tree decomposition which is given. So, remember when we were computing or in fact, we have already done this when we were doing the done like. So, basically the dynamic

programming algorithm is same is trying to make a systematically or way of the whatever divide and conquer algorithm be made.

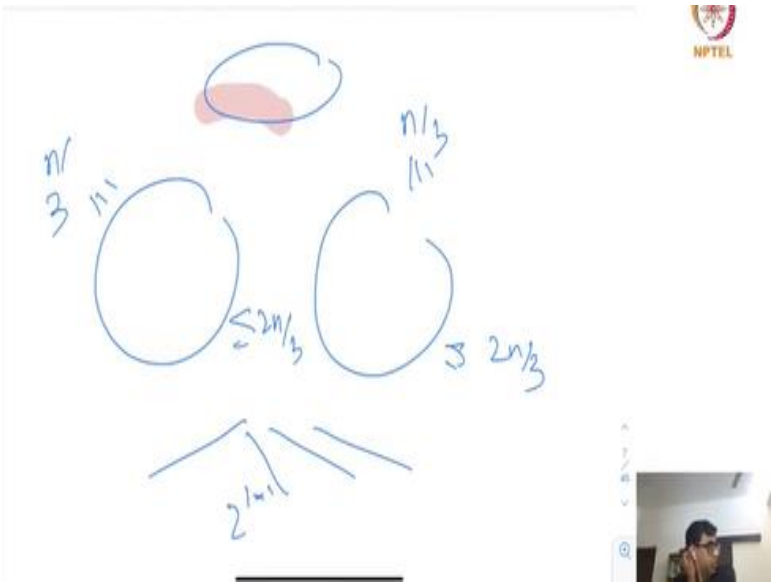
So now let us see or now we will try to generalize the algorithm for max weighted independence that we have made on trees to these kinds of new structures that we have computed. So, in this part of our lecture we will assume that we are given a tree decomposition of a particular width and given that decomposition all we are carrying about at this point of time, how to do dynamic programming over such a tree decomposition.

Basically, we are given a tree and this back function which assigns to every node in the tree, a subset of a vertex subset of my graph which satisfies the property. So, now if you remember we computed two values A v, B v for each vertex of the graph. Now we are going to compute several values for a particular bag. So, what is this? What is so this the bag or the is going to compute is M of x, S the maximum weight of an independent set I subset of V x with I intersection B x being equal to S.

So, what is this? So, suppose this is my bag, I am going to keep for every subset of vertex here. What is the maximum weight independent set for the graph induced on below? So, vertex up so this is remember B x is the bag red colour. And what is V x? Vertices appearing in the subtree rooted at x. So, what is make the weight of the maximum weight independent set I, which is contained inside this and whose intersection with my bag is equal to x.

So, for example what are the possible? If I look at this. What are the possible? Bag what is the possible subset of b, c, f empty set b, c, f these are single tons like then b c, c, f so on and so forth and b, c, f. So, we are trying to compute a maximum weight independent set of graph induced below with some properties.

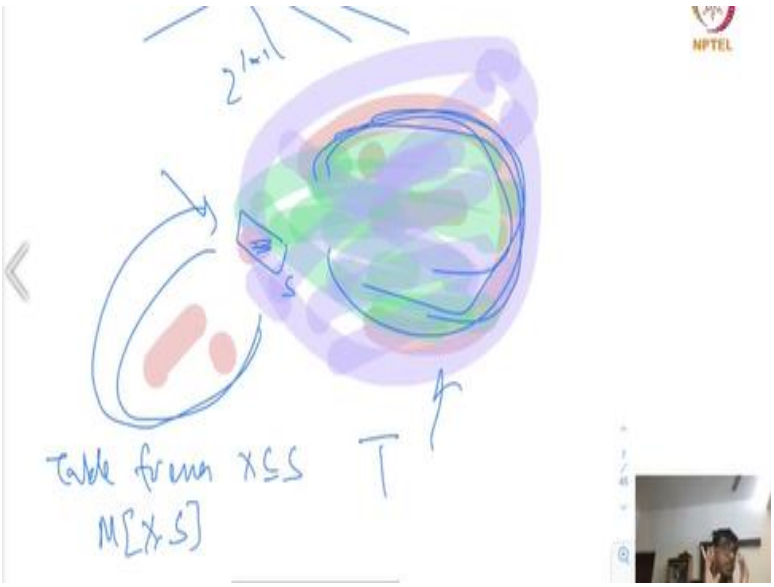
(Refer Slide Time: 21:50)



Now if you are, if you recall correctly how are we doing divide and conquer, we said hey let us find a separator. And for every connected component we say hey look at give me a maximum weight independent set whose intersection with this is this. So, that lead us to and remember, so like for our purposes of explanation let us just assume that we have only two components. And otherwise, you could group them and make into two individual sub graph as we have already seen it.

And each of them has size say n over 3 and it is like, at most at least n over 3 and at most two n over 3 same here. Now what did we do we say hey. So, we actually branched into 2 power the if this was a bag or the separator. So, then we branched into 2 to the power something and we try to find an independent set containing that from each of them and we took it.

(Refer Slide Time: 23:00)



So, now rather than doing this now what is the rule. So, the bag is like this is a small part of graph which is separated from the rest of the graph using this small separator. So, now to compute everything here you are saying okay to compute the maximum independent set of this graph, how will the graph how will it look like? Well, it could look like it will contain something here, something here and something here.

So, all I am going to so, but since there are no edges from here to here because this is a separator. We are going to keep a table which is going to say fine. Any partial solution has a property that from this separator it is going to contain some subset and something. So, we are going to keep we are trying to understand that how what all potential partial solution could look like and given those partial solution what could we keep? What could be stored?

That just that much amount of information is enough if we pro. So, for example look at this part of the graph, look at this part of the graph. So, basically what means is this? If I give this part of the graph hey here is a table. So, what will I give to this part of the graph and let us try to understand that whenever I say like this. So, I say rather than giving in this graph I will give him a table, which for every suppose this is s for every X subset of this.

It basically say M X, S it stores. What is the maximum independent set which is contained inside here with this? So, given this table T and if I give this table to this portion of the graph like

which portion of the graph, we will give this to this portion of the graph. So, what is this? We should be able to find a maximum independent set of the whole graph if I give to this graph, if I give an algorithm this induced sub graph on a small and the table.

So, using just this two information he should be able to compute the max independent set of the whole graph. That is a whole idea of dp is that. Imagine what should I store that if I just pass this information to the rest of the graph, he will be able to compute the problem of my interest by looking at the information and the graph which is small. And this is the idea which we try to, what you call.

Implement whenever we are trying to do dynamic programming over graphs or boundary tree. So, I hope that is clear. So, you should think that if this is a portion of the graph what table what information should I pass on to this graph that he will be able to tell me. So, it is like a game if in some sense like. So, we may be this portion left hand portion of the graph could be very big. But we are weird but the right hand said we can say look hedger portion of information.

What is the smallest set of information that I can provide to the right-hand side? Such that, we can design an algorithm which will give mix a maximum weight independent set of the whole graph. Just at the graph induced on the small portion or the right-hand portion and the table or the set of information that we have provided. So, a set of information but notice one thing right hand side could be any graph.

So, this information that you are passing from left hand to right hand has to be universal. In the sense you give me any graph in the right-hand side, if you have given me the right information for the left-hand side, like the information given any graph on the right-hand side we will be able to compute the max weight independence. So, in some sense you think that you never see the right-hand side and without seeing the right hand side you only see a left hand side.

You have to compute set of information, that give it any right side if I give this information then he or she an algorithm will be able to compute the problem of our interest with respect to this information and the induced graph. So, like in some sense what I am saying to you is the

following, is that I have a portion of the graph. Now I will give you information that irrespective of whatever graph you can come up from the right side if this plus, this is the real graph.

Then this information is good enough, that just with this information and this graph you will be able to compute the problem of your interest. Now like right hand side could change but my information never changed. Of course, the answer will change based on how the right hand side will work. But I have been a but for the left hand side I have been able to compute a succent information that in completely encodes the problem of our interest.

And that succinctly encodes is what we call table or the dynamic programming table when we have do it doing graphs algorithm over graphs amounted. So, for the max weight independent set what is the information we would like to store. So, the information is for every subset of my bag, I would like to say what is the maximum weight independent set of my graph induced on the vertices below or the which will constitute in the some sense if you think of red.

So, that automatically separates my graph into left and the right, right is basically all these things and left is all these things. So, this is your left and so and b, c, f is a separator. So, what information? So, the information we would like to store is for the left hand graph and for every subset back, what is the maximum weight independent set whose intersection with my bag is exactly equal to S that is it. Now so this is an information you would like to.

So, next question is how to determine M of M x if all the values are known for the children of x? Because whenever we were doing dynamic programming over trees if you recall what we did we had information for every children. Based on the information, the children we were able to compute the information for the node we were considering. Now similarly we will say hey look at b, c, f suppose imagine that this table is filled for all your children.

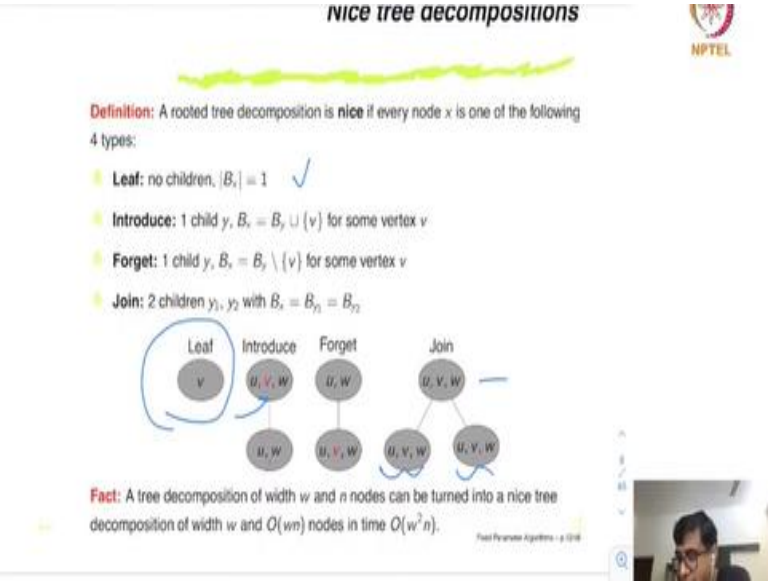
Can you come up with a table for me? So, it is like this, so why did we come up with this tree is like a guide for us how to do our dynamite programming. So, that is like bottom up. Now and at any point of time you would like to store information for the graph induced and below. So, if I

look at c, d, f that will constitute a graph induced on everything containing this and below which is like whole graph.

So, if you are able to store this information then, look at the root node what we would have stored. Well, what is the maximum weight independent set? Whose intersection is every possible way? So, what is the maximum weight independent set of my graph well its intersection with some will with the root bag is going to be something, so just take maximum of all these values and that is your max weight independent set of your graph.

So, we understood how to do dp, but how do we pass on this information? How do we change this information?

(Refer Slide Time: 30:49)



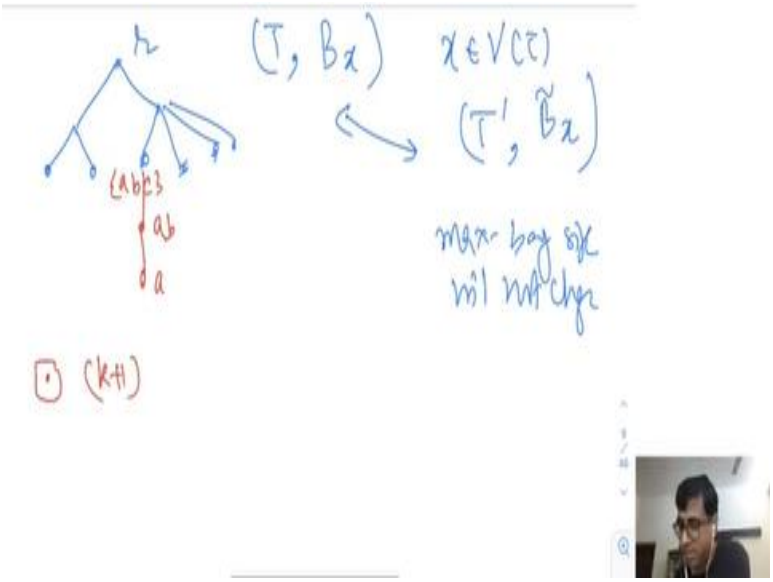
And to do that we will define this notion of rooted nice tree decomposition, which helps us or rather allows us to do dynamic programming algorithm much more effectively, much more easily. So, rooted tree decomposition is nice if every node x is one of the following times. No children this is called so basically the back size is one this is called leaf node introduce node so this is your leaf node a single term and it is a root tree. So, what is introduced node?

So, introduced node is basically a graph where you have u, w and they the difference between the like child and this is just one single node like. Say for example we have introduced v here u.

What is a forget node? It is a node where the difference between it and its children is just by one. And what is a join node? Join node is basically a node of degree two. So, notice that what is the difference between a normal tree decomposition and nice tree decomposition.

So, normal tree decomposition could have like we do not satisfy all these things like I mean bags are all over the place all we care about the; bag size. But I mean but when we have such structured tree decomposition, then it becomes very easy to do dynamic programming or it allows us to do dynamic programming algorithm in a easier way. So, now is, so the very basic fact is a tree decomposition of width w and n nodes can be turned into a nice tree decomposition of width w. And at most order w n nodes in time this.

(Refer Slide Time: 32:36)



So, let me tell you how actually people go about proving this fact. Let us see if I can cut this further so I guess this is known it does not matter we have this fact. So, this is not very difficult, this is not very difficult. So, imagine, so I will let me explain to you how we so suppose this is fix so this is any tree decomposition tree. And what is the notation is used? So, some I mean let us call it B x does not matter you have used the node x and x in V T.

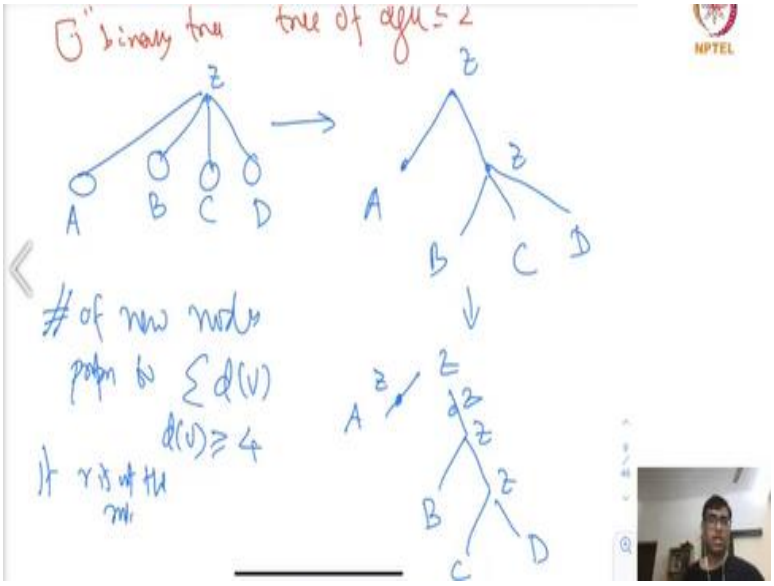
So, this is now I am going to give you a tree decomposition another tree decomposition say T prime and like say another B tilde x. Such that what is the property? Like max bag size will not change. It is very easy, so suppose this is this is how it is. And so, you start with this you make it

somewhere true does not matter. So, what property do we need? So, first we need that, so let me tell you how I mean how things will be done. So, suppose let us make leaf nodes.

So, suppose here we had a, b, c here assigned, at this. So, now what will I do? I will add one suppose this has bag had three then I will add another two nodes and I will just arbitrarily say ab and just say that is it so you arbitrarily start reducing one by one size. So, notice one thing is that by doing this what has happened is that every vertex appears everywhere, every edge appears everywhere and because of this the connectivity property will also be whole.

So, now notice that all the leaf bag we have got all the leaf no children and B of x =1. Now so this is an operation we do and notice that this never changes the bag size and it only increases the number of bags, say for this node like currently for how much of a number of leaf nodes were I mean this can suppose if the width was k, something like k + 1 or at most k more k + 1 per this thing we are going to add.

(Refer Slide Time: 35:56)



Now notice so this is what we will do for this now, what we will do is let us make this a binary tree or let us say tree of degree at most 2. Now look at a node. So, what will do transformation? So, the transformation we will do is the following so suppose just to make you. So, let us call this node so I will do this first this this and let us call this A B C D and this is like Z like. So, I Z keep, I k keep a I make a copy of Z and make B C D.

Now and then what I will do? I will make a transformation further Z A. Now this Z, so notice that when I did this transformation, I have made like the top Z has become a degree at most two. Now this Z I will again, I will do the same operation. At every time I will sacrifice one, I will make Z C D. Now we are done, because now this otherwise if G also would have been more you will make copy, copy, copy and copy and everything and you will do it.

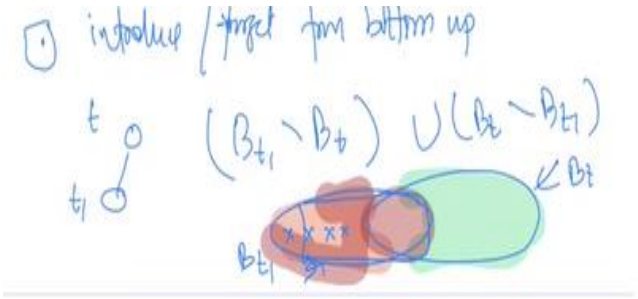
Now this is how you make a tree into one, now you can ask what happens to all the property well. You just so whatever bag was the sign said here you have signed it so vertices appear everywhere it did appear everywhere and connectivity, well connectivity will remain the same because you are just copying the vertices copying this bag here. So, this is how you can convert this into this.

But this is not a like but this may not be a join node. Why this is not a join node? Because look at this it could be that this is fine Z this is A. But this is not but to make and join node it is not very difficult to just subdivide every edge and keep said here. I mean it means nothing, so this is how you could make sure that by doing this operation and how many number of new nodes will add number of new nodes.

You can check is basically look is proportional to summation degree V, where degree V is greater than equal to 3 like or the like children of V. Where degree of V is 3, so then in fact degree of V should be 4, if r is not the root, because then you have degree greater than equal to 3 and you do this. So, basically every time when you will make a new node you would have basically you would have reduced the number of vertices of degree like number of edges which participates into greater than equal to degree 3 by 1.

So, the, so since the number of edges in a tree is upper bounded by minus one you will do this operation not more than some order n times and this is. So, this will add some another order m. So, this is how you make your graph binary and this and how do you make a graph?

(Refer Slide Time: 39:53)



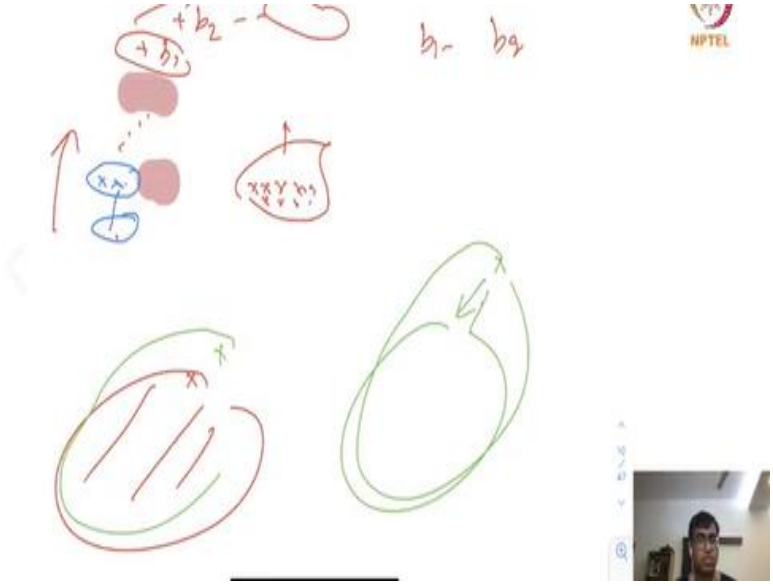
Let us say how do you make your graph introduce node and forget node that is also not very hard. So, you do this introduce forget from bottom up or whatever. So, look at you are doing this bottom up look at for any node and look at its parent, after you have done this operation. So, suppose, this is like my t 1 t and this is t 1, so this is my parent. Now what will I do? I will say look at B t 1 minus B t. So, basically what I am saying is union B t - B t 1.

So, suppose this is how the world looks like, this is what. This is and this is like common portion, this is a common portion. Now what will I do is the following. So, what is this tells us? So, suppose this is like B t 1 is this and this is B t. So, my objective is to go from back t 1 to t. So, now what will I do is the following. So, I say, I need to go from here to here so what suppose this has elements, what will I do is that, I will start forgetting them I will start forgetting them.

So, first I will so what will I do in the next we will only have this portion. So, one less but like you keep everything here like, you keep this push this common portion you do and the rest of the portion you just look at this this portion like. So, what is this? So, you this is a portion. So, common portion is this, you just arbitrarily order the vertices in one way and in every node keep forgetting one thing.

Now notice that the difference between this is just one vertex that you have forgot. This is same as what we did for the when we were doing for the leaf node.

(Refer Slide Time: 42:36)



So, once after doing all this operation, notice that in each operation nothing would change because the same the property of connectivity vertex edge remains. So, finally what will you reach after doing this step? You will reach just this rate portio the common. Now what I need to do? I need to get to B t so I need to introduce these green vertices, so suppose these green vertices are like b 1 to b q. So, now add b 1 next time, next time add to this b 2 so on and so forth.

And finally, you will reach what? The common portion and all the b 1 to b q vertices they are intuitive so, whatever. So, in the first part I forget one vertex each, in the second part I introduce one vertex each till I reach this. So, notice that between t 1 and t how many new nodes will I create, it is basically sum of number of vertices in t 1 and number of vertices in t. So, number of bags that will get introduced is like some ordered w.

So, again this is like some order n times k + 1 or something. So, basically you will never introduce more than order n k bags throughout this processor that is it. So, what we have been able to do by doing what have been be able to do? We have been able to so starting from any given tree decomposition we can come up with another tree decomposition which is a very special property that it is like no vertices degree more than two.

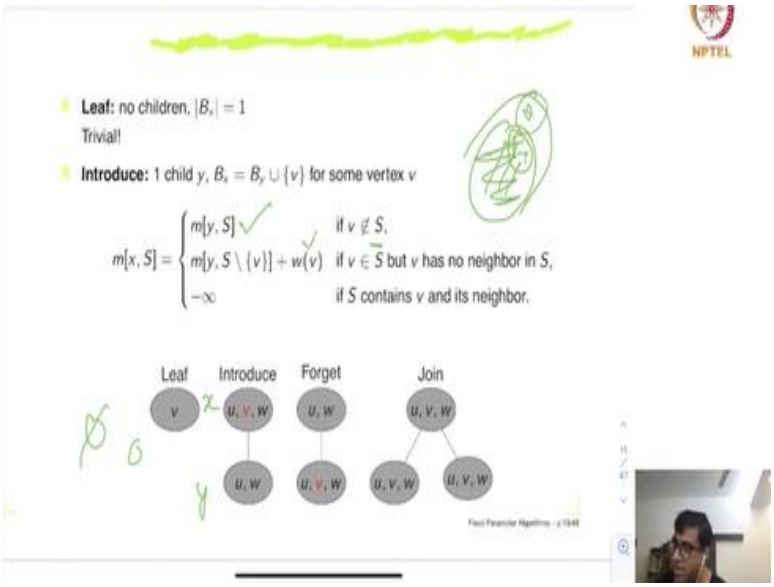
If I look at any like if I look at a look at a vertex and its parent the difference between them is like either it is a same bag or this bag is one more than this bag or this bag is one less than this. And if you think of in terms of graph what is happening. So, notice what is happening in terms of graph? So, suppose here is my bag look at the graph induced on this. When you are forgetting the number of like the vertices which are below this does not change it.

So, basically till you are forgetting as a graph nothing changes. So, if you think from the graph perspective forgetting is just a syntactic way of helping ourselves out. But as a induced craft nothing would change because, this bag is contained to the previous bag. So, the vertices which appeared in this bag and below is same. And but look at the introduced node, what is it look at the graph and you have just introduced a single vertex.

So, basically what happens as a graph if you look at this, now I have got a graph which is one vertex more than the previous graph. So, we could do some of these like so just because now it is like if you remember (()) (45:27) compression when one vertex are added life becomes more easier to handle. Similarly, here may like it is not very hard to deal even if you add subset of vertices because the number of vertices that you can add at any step is bounded.

But when you just add one making your dynamic programming writing your recurrences becomes so much more easier and cleaner that it is it is a very good way. So, this is why most dynamic programming algorithm if you see on you start with a graph of boundary tree with you. Like you convert that into a nice tree with a nice graph of like nice tree decomposition and then you do dynamic programming algorithm on that particular decomposition. So, now what we will do?

(Refer Slide Time: 46:14)



In the next 5, 10 minutes is or 5 minutes is to see couple of more examples of how to do this dynamic programming and how it becomes so much easier when we have nice tree decomposition. So, how do you go about it, so remember what we have to compute so suppose so now we will talk about the like dynamic programming algorithm for max weight independence. So, what is the max independent set of leaf child?

Well back size is one, so it is trivial like if it is empty, then you put 0 if the back contains v then you add the weight of this vertex done. Now look at the introduced child, what is an introduced node? Introduced node is what like look at u, w and u, v, w. Now what are the two possibilities? When you have introduced. So, what is this? Either, this newly added vertex is part of the S or it is not part of S.

If it is not part of S then what happens? So, this is so this bag is called x bag and this is called y. So, I am looking for a max weight independent, so look at the graph as I told you right here is a graph and here is a v added to the graph. So, what is the max weight independent set? That does not contain S same as max bit independent set of S in the smaller graph. So, you put whatever you have stored previously.

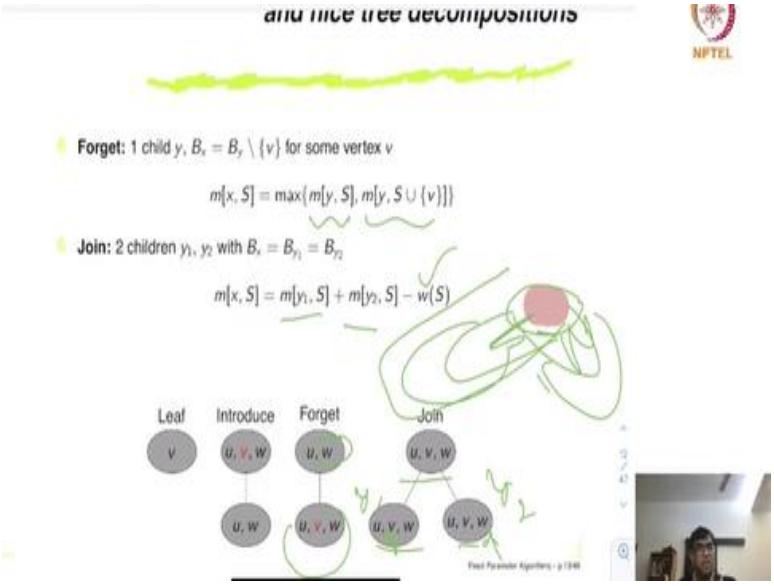
Now but if I decide to add v then what happens? I have to add its weight, but if I decide to add weight its neighbours cannot be part of this. So, basically, I am looking for delete all its

neighbours but where are the neighbours of e, neighbours of v are in the bag s itself otherwise that cannot be a separator. So, that is the property we use to say that look, look at the neighbours of s. So, we are looking to find a max weight.

What is that? We are looking to find what is the maximum weight independent set of x, s. So, what is the max weight independent set, whose, what you call whose intersection is this is that I remove the neighbours of this compute the max weight independent set of my child. But v has no neighbour in it this is important, because if v has a neighbour in s then you cannot find a max weight independent set whose intersection with this bag it just s.

Why? Because well v and its neighbour both cannot be together, then you will put minus infinity. But if these neighbours are not there then you say look as we have done. So, what is the max weight independent set of this graph? As well it is a max weight independent like v and the max weight independent set s - v of the graph induced below, which we have already stored here. So, that is it, so now we are done with the introduce node.

(Refer Slide Time: 49:20)



Forget it is very simple, what are we trying to store? What is the max weight independent set? Such that its intersection is this S. Now what is the difference between here and here, you have got some S. So, max weight independent and remember I told you, this as a graph I do not change anything. So, what is the maximum independent set? Either it contains v or it does not

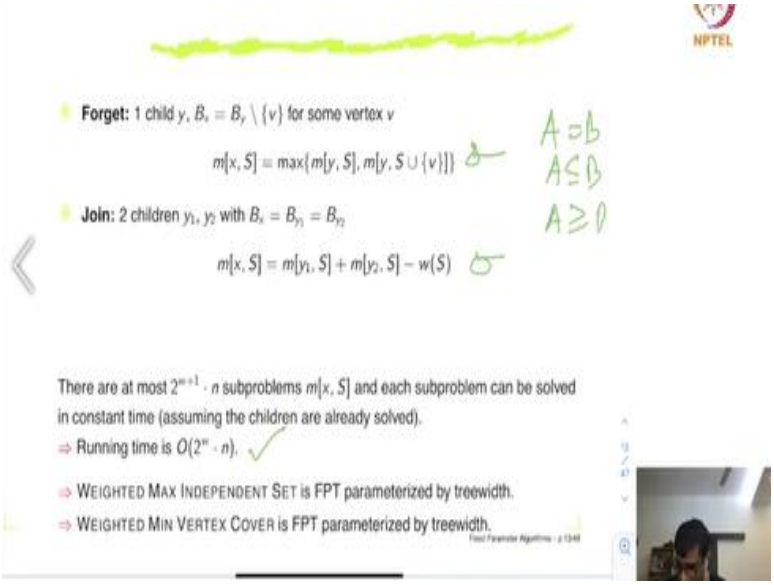
contain v. But I know that both of these values are stored here. So, I just take the maximum of this.

What is the maximum weight independent set of at this node is basically max of these two, either it contains or it does not contain whatever is the maximum value you do it? But now this is an interesting case is join node. What happens at join node the bag, bag, bag remains the same like in the terms of vertices. So, in terms of your graph this is exactly what happens. So, this is like if you look at the graph, what is the graph?

So, if you look at the common u, v, w here is a portion of graph. Now whatever is what is the max weight independent set? Whose intersection so well, so it is a max weight independence it contained here and it is a max weight independence will contain here. So, now what are you doing what is so it has a two children y 1, y 2. So, this is its children y 1 and y 2. We have already stored what is the maximum weight independent set containing this in the left side.

So, this is it, at y 1 we have also computed what is the max weight independent set here containing this. But now weight of S is added both in this and both in this. But so, you subtract one weight of s and that is it. So, this is how you can increment your what you call your table.

(Refer Slide Time: 51:22)



And so, there are at most 2 power w + 1 times n sum problem and each sub problem can be solved in constant time assuming that the children are solved. So, the running time of this algorithm can actually is like some 2 to the power w times n. And that is so that weighted max independent set is FPT parameterized by three weight and that also shows that weighted minimum vertex coverage is FPT parameterized by tribute because they are complement to each other.

But one more thing I would like to make a point here that I have not told you why this algorithm is correct to prove that this algorithm is correct you have to show that each of the recurrences you write here they are correct. So, how do you so that some recurrences is correct? So, basically you have to say that if suppose a = b then you have to say if this value is less than equal to this value this value is at least greater than equal to this value.

So, you have to show this all the time. So, like a is greater than equal to less than equal to b, so like this value is at least this much this value at most this much you will have to show this for each recurrence. Please refer to the textbook there is one completely clearly how this proof is done is given in the textbooks. I would request you to go and check the textbook for max weight independent set proof. But I have given you how to come up with references.

It is very natural how to how to update these recurrences. And so let us just do, I think at this point of time I will stop for this lecture. We will see some more examples of how to do dynamic programming or across a bounded (()) (53:06) in the next lecture. Thanks a lot.