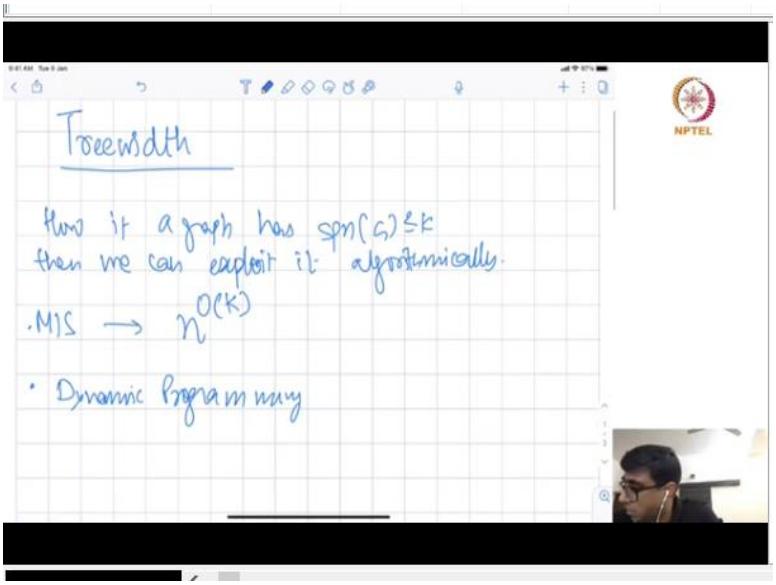
Parameterized Algorithms
Prof. Neeldhara Misra, Saket Saurabh
Department of Computer Science and Engineering
Indian Institute of Technology-Gandhinagar

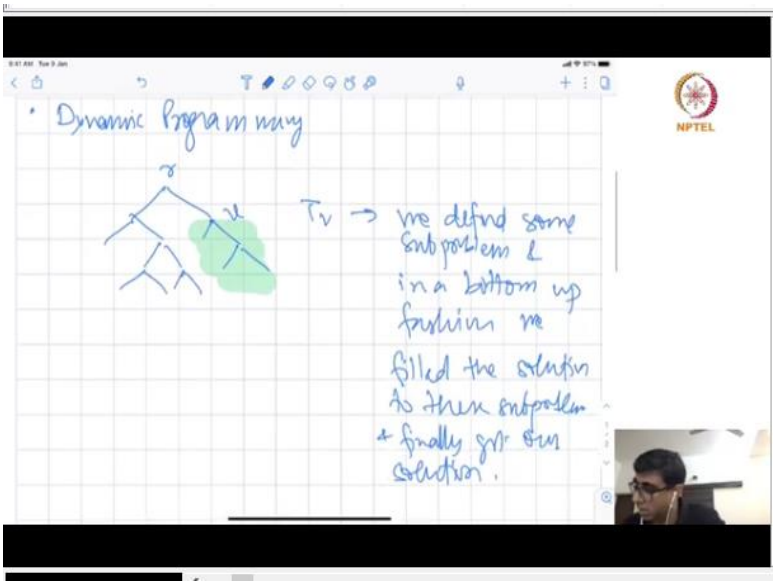
Lecture-23
Treewidth and Constructing Treedecomposition of Few Graph Classes

(Refer Slide Time: 00:16)



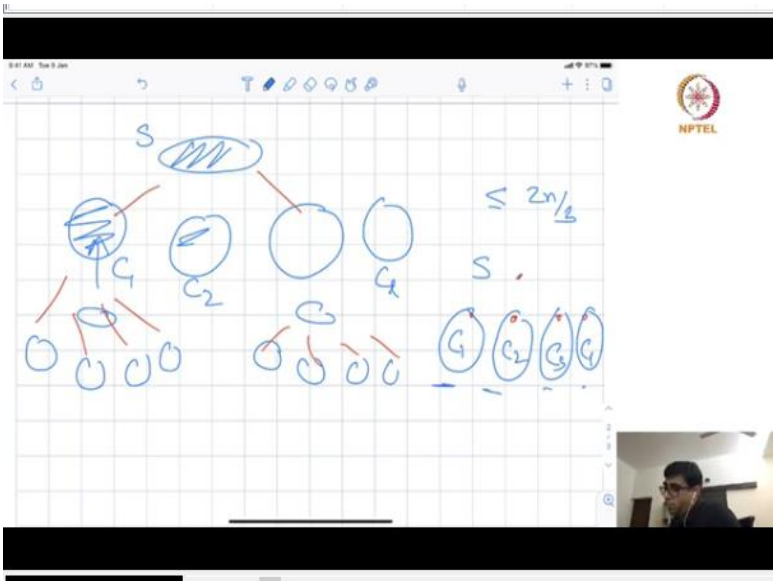
Welcome to the third lecture on treewidth. Up until now we have seen how if a graph has separation numbers small then we can exploit it algorithmically and then but if you try to do even max independent set this algorithms run through in time n to the power O of k term. So, in desire to design an FPT algorithm, we tried to take an approach of dynamic programming and if you notice, we given dynamic programming algorithm for trees and also for the subclasses of grades.

(Refer Slide Time: 01:12)



But if you notice for the trees, the way the problem worked was that for every vertex v we routed at some vertex route and we had this notion of T v which is basically a subtree rooted at a vertex v and then we defined some sub problem and in a bottom of fashion, we filled the solution to the sub problems and finally got our solution which was great. Now, an ideal approach or ideal way would be if a graph that has.

(Refer Slide Time: 02:25)

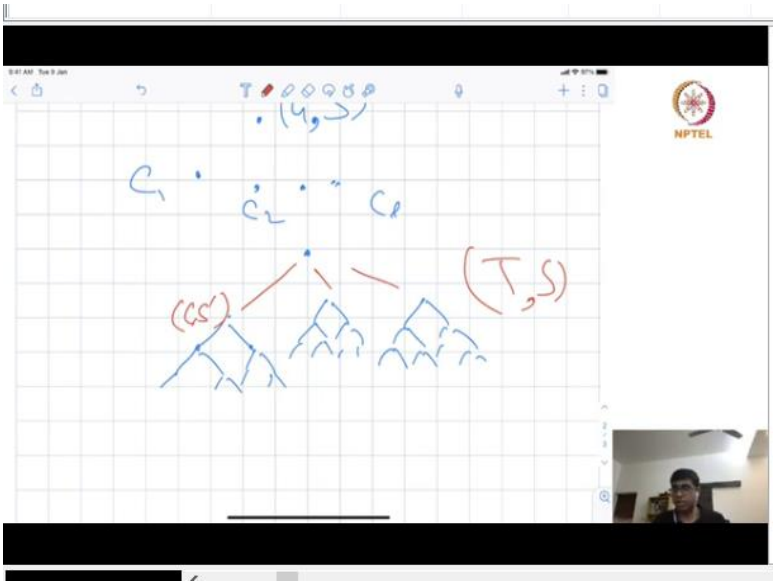


So, now, let G be a graph that has vertex separation numbers say k. Now, notice that the way we use this algorithm, the way we use this structure for computing max independent set while doing divide and conquer is as follows. So, how did we use? We actually found a separator S and then we actually partition these 2 into 2 components where each of them had sides at most 2 n over 3.

And now, again we found so basically like in the beginning, we can think that we have this tree because now what are we going to do? You are again going to find a separator for this component something and then you are again going to get smaller. So, notice that the structure is like a tree. This is already giving us some sort of tree structure. So like even inbuilt tree structure when we decompose our graph, we remove we have component view fine.

So basically, you can think of this that there is like or rather think, let us not think in terms of 2 but like even if there are some several components, we delete some vertices and we have several components. So, in some sense, what are we saying fine, head a tree, So, now let us associate this set S to this tree and say hey, go and find a decomposition of these graphs or these components say C 1, C 2, C l. So, this is like component C 1, C 2, C 3, C l and suppose you are able to get some tree like decomposition of these objects.

(Refer Slide Time: 04:26)

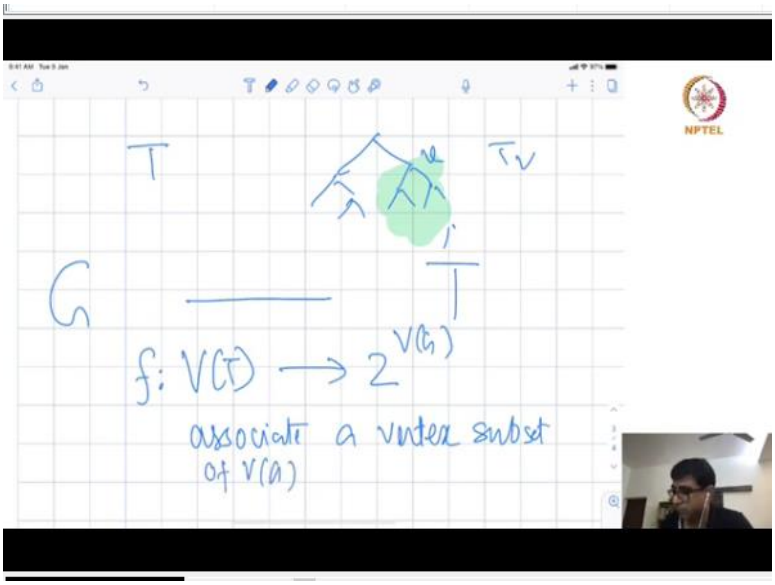


So, what happens, so, like now, if I look at this root vertex, we say that look, this is like suppose I write G, S, we say look, this represents the whole graph, where if you delete this set S, you get into several connected components. So, now these components C 1, C 2, C l, suppose inductively or in some way we are able to find some tree then in some sense, we are able to get a decomposition tree for the whole graph by adding an extra node here and add in these S to get a tree T.

Now, these components again will have like whatever the component and some other vertex set spraying and such kind of these things can happen. So, notice that inherently when we are

trying to do this divide and conquer and we are distributing the components, the way these components are forming actually is a tree like manner or a tree like fashion. So, that tells us that if the graph has a small separator everywhere then we should be able to explore this structure to also arrange this graph in a tree like fashion. And we say look, this part of the graph is associated to this kind of tree and so on and so forth. So, this is the kind of objects we would like to say.

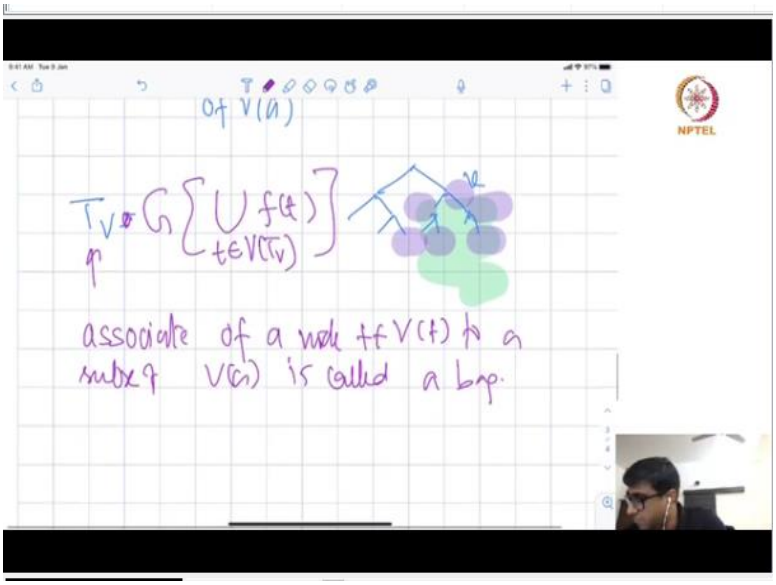
(Refer Slide Time: 06:05)



So, what are we saying, that given a graph G our goal is to associate a tree T in a way that what else to be want, a pieces of graph separated by small by some S is represented well. So, in some sense what are we going to do the way we had for the tree, so what are we going to do? So, like if we recall tree we had this tree and we automatically had an association look at v what was T v?

T v was subtree rooted at v. Now, our graph is not a tree, but I have a graph G and have a tree T. So, given a graph G I would like to associate a tree T and now what I would like to say? I would like to associate a look at a vertex set of T and you are going to associate not one vertex but associate a vertex subset of v G. So, this is what the function will do now. So, and you will see what this generally. So, we will get to that how we do these things, but now what will this mean, what kind of graph.

(Refer Slide Time: 07:54)



So, suppose this is T and I have a vertex v. Now, T v will again mean this, but what graph do I consider? But now this will be graph induced on union t. So, like with respect to T v the graph associated will be graph induced on T in vertex set of T v and f of t meaning look at the vertex sets, look at this function f it is assigning some subset of vertex here, here, here, here, here, here.

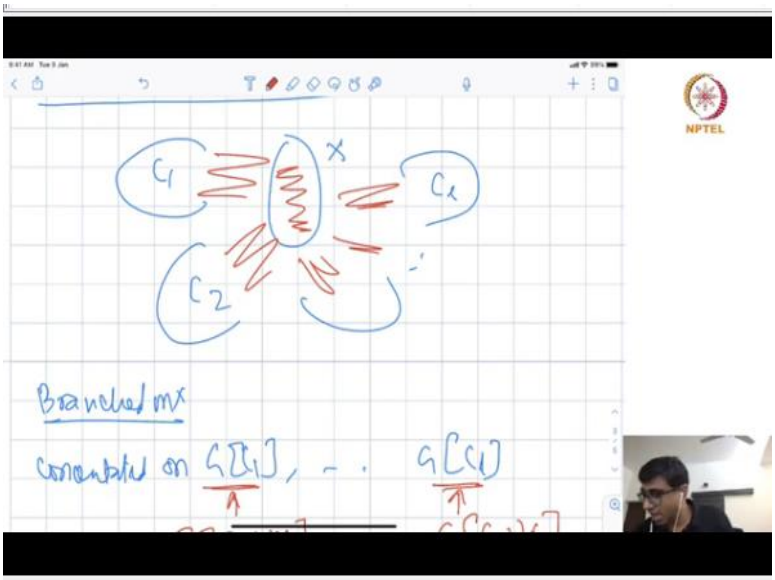
Basically you take the union of all these vertices involved in this sets that is a graph it represents and so association have a node t in V t to a subset of v G is called a bag. So, we will call a bag so basically tells look at this tree every vertex in tree is associated with a subset of my graph a vertex subset. Now, look at this tree rooted at somewhere and look at any vertex do you have a notion of subtree rooted?

Now, I asked myself when this itself was a tree then the meaning was that this is a subtree rooted at t, this part of but what part of the graph am I considering, this is basically the graph which is union of graph inducer union of vertices which are appearing in the bags which are associated below this vertex V in the tree. So, there is a natural association of what graph to consider.

Now you may ask what are these bags. So, basically as I was trying to tell earlier, look, when we are separating this S. So, this function will basically try to take a piece of the graph and associate that set S to this vertex v**.** So I say, look, now I am going to make a tree where I am going to make an association, I am going to separate S from here, let us associate let us make a node V in the tree and we will make all these things clear.

I am just trying to give a motivation how people might think about how to get a decompositions. So, this is the kind of things people try to do to associate and now let us get to. So, what is the natural way of going? So, for the MIS what did we do? We said okay fine.

(Refer Slide Time: 10:43)

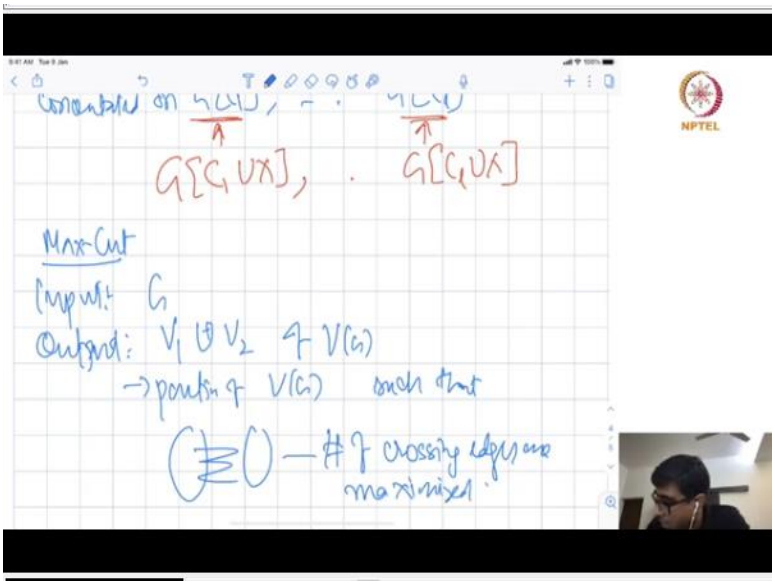


Look at a set x. And look at a connector component several of them say C 1, C 2 dot, dot, dot C n, what did we do when we were divide and conquer, we actually branched on x and concentrated on graph induced on C 1, but if you notice, there are also is this here, there are is this here. So, when we are trying to decompose our graph, it is important that the piece of the graph we are trying to decompose is also there.

Because look like one particular strategy worked for designing an algorithm for max independent set, but now given a graph, we would like to decompose such that there is an information about every vertex and edge in the graph in my decomposition itself. So, rather than when we get an x we say look rather than saying find a decomposition for C 1, C 2 this the way we will try to work is.

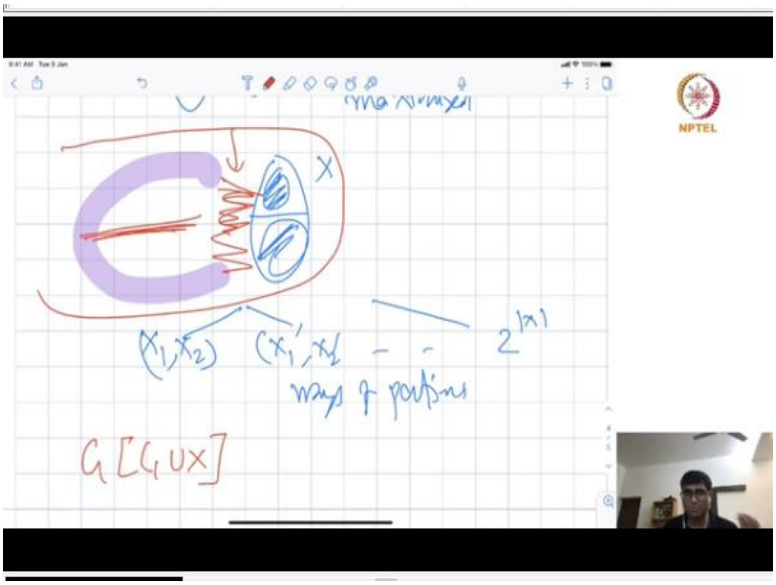
So, that every edge is counted in somewhere, because, look, for example, if you would have been doing a max cut example, just to understand that, why it is important that you do not just focus on graph induced on C 1 to C l. But actually, actually we should focus on graph induced on C 1 union x graph induced on C l union x, when we are trying to recursively decompose my graph into smaller, smaller pieces, why because x acts as a porter or as a barrier when going from any component to any component.

(Refer Slide Time: 12:42)



And say for example, support I was trying to solve a problem such as max cut, what is this input in G, output v 1, disjoint union v 2 of V of G, that is partition of v G, such that number of crossing edges are maximized. This is what is the classical max cut problem is.

(Refer Slide Time: 13:23)

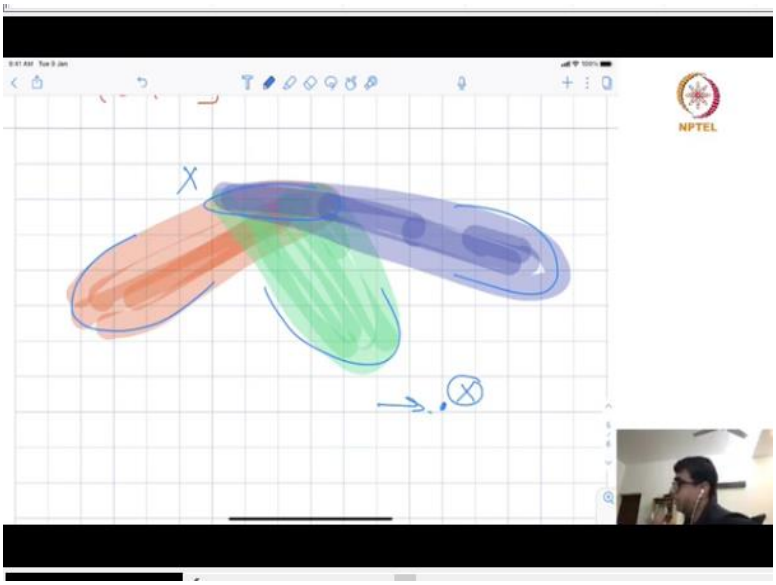


Now, if you find a set a X here like what will the natural way of branching? We will okay look at an optimal partition, an optimal partition could be in one part could be this, other part could be this. So, this is basically going from X 1, X 2, X 1 prime, X 2 prime basically 2 to the power mod X ways of partitioning. Now, given this partition notice look at a component here.

Suppose, I had a component here. Now, it is important for us that how these edges are there, look if I forget this red edges and find an optimal partition of this that may not lead to an optimal partition, I cannot combine a solution to this to a solution here, which because whether because this red edges are across the cut or not also depends on like because these red edges also needs to be counted in the cut and hence it is important for us that rather than recursively trying to solve just on the component, we actually solve the problem on graph induced on C 1 union X.

So, because of all these things it is important that whenever we are doing a graph decomposition, we make the graph decomposition as natural as possible as possible, so that you can exploit the decomposition of a graph to make or to solve a design algorithm for as many problems as possible and not only max independence. So, a natural way people go about doing is that whenever we are decomposing, it is exactly the same way that divide and conquer tells us okay fine. In a tree like fashion you do it, but now, what happens?

(Refer Slide Time: 15:31)

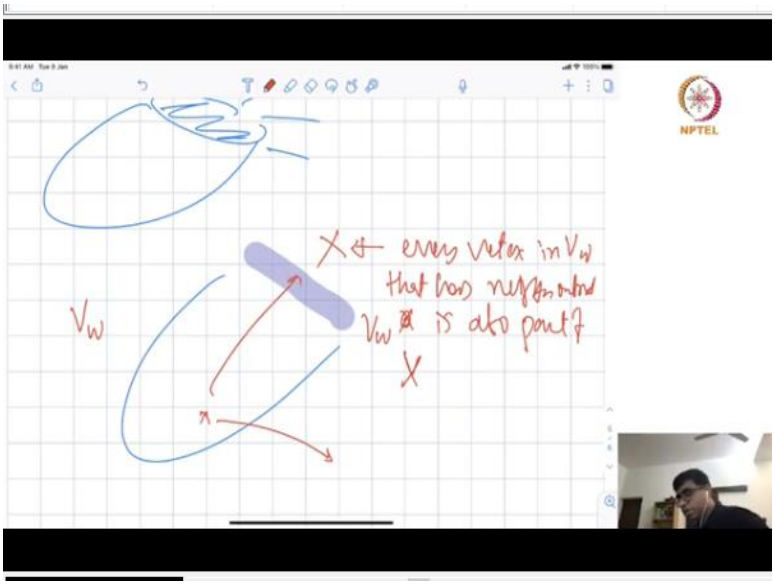


So, basically what we are trying to do is that we get the set x and then here are some components. Now, what are we going to do, we say okay, fine. Let us, decompose this piece of graph, this is one piece of graph here the common; let us decompose this piece of graph. So, and what is their common place is the set X.

So, once we have been able to decompose these pieces, we will put them together, but there is something like a natural question is whenever we are decomposition and whenever we are trying to make this tree, I told you that we should write the object which helps us in

decomposition. So, whenever I am making tree for this particular node, I will say okay, X is a set which separates the current graph.

(Refer Slide Time: 16:45)

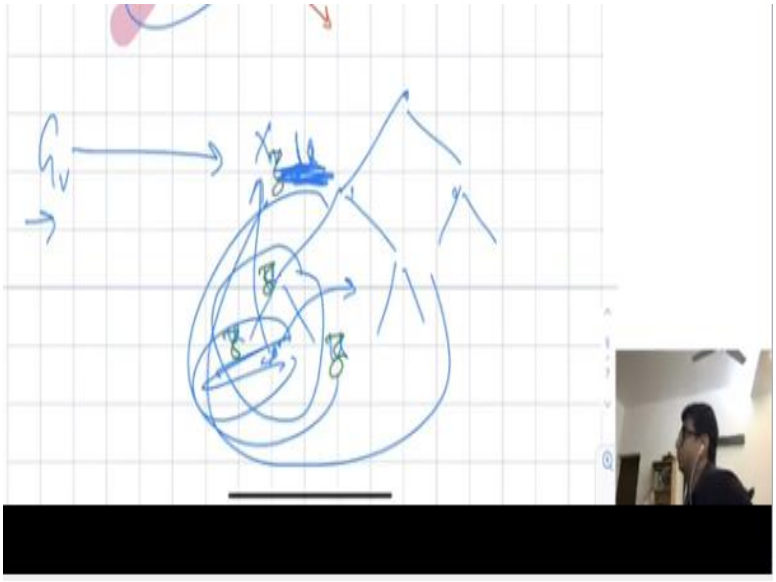


But notice that what is my objective. So, whenever I was doing DP or something, it is important that the set of vertices who has neighbour outside is part of my separator. So, in general what we try to do is that at any point of time when we are trying to have. So, we have a graph and we have a separator here and these are the only vertices which has neighbours outside.

Like, I mean rather suppose this is X every vertex say this is some V w, every vertex in V w that has neighbour outside X is also part of neighbour outside V w. If a vertex is here which has a neighbour to some other vertex then this guy must belong to the set X. So, this is the property of this. Now, notice that recursively when we have this piece of the graph, what will try to now, let us further split it.

Now if we further split it, it is possible that we find a decomposition this. So, now notice that a vertex which was previously used for cutting now can also be used for cutting further and what does it imply that if a vertex at any state if a vertex has a neighbour outside it remains even when we go recursively below, below, below, below.

(Refer Slide Time: 18:42)

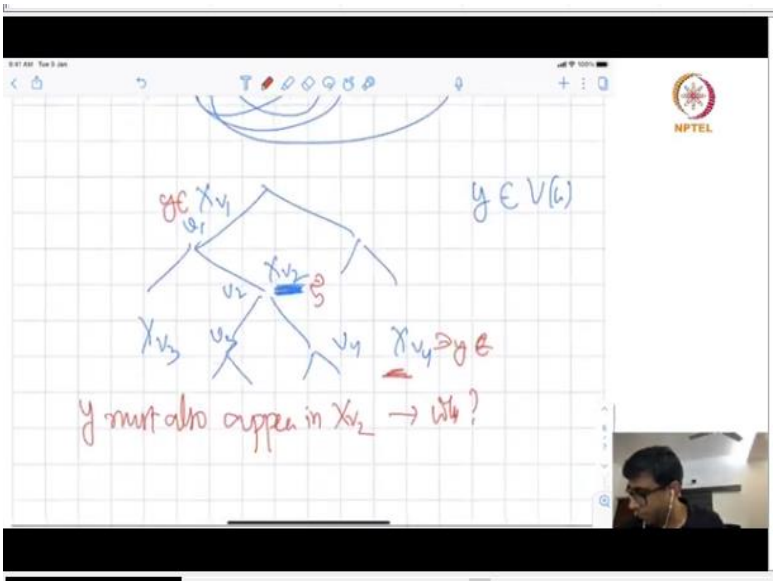


So now, because of that, if you notice, if I have a tree**,** what it tells us. This basically tells us when I look at the set, oh look at the graph below; these are the vertices which has neighbours outside. Now notice it means that wherever these vertices, now look at if I look at the graph here and if this vertex V also is here, it means this guy also has neighbours outside.

So, if all I a m storing at any point of time, suppose look at this decomposition, what am I storing at any node v of the street, I said look at the graph v and for any graph v I am containing a set say here X v and whatever property of X v? The property that any vertex any vertex, any vertex here, if it has an edge outside, it is a part of X v. Now, notice that it implies look ahead, I just told you a same vertex could appear several times in helping us in separating.

So, every time it appears in separating it could enhance as well as when once it becomes a part of boundary, it becomes boundary. It means what happens? If V appears here, V appears like look at some vertex, look at this set, let us I think I have used, I think I should not use the vertex, I think we should use ser and some Z here X z. Now, the point which I am trying to tell you is that it is very natural because of all this that if some vertex look at, so, what would we associating to.

(Refer Slide Time: 20:35)



So, what I want to say that look at what we are associating towards tree, some subset of vertex. So, suppose this is v 1, v 2, v 3, v 4 and suppose here we have associated X v 1, X v 2, X v 3, X v 4 and suppose some vertex y of v G appears here. And now suppose y appears here then my claim to you is that y must also appear in X v 2 y as I told you. So, at any point of time the set of vertices basically what are we storing here is a set of vertices from below who has a neighbour outside.

So, if this y already has a neighbour outside so, like it is going to contain all these vertices. Now, look at X v 2, if y is here, it means y also belongs to the graph induced on this, it means y also has neighbour outside which implies y should also belong to this. In some sense, this is exactly what he is trying to do. So, basically what I am trying to tell you that we need some property from these bags where what is the property that if some vertex appears in several bags.

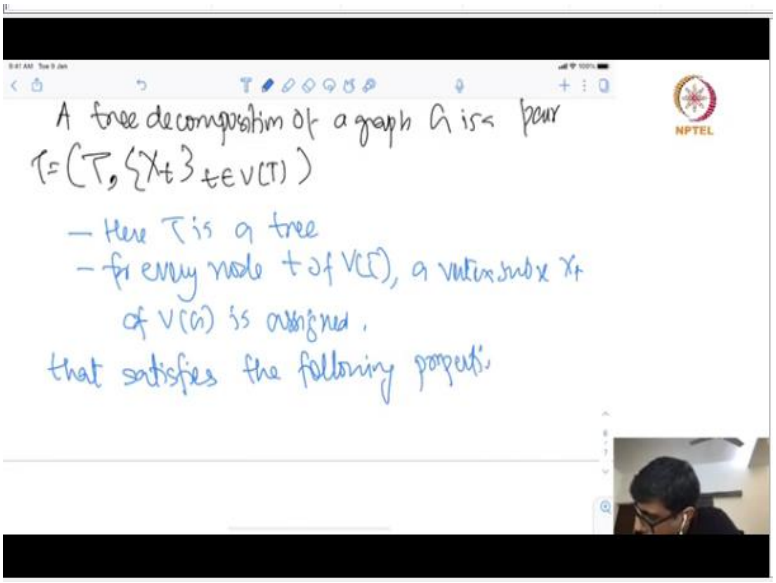
Then look at the set of vertices that forms of the tree which corresponding bags this particular vertex appears, then they should not be disconnected, because keep in mind, I am just giving an intuitive picture. So, what are we trying to store, at any point of time I have a piece of graph and the set which I am storing at this particular vertex is those vertices in this set which has a neighbours outside.

So, if a vertex had a neighbour outside before it will remain, so, if this vertex appears in any smaller into sub graph, so on and so forth. So, there are few specific property that we need to maintain. And to do all these things, people came up with this notion of treewidth, which I

am going to define now. So, up until now, it was just like at least whatever I tried to give you is the set of properties said to be should be X that we should expect from the treedecomposition or the treewidth or the decomposition which we are going to define now. Okay, okay. Okay.

So, let me not waste my time and formally define these 3 things. And once I formally define these things, try to go back to the 2, 3, properties that I have been talking about up until now and see where do they fit in. So, let me formally define this.

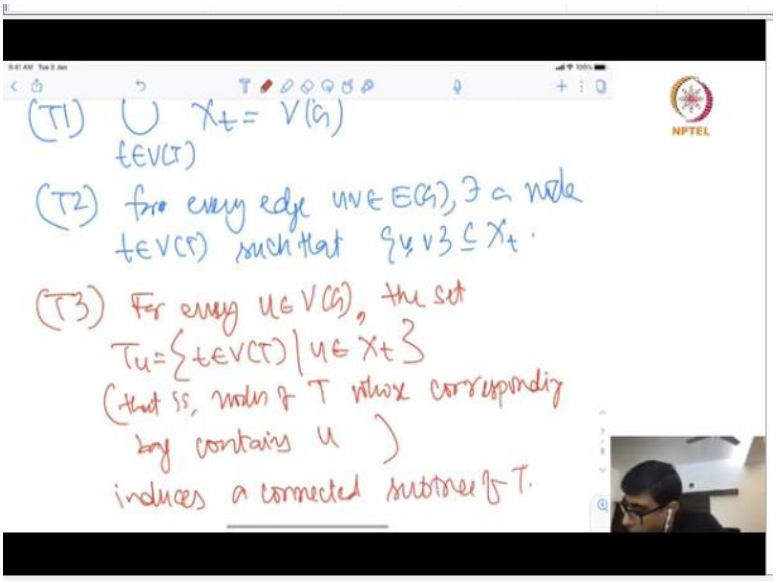
(Refer Slide Time: 23:38)



So, what is a treedecomposition of a graph G? So, treedecomposition of a graph G is a pair, what is a pair t = X t in V T. And what is this property, what is this? Here T is a tree for every node t of vertex set of T of vertex subset X t of vertex set of G is assigned. So, basically the way I have been explaining up until now.

So, basically a treedecomposition graph is a pair or tree and X t which is like for every node you are defining what we call associating a subset of vertex set and what are the properties that it like it satisfies tree some very, very basic property, that satisfies the following properties. So, what are the basic properties?

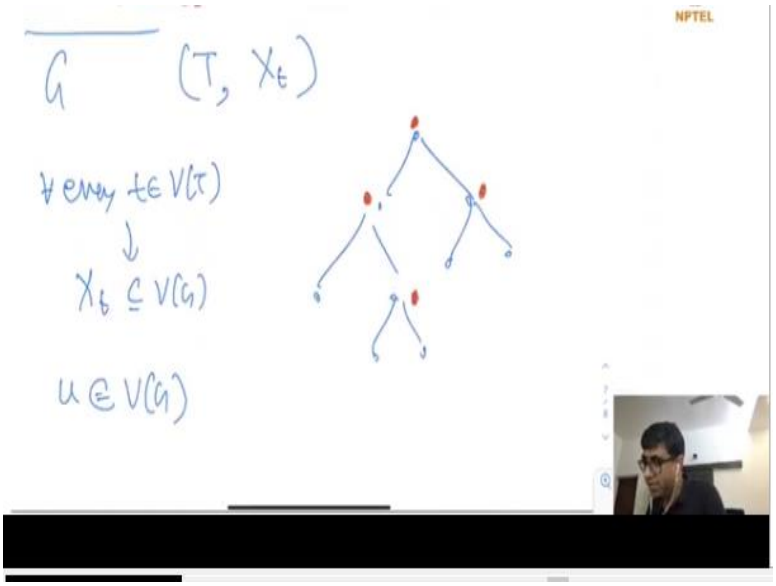
(Refer Slide Time: 25:43)



So, the property first is treedecomposition properties that if you look at union t in V T and you look at X t this is equal to vertex set of G. Like every vertex appears in some bag and it makes sense I mean T 2 is that for every edge u v in E of G there exist on node T in vertex T such that u, v is a subset of X t. Meaning for every edge u v there is some bag that contains both u and v.

This is also like every edge occurs somewhere and finally and this is an important property that we have been talking up until now and what is T 3? For every u in v G the set T u what is your set T u? It those nodes in vertex set of T such that u belongs to the corresponding bag that is nodes of T which corresponding bag contains u. So, for every node the set T u induces a connected subtree of T. So, what is the meaning of this?

(Refer Slide Time: 27:54)



So, what it basically says is that we will come to that. So, you have a graph G for a graph G I am going to associate a tree and some X t. So, basically think of this X t I mean which I use a function f before and that are people actually use the function beta to represent this, but like basically I have a graph for this graph I have associated a tree and for every t in V T, I have associated some X t, which is a subset of vertex set of G and what your property?

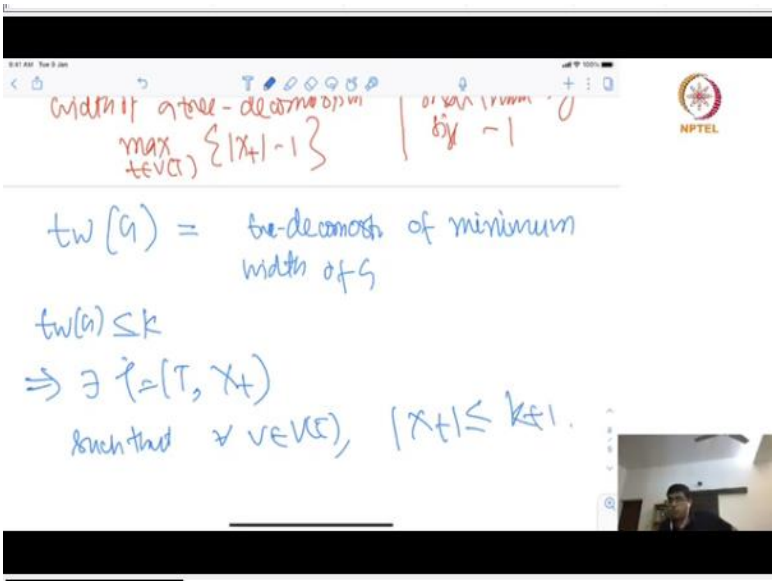
If you give me a vertex V there is a V is going to appear in some X t or some node here, if I am going to give an edge u V it is going to occur in somewhere. But the last property tells let us look at a fix of vertex u. Now, let us look at all those nodes which corresponding look let us look at u in V G. Now, let us look at all those bags that contain u. So, suppose it contains this, use part of this.

Now, look at the tree induced on red color vertices that must be connected. So, for example, the picture which I have drawn here is not right, because look at the tree induced on this tree red vertices it is disconnected. So basically it means that even this vertex must be included. And as I told you we will see a proof but basically, it is like we are trying to decompose via separators some pieces of graph.

And once you become special and different special if you appear in some separator at some point of time, then you remain a special because I mean, why are you separated because the reason you were separated and I mean so we will be only dealing with a minimal separator. Because you are separating someone from somewhere, it means this particular vertex must have some neighbour outside. So, basically every vertex that appears in separated is kind of external for that piece of the graph.

And once your external for that piece of the graph you remain external for every piece of the connected set in which you are part of the below and hence, you are part of them, which implies that once you are not like you are gone. So, this is important to maintain this property and you will see that how this connectivity property of the subtree is an important be exploited algorithmically. So, that was to give you some basic now. So, this is a definition of treedecomposition of a graph. So, tree decomposition of a graph is a pair T, X t which satisfies some property**.**

(Refer Slide Time: 30:57)

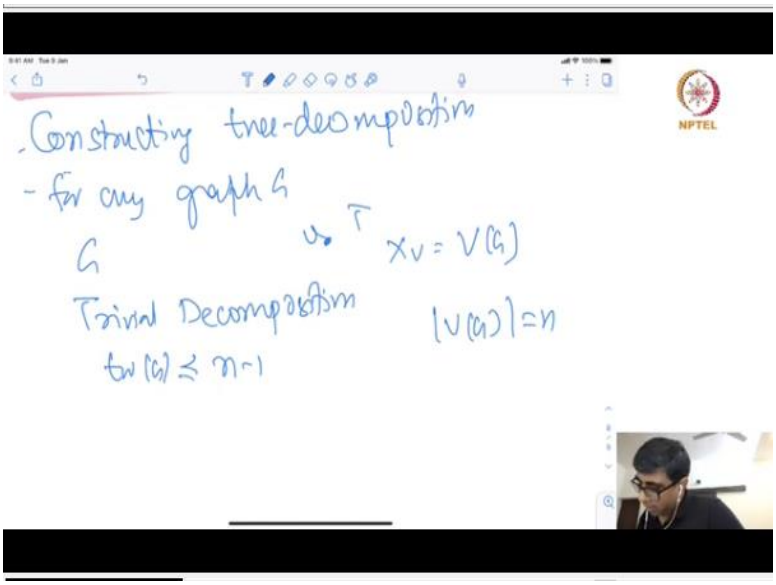


And what is a width of a treedecomposition? Width of a treedecomposition is equal to max over t in V T. So, basically width of decomposition is maximum bag size minus 1 and the reason why there is a -1 here to classically mean that tree should have treewidth 1. Now you may ask so what is a tree decomposition of a graph? So, tree decomposition of graph G is basically is a tree decomposition of minimum width of G.

So, for example, right treedecomposition is a minimum width of G. So, given a graph G I could have several decomposition of that graph, but treewidth is find the decomposition where the width is the smallest, where no bag is larger. So, whenever I say treewidth of G is at most k implies there exist T T, X t such that for all V in V T size of X t is upper bounded by k + 1**.** So, basically I can find a treedecomposition of my graph where no bag has size more than k + 1.

So, having said all this let us try to make our hands dirty before and we will also prove some structural properties later on is that let us try to construct some treedecomposition.

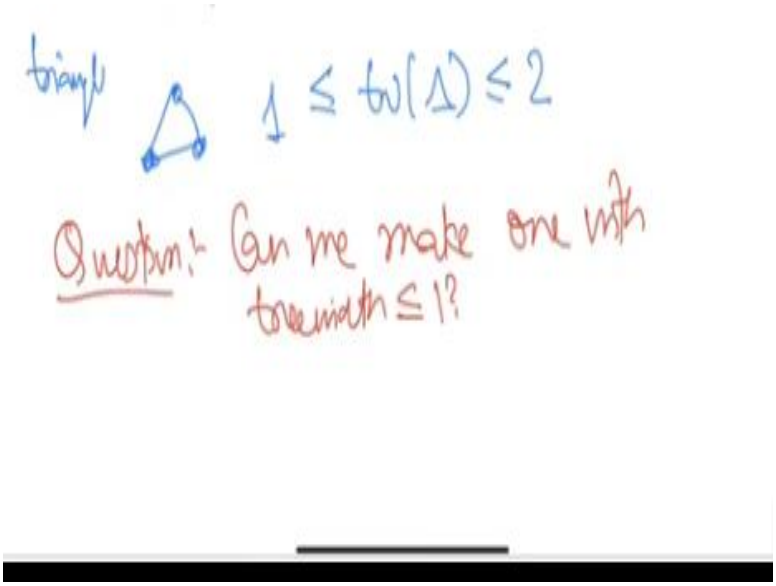
(Refer Slide Time: 33:23)



Constructing treedecomposition. So, for any graph G has my treedecomposition. So, graph G you have single vertex V. So, my tree consists of single vertex and support this is V X v equal to vertex set of G. Definitely it satisfies all the property every vertex appears, for every edge there is a bag containing this and definitely for any vertex it appears only here, so, it is a connected thing.

But this is like a most trivial decomposition. And so for any graph G what does this imply? The tree width of G is at most n – 1 if vertex set of G is such because you have added everything. So, we do know how to compute a treedecomposition of a graph.

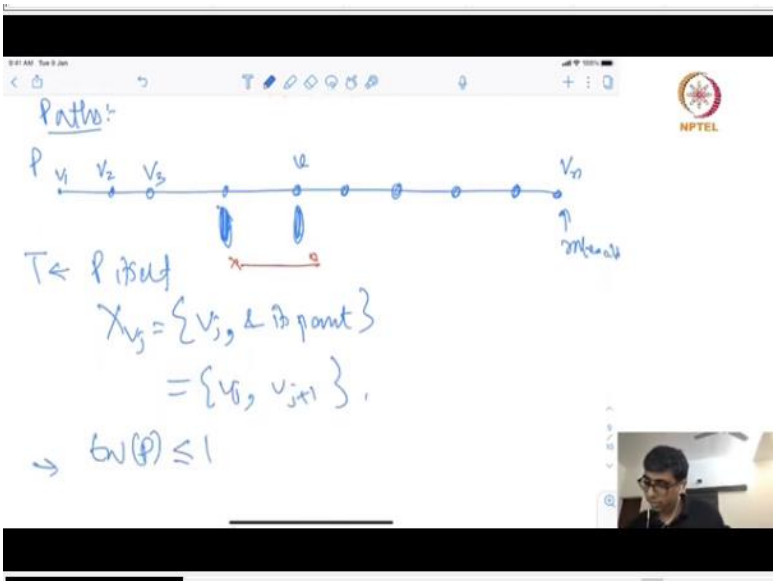
(Refer Slide Time: 34:41)



Now, let me take a suppose this is my graph triangle so, if I make a trivial decomposition, this is a trivial decomposition that implies that the treewidth of a triangle is at most 2.

Question can we make one treewidth at most 1. And notice that because we need a bag, which contains an edge, which implies that the treewidth of triangle is at least 1, because you need to have a bag containing at least 2 vertexes. So, any graph which has an edge, the bag must contain 2 vertexes, in particular, a back containing the endpoint of a particular edge, which imply that the treewidth of a graph which has at least one edge is at least 1. So, I know the treewidth of a triangle is at least 1 at most 2. Is it 1? It is a good question please try. What about path?

(Refer Slide Time: 36:25)

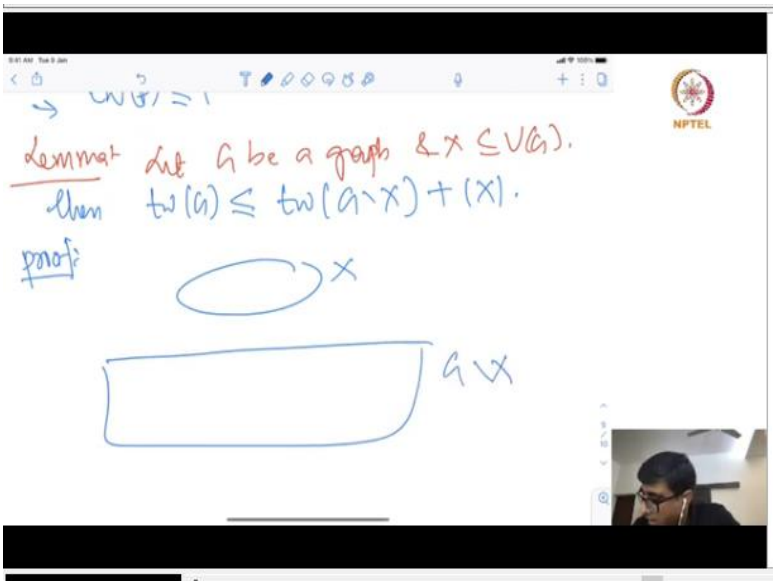


Remember the way made our DP algorithm. So, this is our path and suppose path as a T, I am going to take P itself and support this v 1, v 2, v 3 and v n. So, I am going to add X v 1 as v 1 and so think of this v 1 routed at v 1. So, what is X v 1 going to contain X vj, let us say X vj is going to contain vj and its parent**.** So, this is going to be vj, vj + 1. Now let us see if this satisfies all the property? Well, yes

Every vertex appears in someone. And now look at about edge, look at any age V i, V i + 1 that edge is part of the set X vi. Now, look at a vertex V, it is a part of which one? So, look at a vertex V, this is going to be part of a bag, which is here and it is going to be a part of the bag which is there. So, these are the bags in which this particular vertex appears, but that is an edge. So, this is connected.

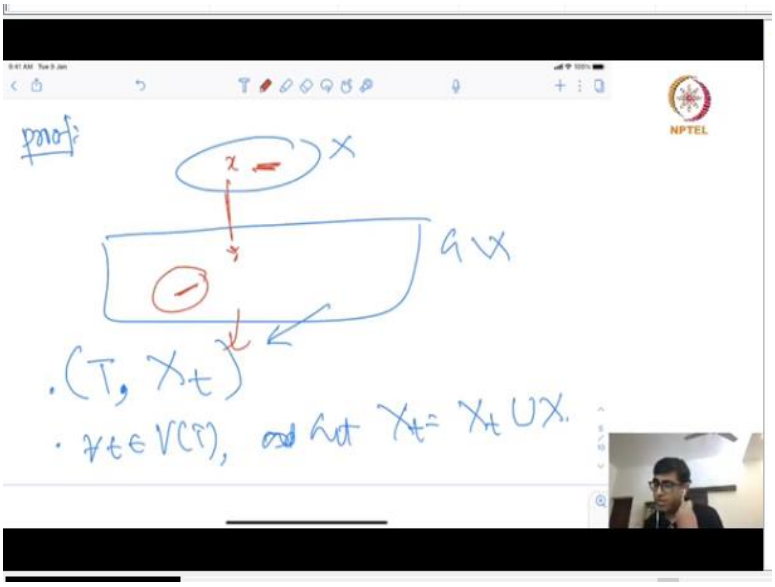
So, this is a valid treedecomposition. And what is the width of this treedecomposition? Treewidth of path is at most 1, because every bag has size at most 2 that is great. Now let us prove another nice result.

(Refer Slide Time: 38:29)



Small lemma. Let G be a graph. And X V subset of V G, then treewidth of G is less than equal to treewidth of G - X + cardinal X. Let us prove, this is easy. So, here is my X and here is my G - x. Now, for this G - x suppose we have tree T and some X t for G - x. Now what will we do?

(Refer Slide Time: 39:20)

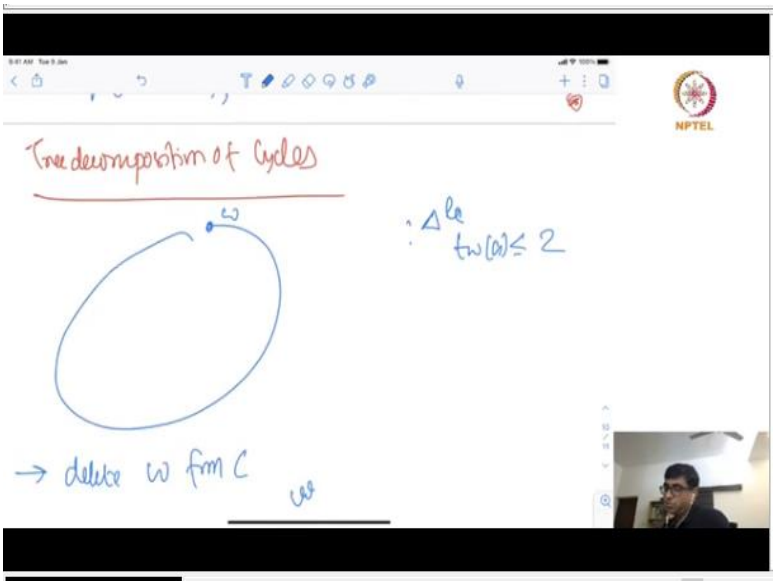


Say for all t in V T let X t = X t union X. Now, that is it. Now, let us see why every all the 3 properties are satisfied. Look at every vertex. Definitely, already every vertex of G - x appeared in the X t and every vertex in x appears everywhere. So, this is perfect. Now look at an edge, edge. So there are kinds of sedge where the edges? Edges are either here, here or here.

These kinds of edges are anyway taken care of because this is a treedecomposition of G – x. What about edges which are going across, but now look at some way this vertex V appears in X t, but in that you have also added the whole of X. So, V and that x is also taken care of and since x is in every bag all the agents from here is also being taken care of. Now, look at the connectivity property of a vertex.

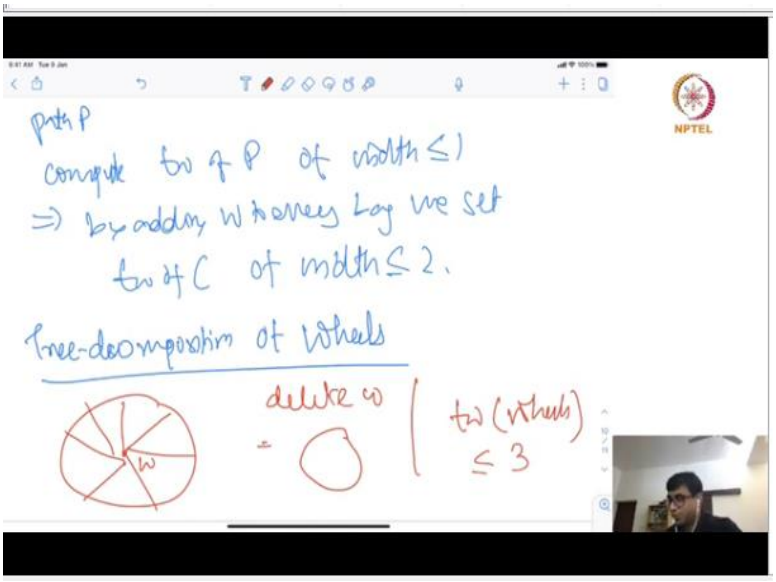
Look at the vertex in G - x that is like all the bags in which it occurs it is already in connected because T, X t is a treedecomposition of G – x. Now, look at a vertex x. In x, but that x occurs in every node. So, that is since T is a tree it is already connected. So, this is a very who just tells us that if you have a tree graph G and you have a such x then if G – x has the small treedecomposition. Then G – x and then the G itself could have a small treedecomposition. So, now exploit this lemma.

(Refer Slide Time: 41:15)



What about the tree decomposition cycles? Fine. For triangle could give is at most 2. But now let us see what we can do. Suppose, delete w from C, what we are going to get then this is the path P, for which and has the w.

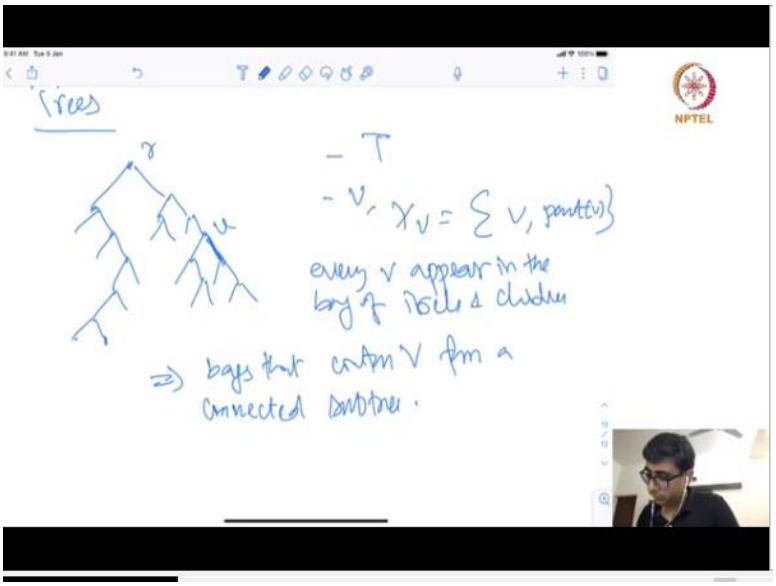
(Refer Slide Time: 42:03)



For path P we know how to compute a treedecomposition of P of width at most 1, which implies by adding w to every bag we get treedecomposition of C of width at most 2, because we just apply the previous lemma. Now path has treedecomposition of width 1, then you add whatever, so this is how we could add layers to further.

What about treedecomposition of wheels? What are the wheels? It is a cycle and they just poke. What did we prove treedecomposition of cycles has width at most 2 and support this is a w, delete w, what are you going to get cycle. So, treewidth of wheels is at most 3, because 2 here and you added 1. So, look at this, how interesting this lemma is.

(Refer Slide Time: 43:32)

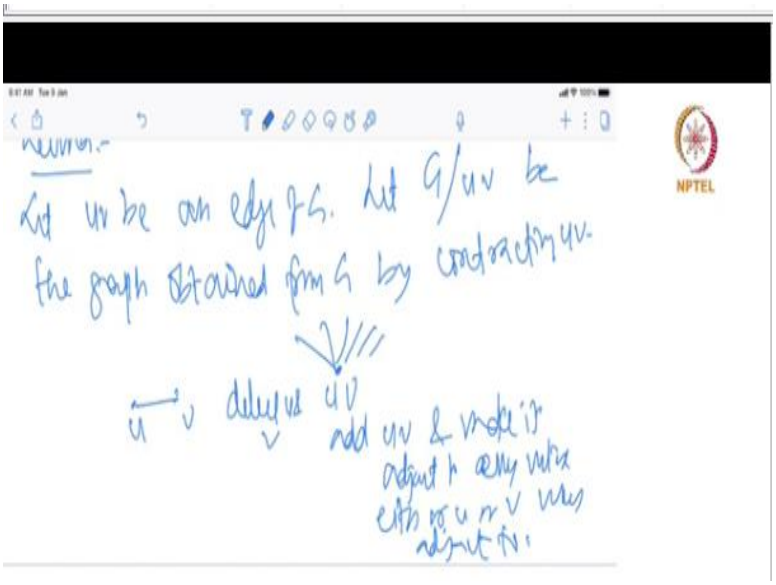


Now let us do more things. And let us say now let us do trees. Trees are no different than paths, but we will see. Suppose this is my tree. So, I just routed some vertex route. So, my T

is going to be same. And for any vertex V, X v is going to be V and again parent of V. That is it, you are done. Now by definition every vertex appears in some of the bag. Look at any edge.

So, for any edge, one is child, one is parent, so look at the bag of the child in that you have this, so this is perfectly fine, no problem. Now what about a vertex V? So, look at the vertex V where all our vertex V appears? A vertex v is going to appear in all the bag of children and itself and children. What does that imply? This implies that bags that contain V form a connected subtree. Now, so what about grids? So, now we have done.

(Refer Slide Time: 45:41)

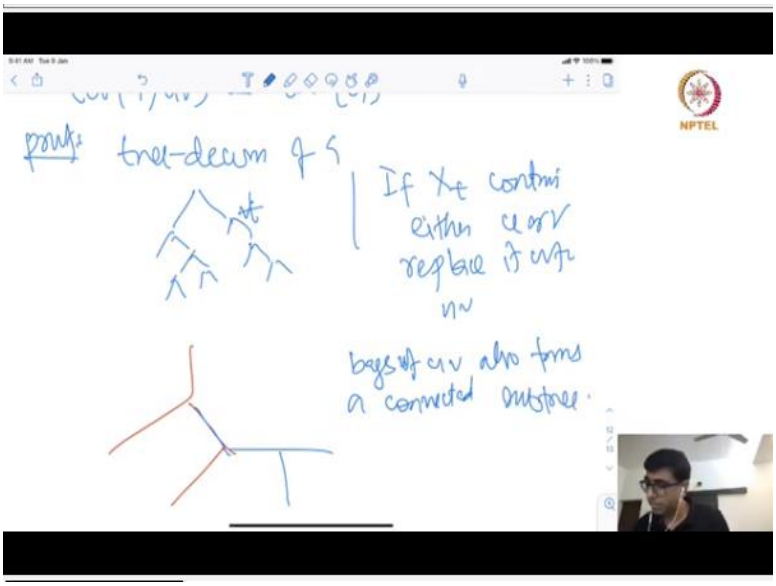


So, now what about this is my lemma? Let H be a subgraph of G, treewidth of H is less than equal to treewidth of G. This is very simple, you take the treedecomposition of G and so treedecomposition of G is also a treedecomposition of H. And you are done, because every vertex appears already in G, every H appears already in G, if a vertex is connected to the vertex is connected still, so this is perfectly fine. So, this is great. So, treewidth is a property which is close, under taking subgraph.

So, now what about another lemma let us try to prove? Let uv be an edge of G and let G contracted uv be the graph, obtained from G by contracting uv. What is contraction means, if you recall correctly. So, basically, what is the contraction, this is an edge. So, basically you make a new vertex uv and make this guy adjacent to all the edges which are adjacent to uv, so you basically delete u and v, add new vertex uv and make it adjacent to every vertex, either u or v was adjacent.

So, what are we saying, we are going to say that the tree width of, so this is another operation, which does not increase the treewidth and proof is very simple.

(Refer Slide Time: 48:02)

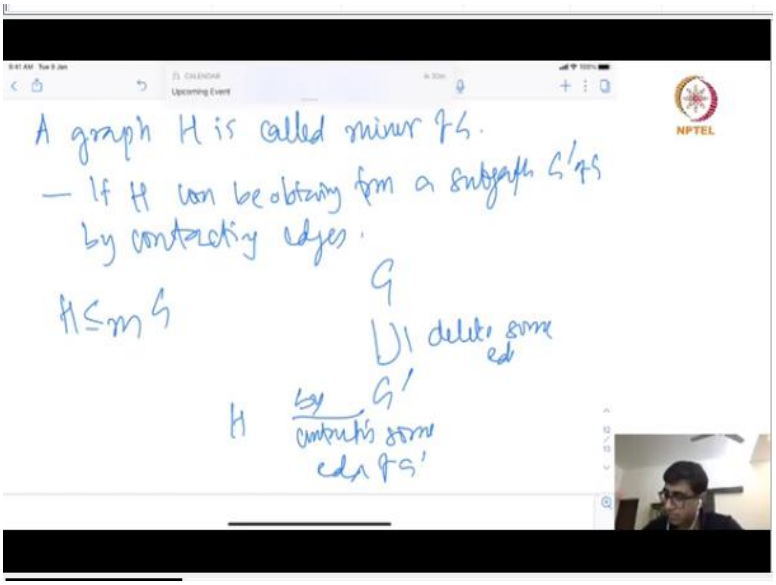


Now, what is treewidth of G? So, treewidth of G is like so tree because, so look at the treedecomposition of G and if some T, if X t contains either u or v replace it with uv. So, it might just contain u or it might just contain v, then you still add uv. If it contains both uv you delete both uv and just replace it with uv. So, notice, you have not increased the bag size. What about edge property?

Well, I mean edge properties anyway satisfied because now what about uv. Look, what about the connectivity property? So, look at the connectivity of u and v, look at u. So, u is going to be some subtree which is connected and look at the v that is going to be some subtree, which is connected. But the blue subtree and red subtree must intersect, why they must intersect, because look uv was an edge.

If uv was an edge if uv was an edge there is a bag that contains both u and v, it means there is a bag which is common in the red bag and the blue bag. So, now if the 2 connected things intersect, they are connected in itself. So, which implies that the bags of uv also forms a connected subtree. So, here is my exercise.

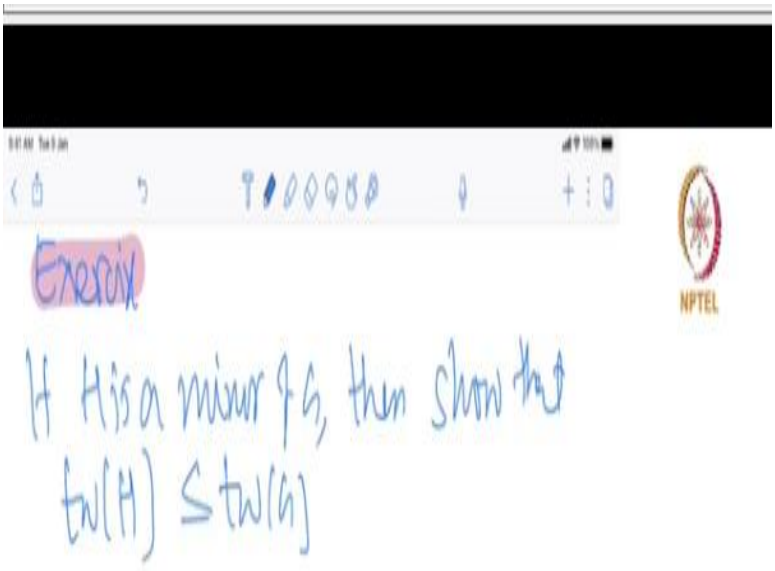
(Refer Slide Time: 50:15)



So, let me define a graph H is called minor of G. If H can be obtained from a sub graph G prime of G by contracting edges. So, what is this tells us, I am going to call H is a minor of G. If you can get H as follows. So, from G first you get a subgraph G, so basically you delete some edges. So, G is delete some edges. So, this is one. And now you should be able to get by contracting, some edges of G prime.

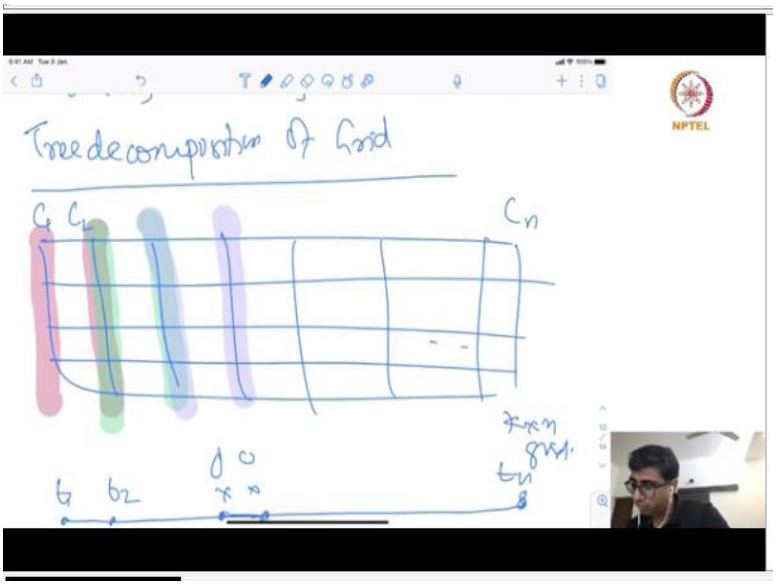
Then I am going to call H is the minor. So, a graph H is called a minor of G, if H can be obtained by following operation you first take a subgraph of G and then you contract some set of edges.

(Refer Slide Time: 51:24)



Here is an exercise for you. You should be able to prove. If H is a minor of G, then show that treewidth of H is at most treewidth a G. So, the last example which I want to make or construct is grids.

(Refer Slide Time: 52:09)

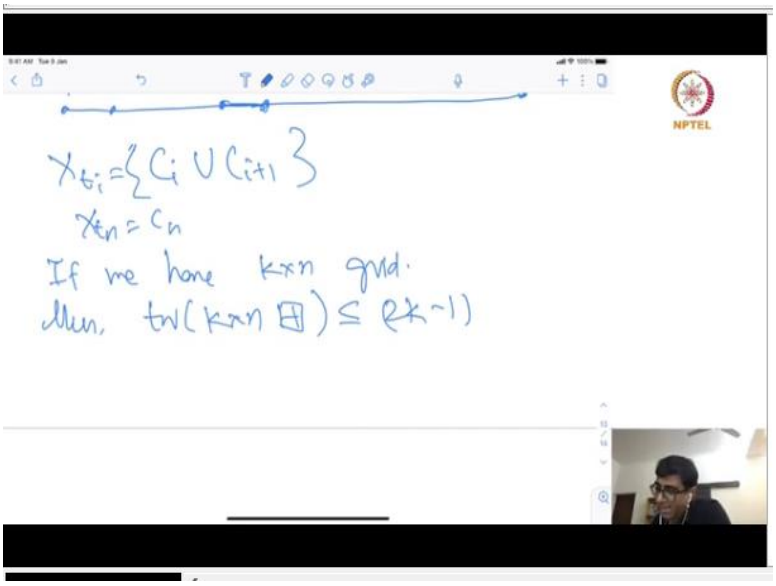


So, suppose I have this k times n grids and I have column 1, column 2 and column n. So, today I am going to give you a path not even tree. So, what is my path is basically consists of a path on n vertices v 1, so let us since we are using t 1, t 2, tn and what is X ti is going to be column i union column i + 1. This is it. So, t 1 is going to consist of what is going to basically consist of this set and this set. And what I am going to associate to t 2. I am going to associate to t 2, column 2 and this.

What I am going to associate to column 3. Like the T 3 is basically so on and so forth. Now, definitely and you might ask what about X tn, so for X tn I should write X tn is nothing just column n. Now notice definitely every vertex appears in some. Now look at any edge. If it is an edge which appears in a column, some particular column itself then that occurs or any other edge is between some column i and column i + 1.

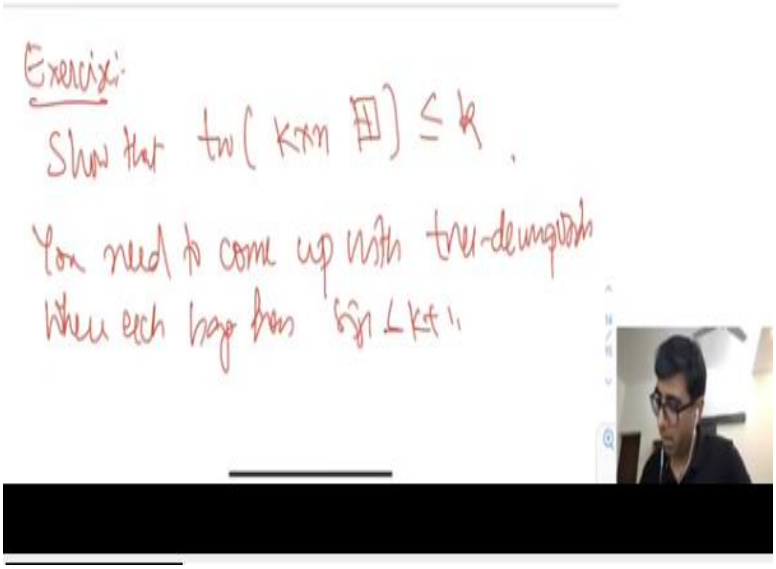
And I know that C i and C i + 1 are together in C i. So, that is done. And what about look at a vertex, a vertex appears in only 2 bags. Like a bag could consist of either support, if a vertex appears in column C i, then this vertex is going to appear in the column the node corresponding to ti - 1 and ti itself. So, like basically every vertex appears on 2 consecutive nodes on my path and hence their bags. Hence, they constitute. So this is done. So, what are we able to show?

(Refer Slide Time: 54:34)



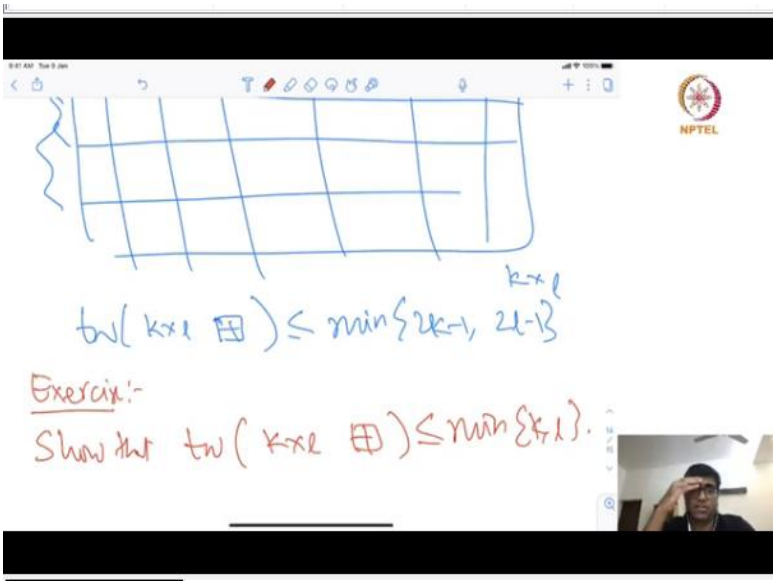
So, we are able to show if we had k times n grid. Then treewidth of k times n grid. Let us denote grid by grid is at most to 2k -1, because, 2k 1. So, this we are able to show. So, here is my question, exercise.

(Refer Slide Time: 55:09)



Let us see. Show that treewidth of k cross n grid is at most k. It means you need to come up with treedecomposition where each bag had size at most k + 1. Try this. It is a simple and easy exerciser and before I end my today's lecture let me make a last comment. My last comment is that we assumed that we have a k cross n grid.

(Refer Slide Time: 56:10)



But actually, suppose we had some k cross l grid we could also make another treedecomposition where we take these 2 together, then these 2 together and these rows together, so that will imply that the treewidth of k cross l grid is actually minimum of 2k - 1, 2l – 1 and here another exercise. So, that treewidth k cross l grid is actually minimum of k, l. So, we will have another lecture later for this week, where we will show some structural properties of this treedecomposition.

Up until now we have seen how to construct what is treedecomposition and I have given you some fairly several examples of construction of treedecomposition and I will show you how people use this treedecomposition in parametrized algorithms in the next lecture, as well as give you some structural properties of the treedecomposition and that should be all for now.