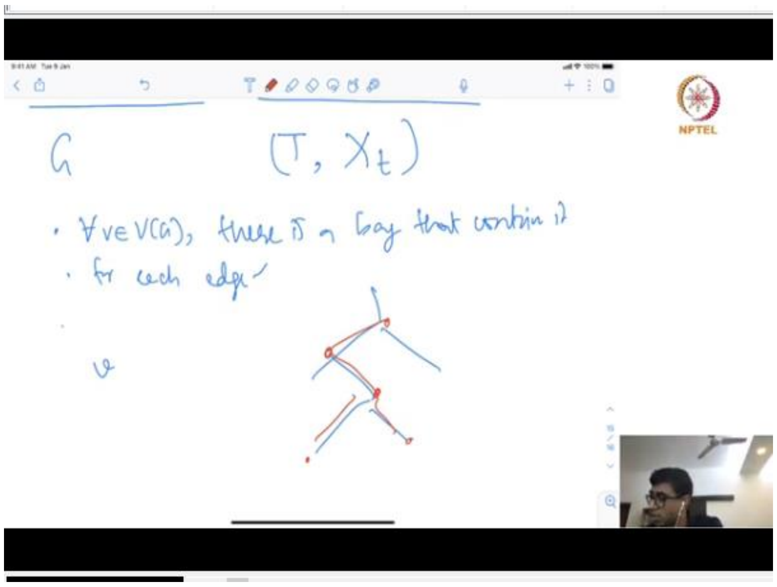
Parameterized Algorithms

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

Lecture-24
Structural Properties of Treedecomposition and Win-Win

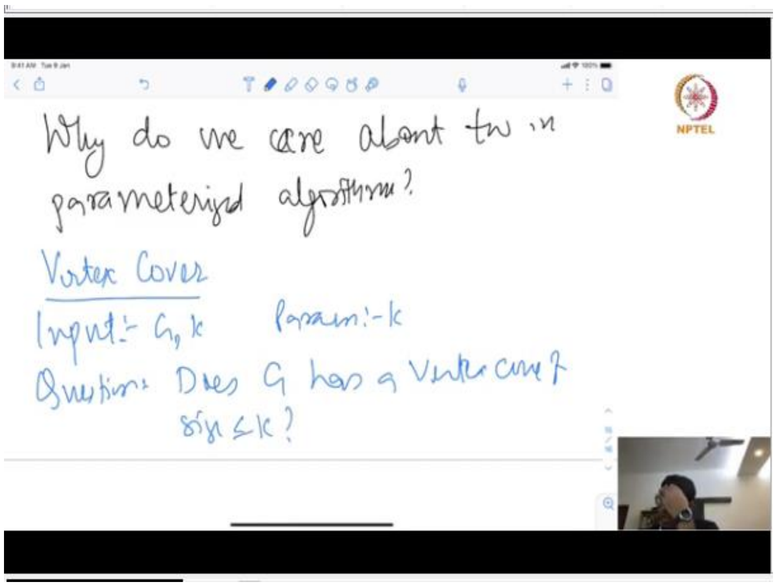
(Refer Slide Time: 00:17)



Welcome to the last lecture of week 5 and in this lecture we will see some more properties about treewidth and treedecomposition that we refined in the last lecture. So, just for a recall what was the treedecomposition of a graph G was a pair T and a set of bags X t associated with every loaded T, with the property that for every vertex in V G there is a bag that contains it similarly for each edge**.**

And what we also saw that look at this tree T and for any vertex V look at the bags that all the bags that contains the vertex V. If you look at that then they form a connected sub tree. So, these are the 3 properties that we were looking for the treewidth and treedecomposition and up until now we saw for several graph classes that we can construct a treedecomposition of small width.

(Refer Slide Time: 01:28)


Why do we care about tw in parameterized algorithms?

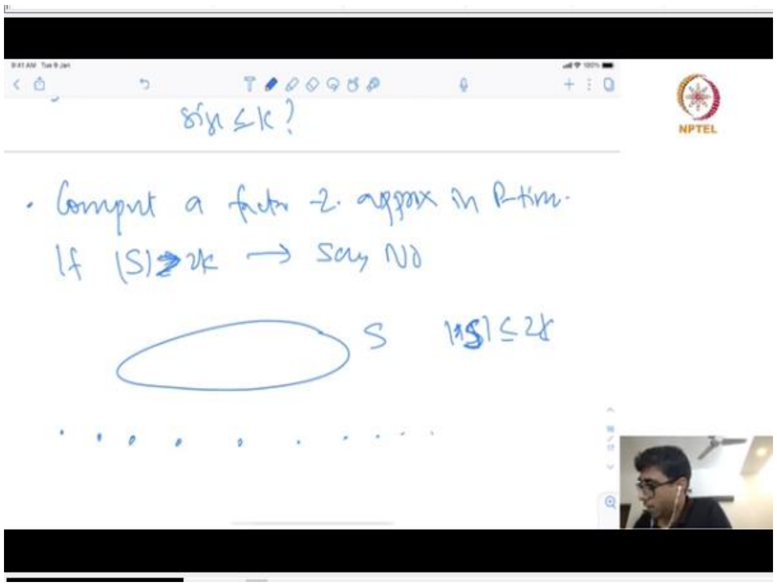
Vertex Cover

Input:- G, k Param:- k

Question: Does G has a Vertex Cover of size $\leq k$?

And but first let me today in our lecture like I am going to talk about is why do we care about treewidth in parameterized algorithms? So, let me try to answer this question first. So, let us look at the first classical problem that we have studied vertex cover**.** So, input was G, K parameter is K, question does G has a vertex cover of size at most K**.** Now here is a way to deal with it so how can we deal with this algorithm problem is following.

(Refer Slide Time: 02:31)


size $\leq k$?

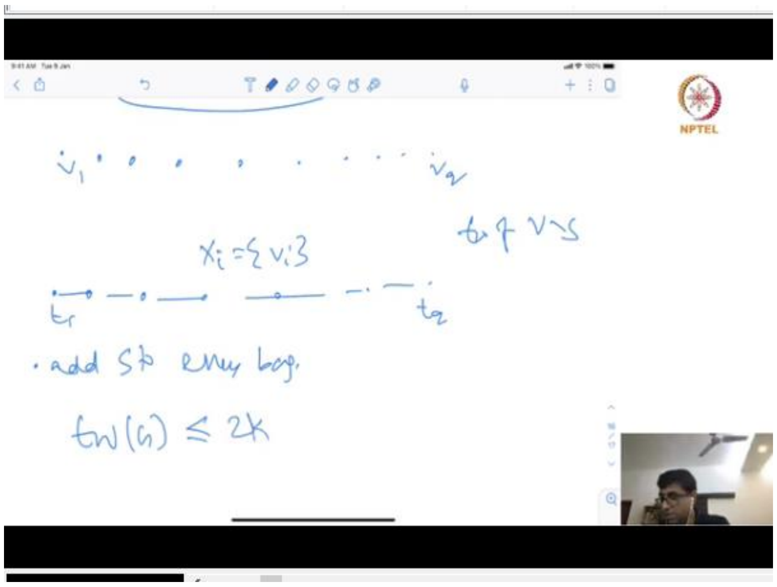
- Compute a factor 2 approx in P-time.

If $|S| > 2k \rightarrow$ Say NO

S $|S| \leq 2k$

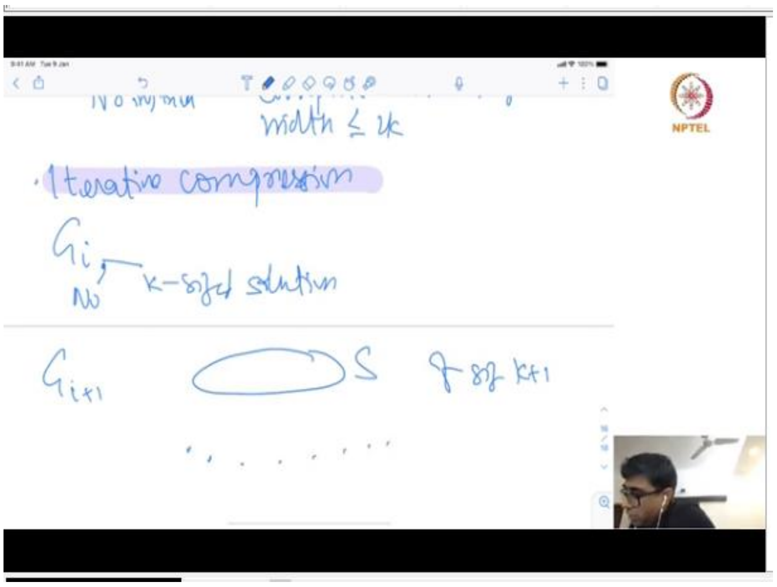
So, compute a factor 2 approximation in poly time in p-time, if size of solution is strictly more than 2k say no or else what do you have? You have a set S who has size at most 2k and what is this? This is an independent set.

(Refer Slide Time: 03:08)



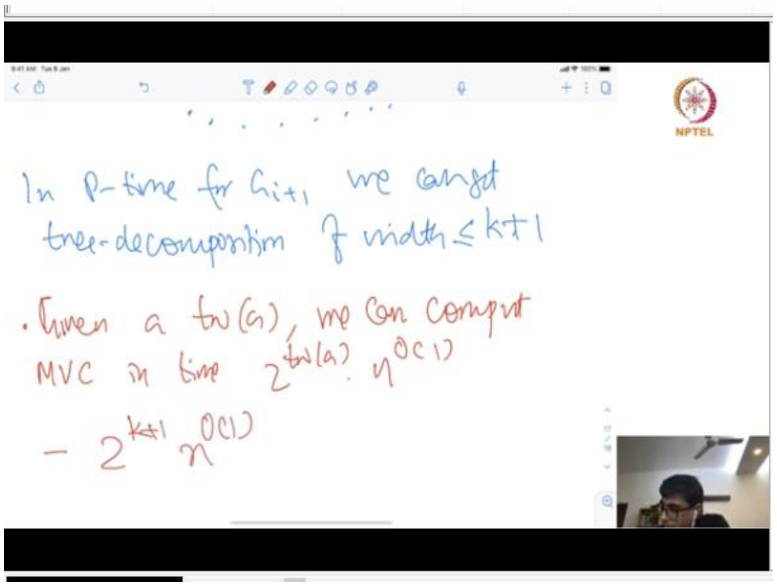
Now this is our poly time, so what we can do, you make a path, suppose this is a vertex V 1 to V q you make a path on t 1 t q and X i is just v i. Now so this is a treedecomposition of V - S. Now add S to every bag what have we got. So, we are able to show that the treewidth of G is at most 2k. Now so what are we able to say?

(Refer Slide Time: 03:48)



So, in poly time we are able to say either this is a NO instance or able to compute treedecomposition of G of width at most 2k. In fact we can prove much, much better, much, much more how look, remember when we do iterative compression. Now in the iterative compression step you have G i. So, either you are able to say that answer is NO or you have a k size solution. In G i + 1 you have a vertex at S of size k + 1 so that everything is independent.

(Refer Slide Time: 04:39)


In P-time for G_{i+1} we can get tree-decomposition of width $\leq k+1$

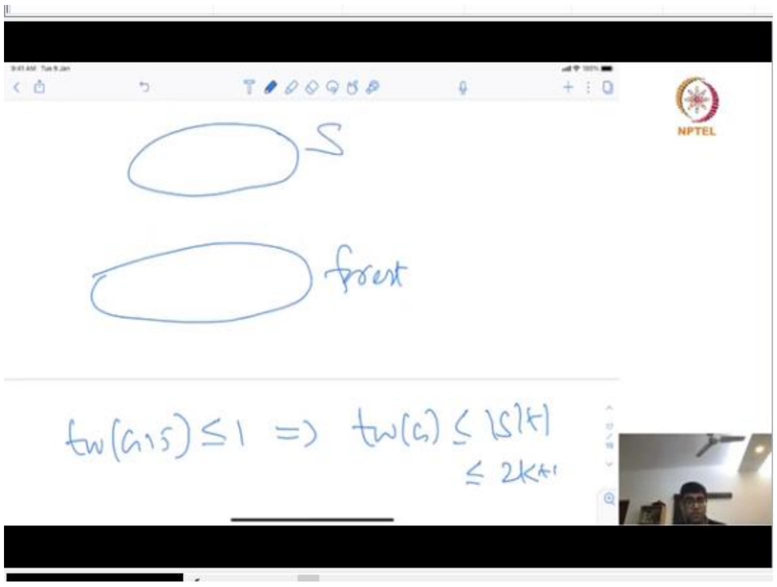
· Given a $tw(G)$, we can compute MVC in time $2^{tw(G)} \cdot n^{O(1)}$

$- 2^{k+1} n^{O(1)}$

So, in fact in p-time for G i + 1 we can get a treedecomposition of width at most k + 1. So, all I am trying to tell you that this is how, this is why we care about and it is possible to show which we will see in the next week is that what is the possible given treedecomposition of given we can compute minimum vertex cover in time 2 to the power treewidth to the power G n to the power O of 1 we will see this.

But notice so what are we able to do? We are able to show in poly time that either this is a YES or NO instance or we are able to come up with a treedecomposition of the graph and we can explore the treedecomposition to dynamic programming or compute some other kind of algorithms and get an FPT algorithm. So, this is another route to design an FPT algorithm for vertex cover in running in time 2 to the power k + 1 into poly time.

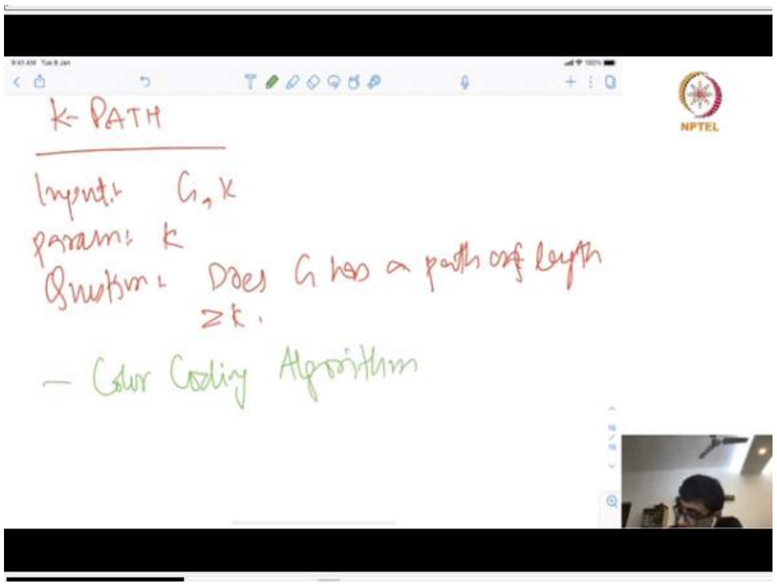
(Refer Slide Time: 06:05)



$tw(G \setminus S) \leq 1 \Rightarrow tw(G) \leq |S|+1$
$\leq 2k+1$

Same story will hold for feedback vertex set. So, what will happen for the feedback vertex set? You will get a set S again you can compute a factor 2 approximation and this is a forest so and we know that the treewidth of G - S is at most 1 which implies treewidth of G is at most mod S + 1. So, if you are able to bound S by 2k you are able to come up with 2k + 1 or if you apply this iterative compression idea then you are able to show that treewidth of G is at most k + 1 or something**.**

(Refer Slide Time: 06:46)



k- PATH

Input: G, k

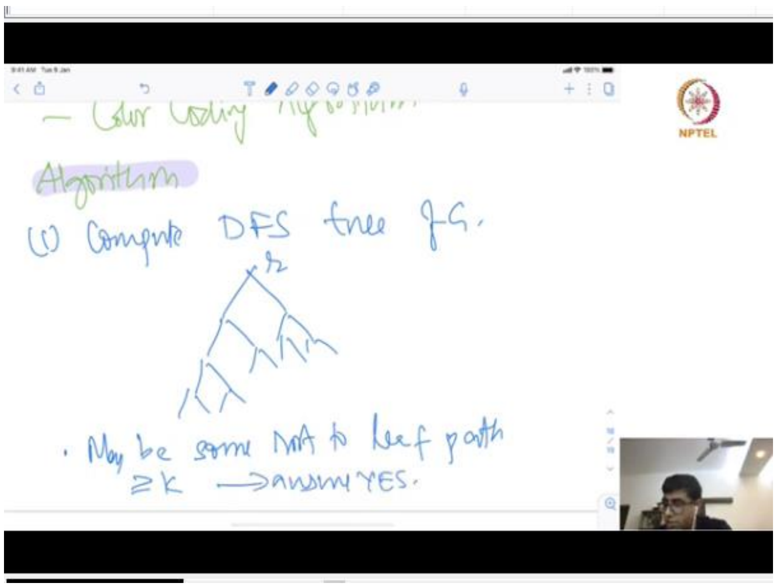
param: k

Question: Does G has a path of length ≥ k.

– Colour Coding Algorithm

So and again if we had a dynamic programming algorithm for feedback vertex set parameterized of treewidth then we are done and actually in fact such algorithms exist and we will see some of these algorithms then we will be start to do algorithmization of our treedecomposition. So, these are the 2 properties which I would. So, now let me tell you some other property. So, this is like a longest path or k path whatever it is called input is G, k, parameter is k, question, does G has a path of length at least k**.** And for this you must have seen in the last week a colour coding algorithm. Now let us see a magic using treedecomposition.

(Refer Slide Time: 08:05)


— Color Coding Algorithm

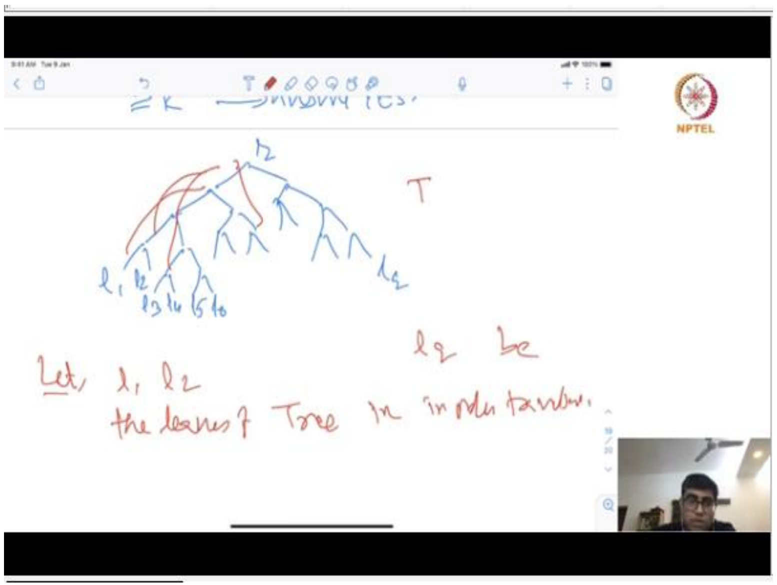
Algorithm

(1) Compute DFS tree of G.

· May be some root to leaf path
$\geq k$ $\longrightarrow$ answer YES.

So, what will be do? So, here is an algorithm, compute DFS tree of G. So, I have computed a depth first search tree. So, what could happen? So, if suppose I started at root maybe some root to leaf path is already length greater than k answer yes.

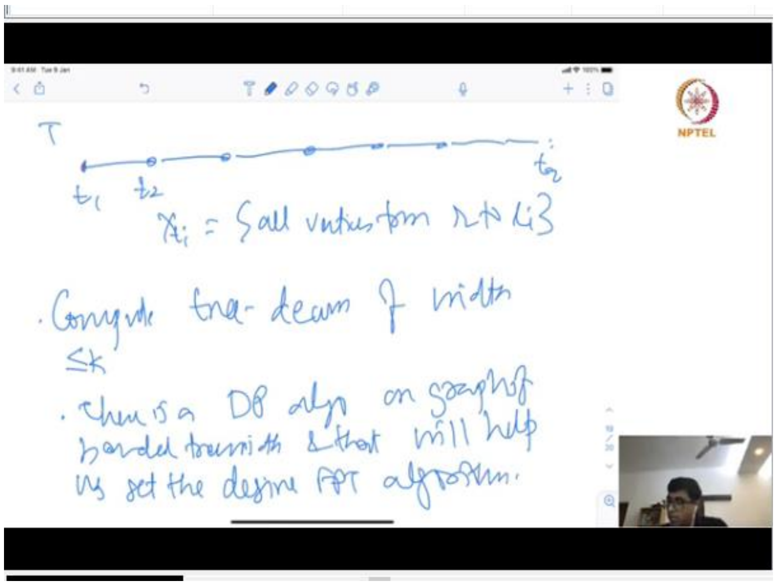
(Refer Slide Time: 08:53)


$\geq k$ $\longrightarrow$ answer YES.

Let, l_1, l_2 ... l_q be
the leaves of Tree in inorder traversal.

But suppose it did not happen then what can you say look at this one, look at this here is a root and suppose we have l 1, l 2, l 3, l 4, l 5, l 6 so and so forth. Now what is the property of depth first search tree? Look at any non tree edge where can the non tree edge be? Every non tree edge must be some root to ancestor like this, like this**.** Now you let l 1, l 2, l q be the leaves of this tree in order traversal fix something does not matter. And now I am going to give you a path, so now I am going to give a treedecomposition this graph what is my tree composition of graph?

(Refer Slide Time: 09:58)


T

t_1 t_2 t_q

X_{t_i} = {all vertices from r to l_i}

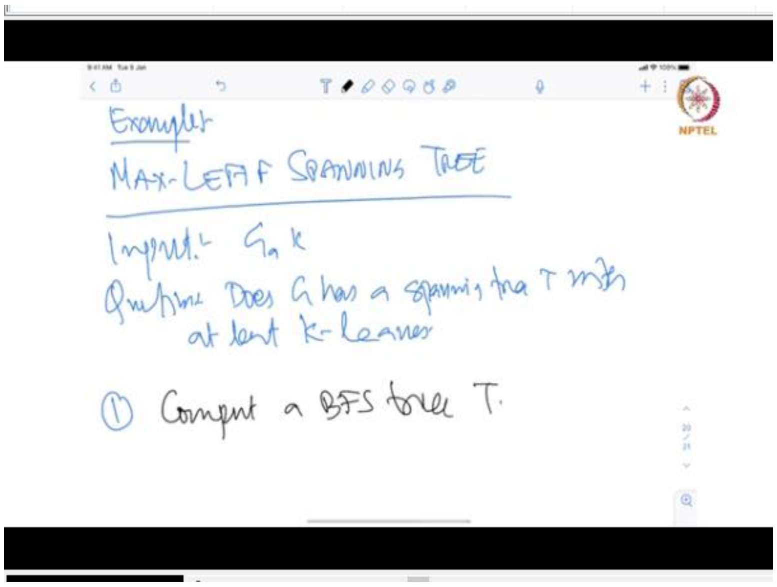
· Compute tree-decom of width ≤k

· there is a DP algo on graphs of bounded treewidth & that will help us get the desire FPT algorithm.

It has q vertices t 1, t 2, t q vertices, and what am I going to add, so look at the t i, so X ti is going to be all vertices from r to leaf. So, basically my first guy is going to be this is my first bag, second bag is going to be this is my second bag, what is my third bag is going to be like this so on and so forth. So, now definitely every vertex is on the root to leave path so every vertex is there. Every edge it is on some root to leaf path, so every edge is there and now the only thing is what about the vertices?

Now there try to show this is an exercise using the properties of DFS that the vertices if the leaves are in pre-order reversal and then you fix this every vertex will appear in the connected fashion. So, you should be able to prove this. So, what are we able to show in poly time we are either able to say this is an YES instance or compute a treedecomposition or a compute a treedecomposition of width in fact at most k. And now there is a dynamic programming algorithm on graphs of boundary treewidth and that will help us get the desired FPT algorithm. So, let me give a last example.

(Refer Slide Time: 12:07)



Examples

MAX-LEAF SPANNING TREE

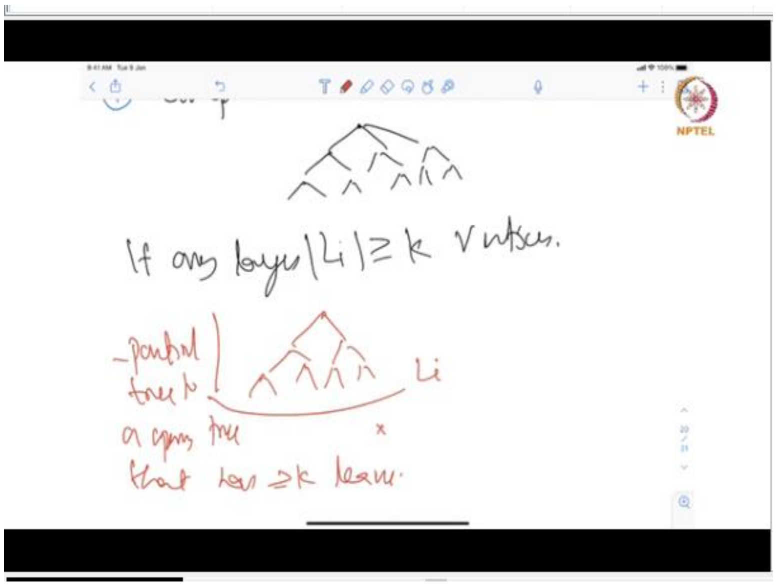
Input: G, k

Question: Does G has a spanning tree T with at least k-leaves

(1) Compute a BFS tree T.

And this is called max leaf spanning tree or rather max leaf spanning, input G, k question and the question is does G has a spanning tree t width at least k leaves. So, what will we do? So, algorithm does follow compute a BFS tree say T.

(Refer Slide Time: 15:14)

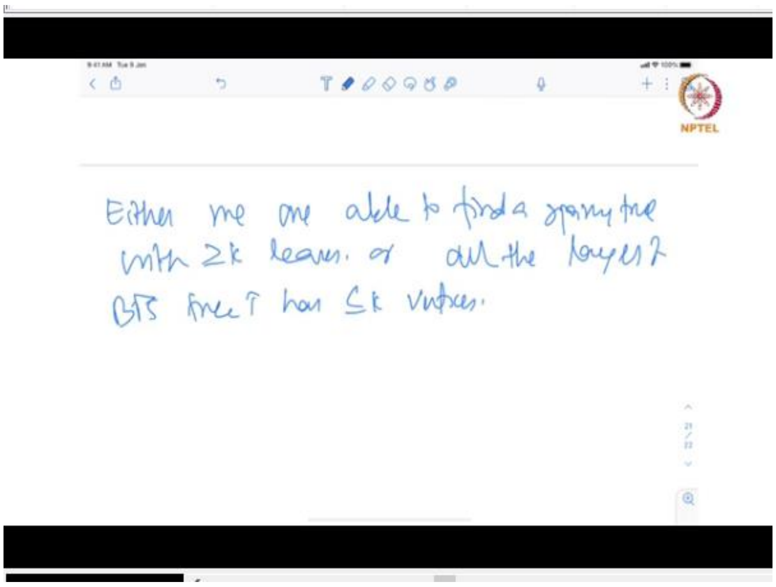


If any layer $|L_i| \geq k$ vertices.

partial tree to a spanning tree that has $\geq k$ leaves.

So, I have computed a BFS tree, if any layer L i has more than k vertices suppose there is a layer which has more than k vertices or greater. So, I have here is our with more than k vertices. Now look at this, this is suppose layer L i. Now because graph G is connected like either you can always extend this partial tree to a spanning tree that has at least k leaves. So, if any layer has more than k leaves then you know that you have a sub tree with at least k leaves then I am claiming if graph is connected. Then you can extend that sub tree to a spanning tree of my whole graph without sacrificing the number of leaves, it is a good exercise please try.

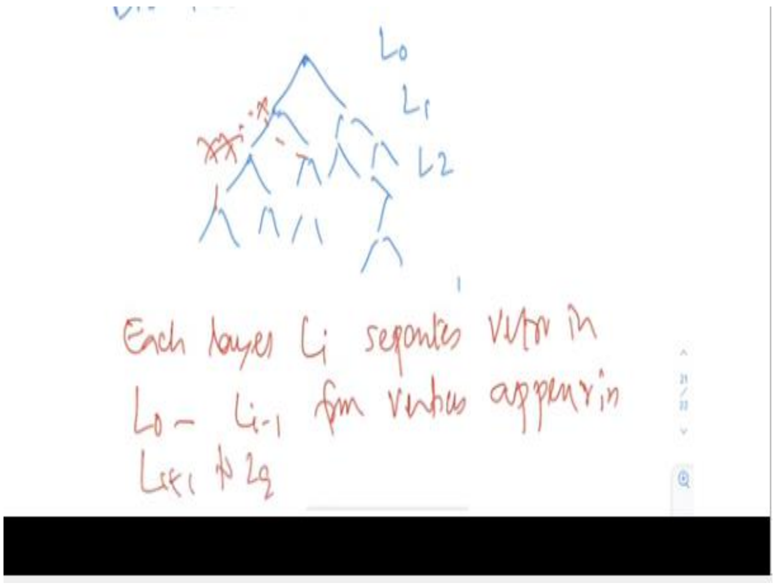
(Refer Slide Time: 16:42)



Either we are able to find a spanning tree with $\geq k$ leaves. or all the layers of BFS tree T has $\leq k$ vertices.

But what does this imply then**.** So, this implies that either we are able to find a spanning tree with at least k leaves or all the layers of BFS tree T has at most k vertices.

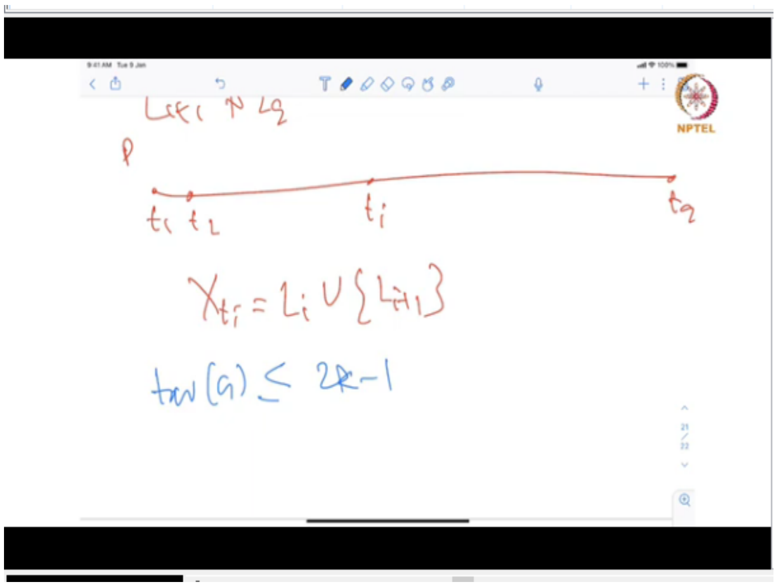
(Refer Slide Time: 17:18)



Each layer L_i separates vertices in $L_0 - L_{i-1}$ from vertices appearing in L_{i+1} to L_q

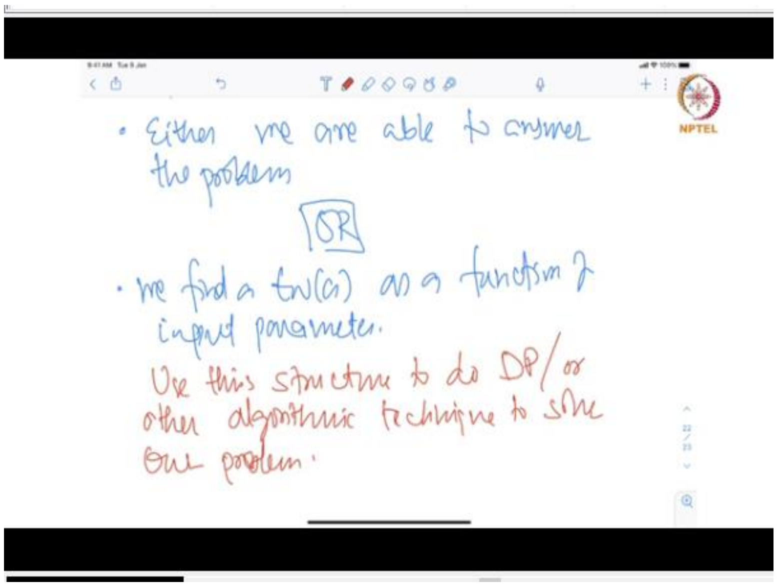
So, suppose L 0, L 1, L 2 so on and so forth. Now notice one thing what is the property of BFS tree? The property of BFS tree is that a node here can only have an edge here, it cannot jump here, there is no such jumping. So, in fact if you notice each layer L i separates vertices in L 0 to L I - 1 from vertices appearing in L i + 1 to say L cube some q. Now we can get a decomposition. Suppose there are q layers then you we can again get a decomposition.

(Refer Slide Time: 18:03)



Here is our path with t 1, t 2, t q and each t i is nothing but what, so X of ti is nothing but say L i union L i + 1. Again you can show that this is very easy to show that this forms a valid treedecomposition and in fact what is the treewidth of this graph then this implies is at most 2k – 1. So, and again if we had a dynamic programming algorithm and graphs are bound to treewidth then we can exploit the structure that we have got on the input graph to design an FPT algorithm.

(Refer Slide Time: 18:52)



- Either we are able to answer the problem

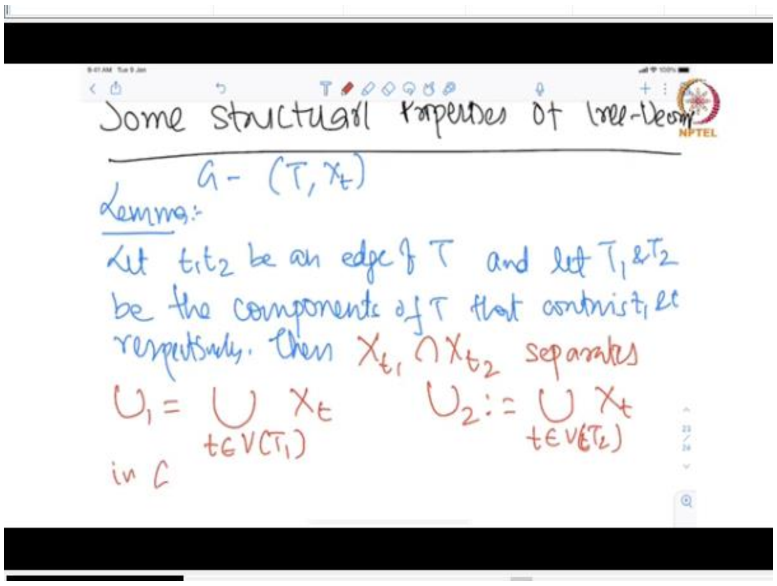
$\boxed{\text{OR}}$

- we find a tw(G) as a function of input parameter.

  Use this structure to do DP/or other algorithmic technique to solve our problem.

So, this approach is what is called win-win approach why? So, either we are able to answer the problem or we find our treedecomposition of G as a function of input parameter and then use this structure to do DP or other algorithmic technique to solve our problem. I hope this set of 4 or 5 examples would have explained you the importance of treewidth or treedecomposition in construction or in the designing of fixed parameter tractable algorithms. Now I will take a small D tour and spend some time telling you some properties.

(Refer Slide Time: 20:27)



Some structural properties of tree-Decomp

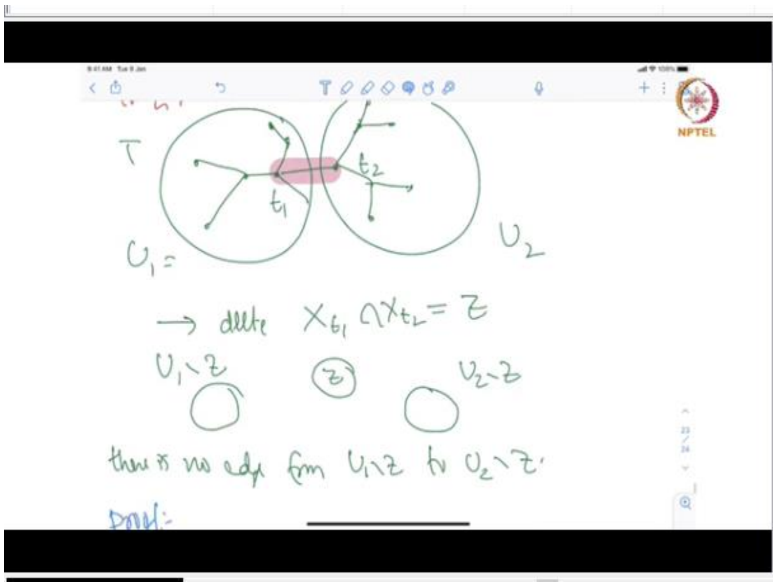
$G - (T, X_t)$

Lemma:-

Let $t_1 t_2$ be an edge of T and let T_1 & T_2 be the components of T that contains t_1 & t_2 respectively. Then $X_{t_1} \cap X_{t_2}$ separates $U_1 = \bigcup_{t \in V(T_1)} X_t$ $\quad U_2 := \bigcup_{t \in V(T_2)} X_t$ in G

Let me tell you some structural properties like of treedecomposition. So, let me tell you some interesting structural properties. So, let me prove some lemmas for you. So, from now onwards you have a fixed graph G and you we are talking about a fixed treedecomposition X t. Let t 1, t 2 be an edge of t and let t 1 and t 2 be the components of t that contains t 1 and t 2 respectively. Then X t 1 intersection X t 2 separates U 1 which is by definition what is U 1? Union t in vertex set of T 1 X t and U t is like union t in vertex set of T 2 X t in G. So, let me draw picture wise what I mean by this. So, suppose here is my tree.

(Refer Slide Time: 22:42)



$\rightarrow$ delete $X_{t_1} \cap X_{t_2} = Z$

$U_1 \setminus Z$ $\quad \boxed{Z}$ $\quad U_2 \setminus Z$

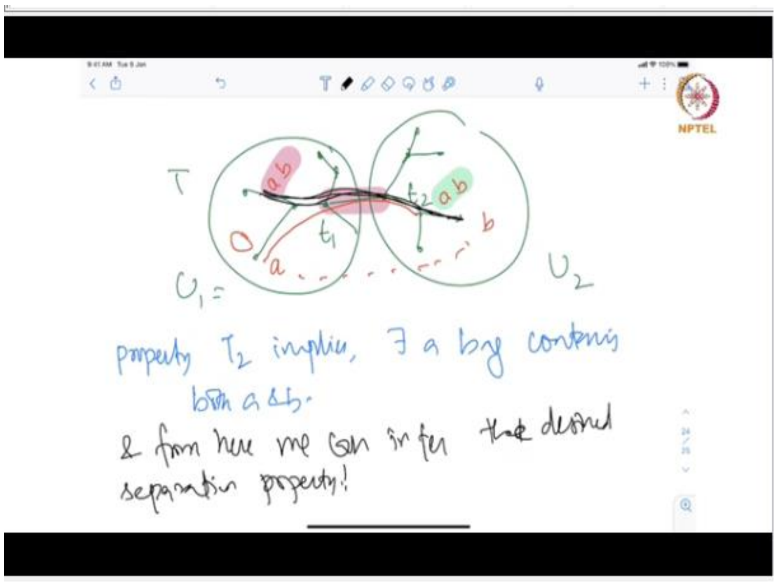
there is no edge from $U_1 \setminus Z$ to $U_2 \setminus Z$.

Proof:-

So, suppose this is my tree T and suppose this is the edge we are talking about. So, this is like t 1 and this is t 2**.** Now what is this meaning of this? It tells us that look at a graph where what is U 1? U 1 is basically union of vertices in the bags here and what is U 2? A union of vertices in the bags this and what it tells us that like if you delete X t 1 intersection X t 2, so this some vertex.

Now there is no edge suppose this is U 1 minus, let us call this Z and so this is my Z and this is U 2 minus, there is no edge from U 1 - Z to U 2 – Z. This is all that it says. Basically it tells us that if you look at the intersection of vertex set then it indeed forms a separation, it separates left part of my graph from the right portion of the graph. Now say why is this true, the proof is very simple just like one line why, hey so let us cut this copy.

(Refer Slide Time: 24:44)



property T_2 implies, ∃ a bag containing both a & b.

& from here we can infer the desired separation property!

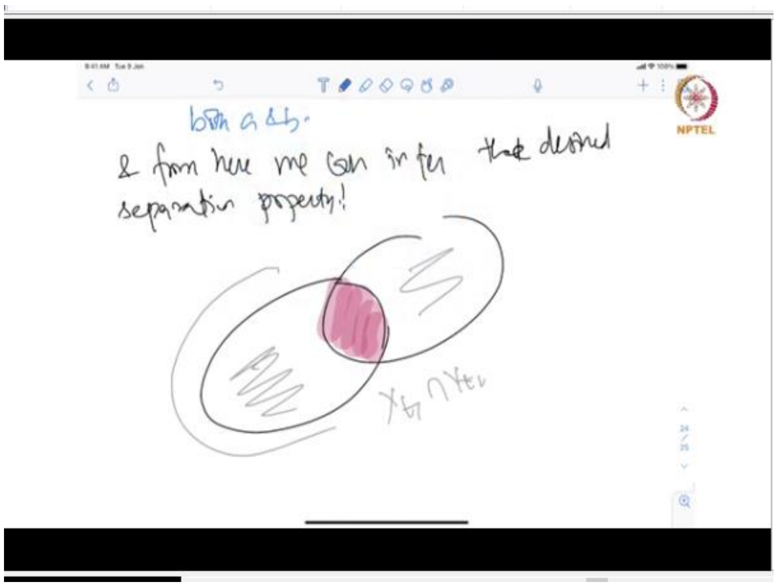
Now suppose there is some vertex here a and there is a vertex b and there is an edge here. Now what is the T 2 property implies? Property T 2 implies there exists a bag containing both a and b. So, where can that bag be? If that is a bag, so see b is in some bag in this side, a is here, so if suppose that bag is somewhere here let us say what happens? So, if this is a bag that contains both suppose it contains a and b then what do we know?

There is a bag here that contains a and the bag on the other side that contains a it means look at the unique path from here to here, all these guy will contain a, otherwise the bags containing a cannot be connected or similarly this bag could have been if this bag is say here and suppose a and b is here then we would have said well if that is the case then look at the path which is this, look at this a, b path.

Now b is also here and b is also in this bag, then b is also part of every bag and in particular b is part of the intersection X t 1 intersection X t 2. So, this immediately implies there is a bag containing both a and b and from here we can infer the desired what you call separation property. Now notice this is how we actually we are trying to construct our treedecomposition when we were discussing which we will see later when we were constructing.

But it does satisfy the property we wanted that if I look at the set of vertices which is part of like look at edge and whatever I have associated if I delete this then it does disconnect the graph which is stored below me from the graph which is stored outside. So, in some sense when I look at this t 1 and t 2 then it basically tells us what is the kind of graph?

(Refer Slide Time: 27:51)

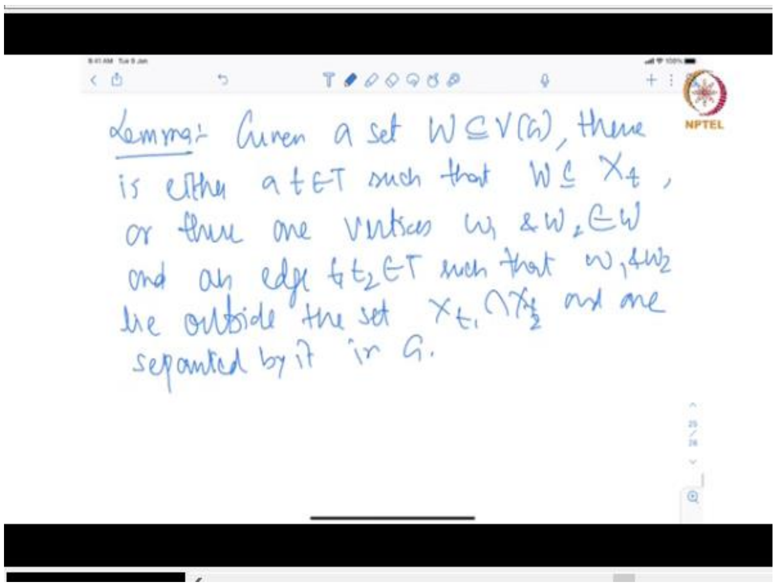


both a & b.

& from here we can infer the desired separation property!

So, this is how the world, if I look at the graph case then this is how it looks like. So, this X t 1 intersection X t 2 separates some vertices from here. So, so this was a graph below this t 1 then this is a set of vertices which is kind of guarding or just separating from the below graph from somewhere outside. So, this is the kind of separation property we were anyway looking when we were trying to construct those decomposition and indeed this decomposition has that property. So, let me prove another property to you which is nice.

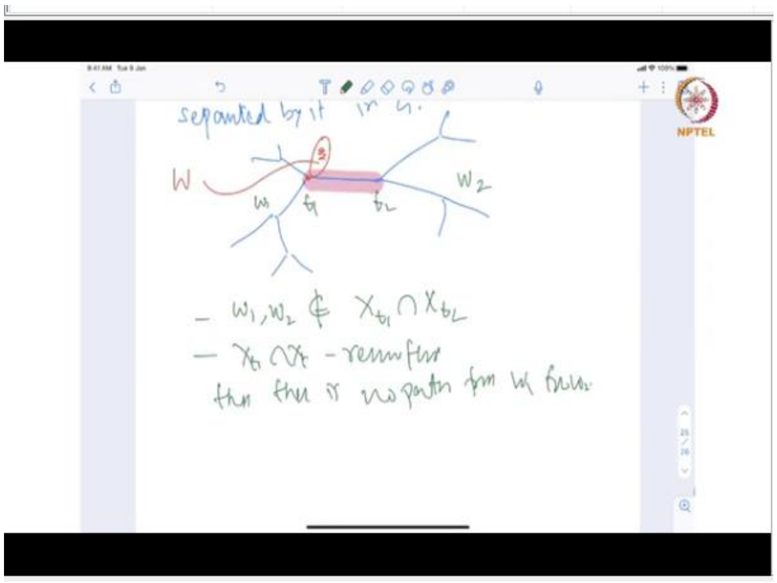
(Refer Slide Time: 28:40)



Lemma: Given a set $W \subseteq V(G)$, there is either a $t \in T$ such that $W \subseteq X_t$, or there are vertices w_1 & $w_2 \in W$ and an edge $t_1t_2 \in T$ such that w_1 & w_2 lie outside the set $X_{t_1} \cap X_{t_2}$ and are separated by it in G.

Given a set W, subset of V G, there is either a t in T such that W is subset of X t or there are vertices w 1 and w 2, in w and an edge t 1, t 2 in T, t2 in T such that w 1 and w 2 lie outside the set X t 1 intersection X t 2 and are separated by it in G. So, what is this tells us? So, let us try to understand what is this tells us. It tells us.

(Refer Slide Time: 30:03)



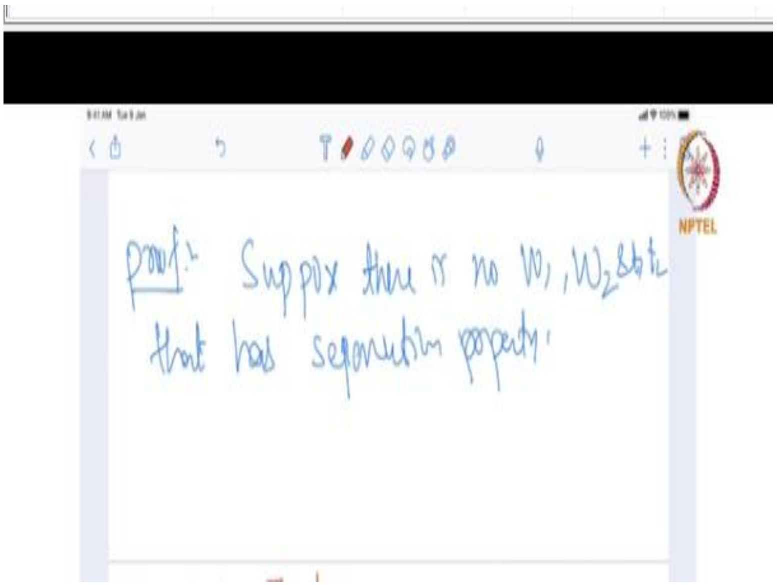
separated by it in G.

- $w_1, w_2 \notin X_{t_1} \cap X_{t_2}$
- $X_{t_1} \cap X_t$ - remove this then there is no path from w_1 to w_2.

So, suppose this is my tree and now you gave me a set w. So, either I will be able to find you one node at a meeting where w is completely contained inside this or I will find you 2 vertex w 1, w 2 and an edge and what is the property that first of all w, w 2 is not an element of X t 1 intersection X t 2, that is a first thing and but we know that if they do not belong to and if you remove this then there is no path from w 1 to w 2.

So, this is just so what does it mean? So, basically what this lemma tells us that if you give me a set w either I can find a way to separate it or this w belongs to the one bag**.** So, now let us prove this. This is the proof is not very hard.

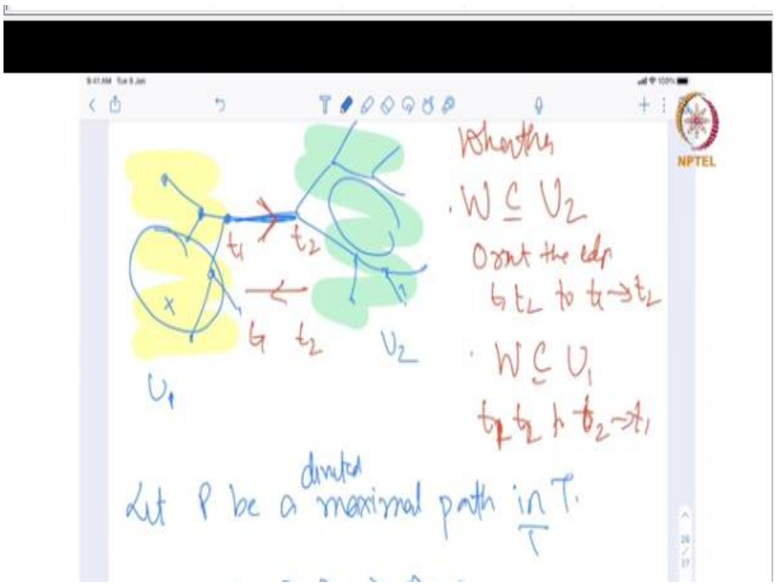
(Refer Slide Time: 31:38)



proof:- Suppose there is no W_1, W_2 & $t_1 t_2$ that has separation property.

So, proof; suppose there is no w1, w2 and t 1 and t 2 that has separation property. So, suppose we cannot separate w by an edge like every edge of the property that like. So, now given this now we need to so suppose this does not happen, that you are not able to separate, like you delete an edge and w 1, w 2 is not there then we need to show that there is a bag that contains whole of w that is a. So, this is a trick which I would like to teach you this is called orientation trick.

(Refer Slide Time: 32:35)



Whether

- $W \subseteq V_2$
  Orient the edge $t_1 t_2$ to $t_1 \to t_2$
- $W \subseteq V_1$
  $t_1 t_2$ to $t_2 \to t_1$

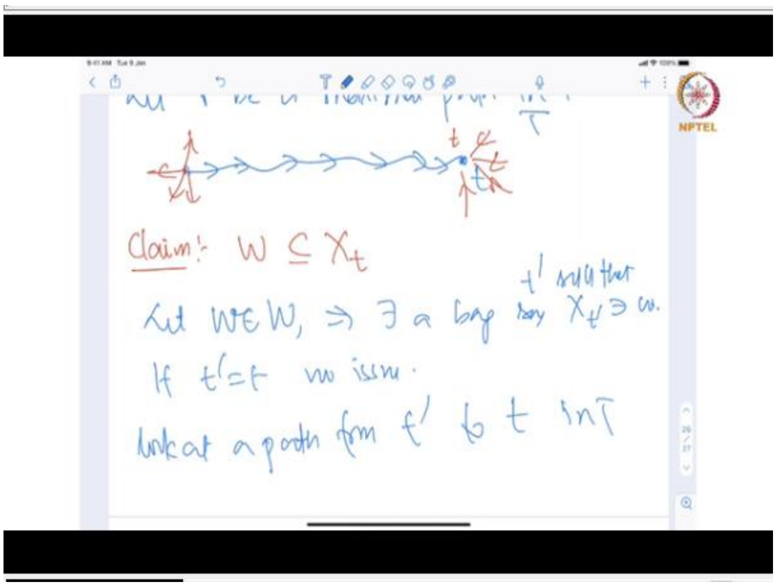
Let P be a $\overset{\text{directed}}{\text{maximal}}$ path $\underset{\overleftarrow{\ \ }}{\text{in}}$ T.

Now we have this tree, so I just pick up an edge and suppose this is t 1 and t 2 and now I ask myself whether w is a subset of say U 2, remember what was U 2, U 2 are the union of all the vertices here and U 0 union of all the vertices here. So, this is my U 1 and this is my U 2. So, I ask myself look whether w is completely contained inside U 2. If that is the case because look it is not possible that some vertex of w is here and some vertex of w is there and that guy does not occur in t 1, t 2; this cannot happen, because otherwise then we would have found the separation and we are done.

So, what is the property? Then the property that look at any edge w is either entirely contained inside here or w will be entirely contained inside this. But it is not possible that there is some vertex of w here and some vertex of w here and these 2 guys do not occur here, this cannot happen. So, whether w is subset of U 2, if w is belong to a subset of U 2 you take this edge and you orient it towards t 2, orient the edge t 1, t 2 to t 1 directed t 2. If w is subset of U 1 then what we have done? We have just oriented then t 2, t 1 like t 1, t 2 to t 2, t 1, this is how you did.

So, basically what I am saying I am looking at an edge I say which side contains w completely, whichever side it contains complete, I point my arrow towards that side of my tree. So, this is what we do the whole orientation. Now and this is fine look and for every arc you will be able to find such an orientation. So, this is perfectly fine. Now what will be do? Let P be a maximal path in T like P be a directed maximal path in T, so suppose this is a maximum directed path, this is in T and suppose this is my last vertex.

(Refer Slide Time: 36:16)



Claim:- $W \subseteq X_t$

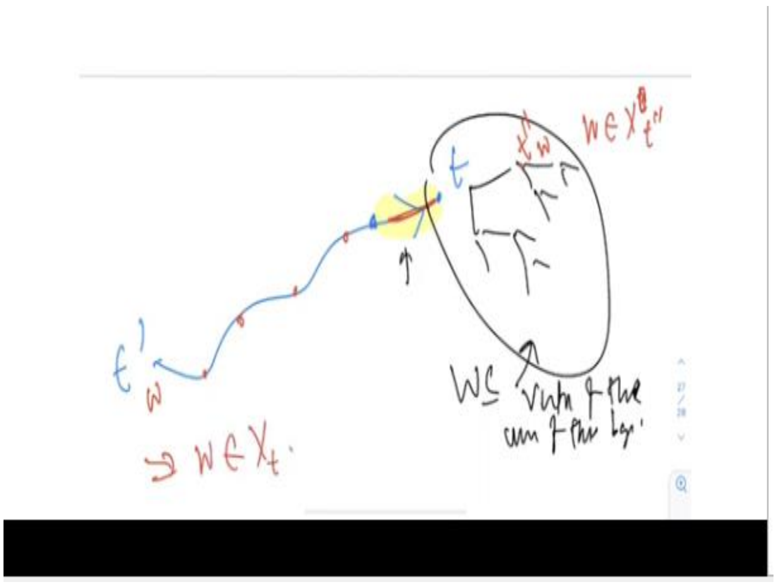
Let $w \in W$, $\Rightarrow \exists$ a bag $\overset{t' \text{ such that}}{\text{say}}$ $X_{t'} \ni w$.

If $t' = t$ no issue.

look at a path from t' to t in T

Now if this is a directed maximal path what is the meaning of this? All the arc which is going are like this, all the arc which is going are outwards from here and look at this T. I claim w is subset of (()) (36:49) let us see why? Let w be element of w. This implies there exists a bag say some t prime such that X t prime contains fine. So, if t prime = t no issue because w is also there. Otherwise look at a path from t prime to t in t not in the oriented. So, now let us look at this path.

(Refer Slide Time: 37:47)

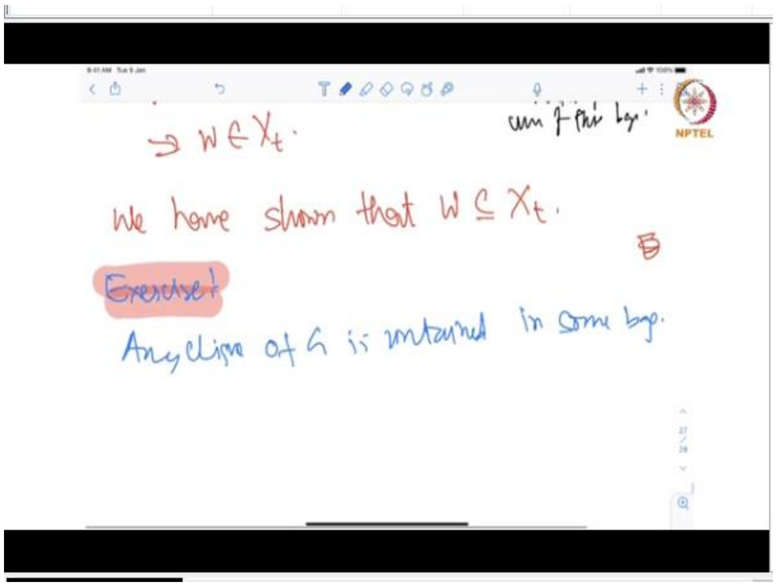


So, there is a path from t prime to t, now look at the last edge look at this ledge, what is an orientation of this edge? So, why the orientation of this edge is towards t because t does not have any and what you call, t is like a sink, t is not a source, because so t like what is the property of t? T is not a source, like it does not have any outgoing edge, it only has incoming edge. So, what is this orientation implies?

Orientation implies that w is contained inside if you delete this edge and look at the world here, there must be so basically what I am saying is that look at delete this arc and look at the objects here, so this is like some let me draw better. So, suppose this is a tree after I delete this as. Now what is the meaning that of this edge it means w is contained inside the vertices of the union of these bags, but then what does imply?

This implies that there will be some t double prime here where w is contained inside X t double prime. Then look at this w is here which implies what, that w belongs to every vertex on every bag on this path and in particular this implies w belongs to X t.

(Refer Slide Time: 40:36)


$\Rightarrow w \notin X_t$.

um of this bag.

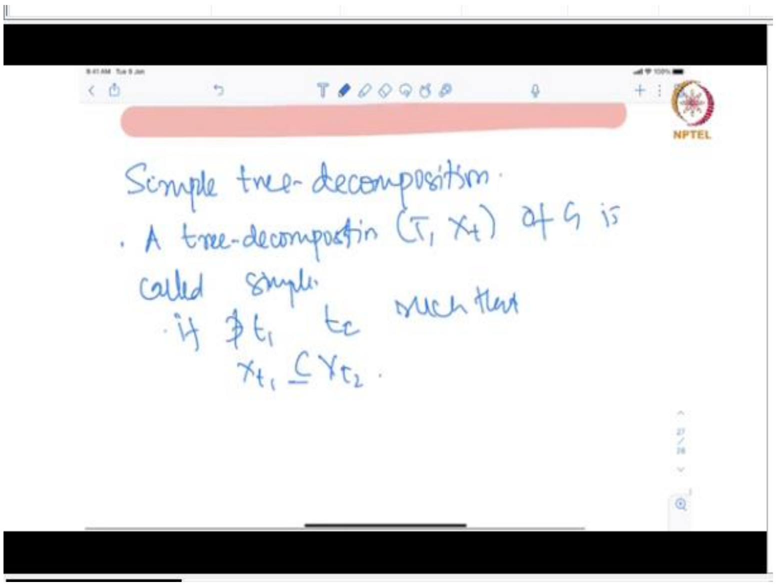
We have shown that $W \subseteq X_t$.

Exercise!

Any clique of G is contained in some bag.

So, we have shown that w is a subset of X t. So, this concludes the lemma and now here is my exercise, so my exercise is show that any click of G is contained in some bag. So, now let me prove one last property about treedecomposition. So, these 2 properties which I talked about is kind of separation. So, if you give him a any big set treedecomposition has a separator which separates it. This is what it is trying to tell.

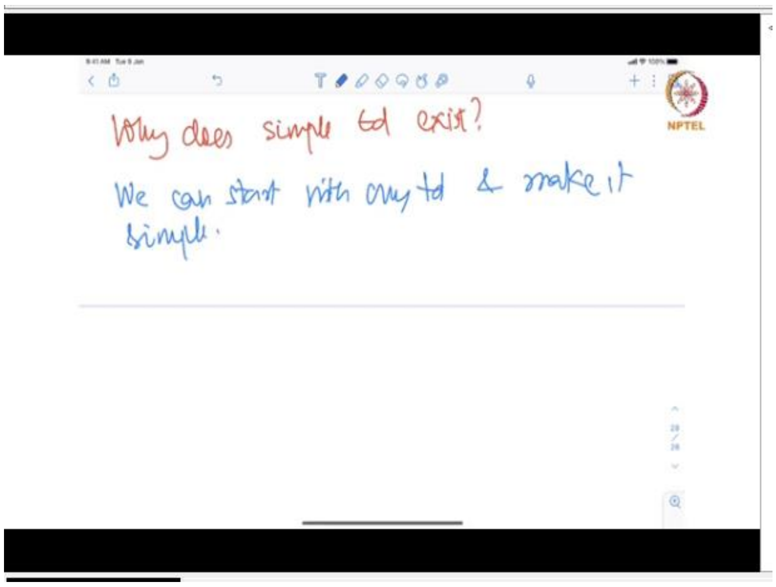
(Refer Slide Time: 41:52)


Simple tree-decomposition.

- A tree-decomposition (T, X_t) of G is called simple
  - if $\nexists\, t_1\ t_2$ such that $X_{t_1} \subseteq X_{t_2}$.

So, now let me tell you last property the notion of simple treedecomposition. So, what is the notion of simple treedecomposition? So, a treedecomposition T, X t of G is called simple is the following, if there does not exist some t 1, t 2 such that X t 1 is a subset of X t 2. Meaning what is it tells us, a simple treedecomposition is the one where no bag is contained inside the other pack. This is the only property of simple treedecomposition. So, now notice that why there are several questions?

(Refer Slide Time: 43:01)

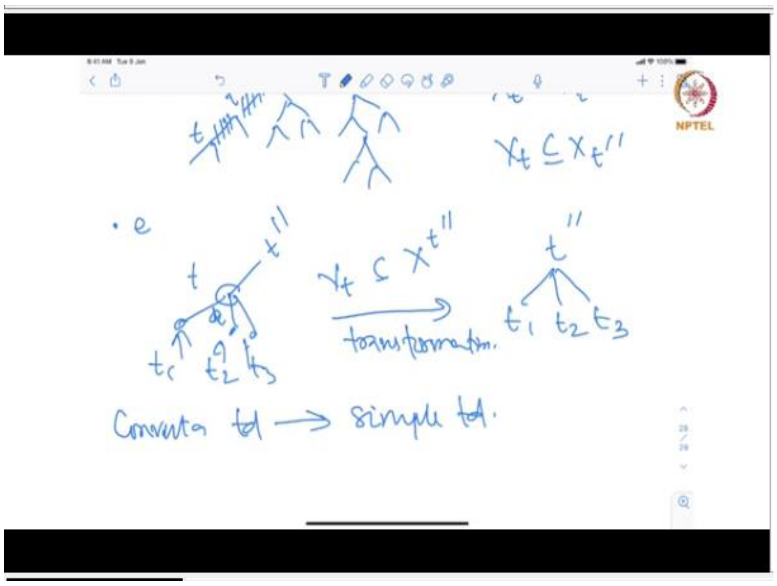


Why does simple td exist?

We can start with any td & make it simple.

The question number is why does say simple treedecomposition exist? That this is, so what you can do this is very simple we can start with any treedecomposition and make it simple, how look**.**

(Refer Slide Time: 43:29)



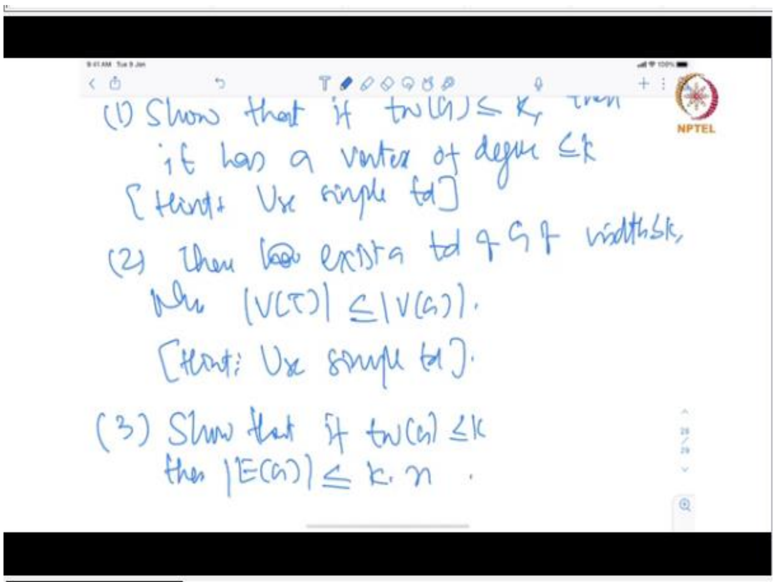
Now notice that suppose there is some say here is like some t and some t prime where X t say suppose X t prime then actually there exists an edge here, t double prime such that X t is contained inside X t double prime why? Because of the connectivity property in fact X t will be part of every bag on this path. So, if this is non simple path then there exists an edge which is non-simple and how can we get rid of non-simple edges is very simple.

So, what you do is that look at this t and t prime. So, in this treedecomposition suppose I found a here I have a t and I have a t double prime and suppose X t is contained inside X double prime then suppose here it is and suppose this has some children then what you do the transformation is that you take t double prime and all these guys you make them directly adjacent to t double prime and you forget like you throw away t.

So, you make some changes, so suppose this is t and it has children say t 1, t 2, t 3 then you forget t and you make t 1, t 2, t 3 directly t double prime and that is it and you can still show that this forms this is a very valid treedecomposition because no property will be violated because a subset is contained inside this so edge property will hold, vertex property will hold and the connectivity property will also hold.

So, you can go bottom up, you start with a non-simple treedecomposition and you bottom up you apply this transformation and convert a treedecomposition to a simple treedecomposition. So, this is a another simple fact that what is important to know. Now it has some very beautiful properties so this is some exercise which I want you to think about it and if you are not able to do we will do.

(Refer Slide Time: 46:01)


(1) Show that if $tw(G) \leq k$, then
it has a vertex of degree $\leq k$
[Hint: Use simple td]
(2) There exists a td of G of width $\leq k$,
where $|V(T)| \leq |V(G)|$.
[Hint: Use simple td].
(3) Show that if $tw(G) \leq k$
then $|E(G)| \leq k \cdot n$.

So, because of the simple treedecomposition, first show that if treewidth of G is at most k then it has a vertex of degree at most k + 1, hint use simple decomposition. Second there exists a treedecomposition of G of width at most k where vertex set number of vertices in t is less than number of vertices of less than at most. Again hint use simple treedecomposition.

Three, so that if in fact say k so that if treewidth of G is at most k then number of edges in the graph is at most k times n, use 1.

So, this is the three exercises which I would like you to try out and if you do not succeed I will do it, but with this I will conclude this week lecture about treedecomposition and next week we will see some algorithmic properties of treedecomposition, for example how to compute treedecomposition, how to design dynamic programming algorithm of treedecomposition and as a final touch we will also see how to get sub exponential time algorithms on very special classes of graphs such as planar graphs, apex minor free graphs and some such things using treedecomposition and some simple properties of planar graph, with this let me conclude my today's lecture.