PROFESSOR: So welcome to this week. We're going to talk about trees mainly. We're going to talk about some very special ones, spanning trees. But before we do this, you will go through a whole bunch of definitions and look at examples to explain how it works.

So let's talk about walks and paths.

So you've already seen a bit of graph theory. We've talked about graph colorings and so on.

But now we're going to talk about special types of graphs, and special structures within such graphs.

Well, let's start with the first definition. How do we define a walk? Well, a walk is something very simple. You walk from one vertex in the graph over an edge to a next vertex, to a next vertex, and so on.

So how do we define this? A walk is a sequence of vertices that are connected by edges. So for example, we may have something that looks like a first vertex. That we call the start. And an edge that goes to a second vertex. We continue like this until the end, say, at vk, which we call the end of the walk.

And if you look at this, we say this is a walk from v0 all the way to vk. It has exactly k edges. And we say that the length is actually equal to k.

So this is a very basic notion. And what we are really interested in is actually paths. And paths are special types of walks in which all those vertices are actually different from one another. But let's first give an example of a graph that you will study throughout the whole lecture, especially when we come to spanning trees.

So let's have a look at the following graph.

So let this be a graph. And for example, we can have a walk that goes from this particular edge. Say we call this v0. It goes over this edge over here. Suppose I want to end over here.

Well, I can go many ways in this particular one. I can go for this edge. I may go over here. I may go all the way to this part over here. I may return if I want to. And I finally, I take this edge for example back to-- over this edge, I will end in the last vertex vk.

So what we notice here is that we have a walk that actually gets to this particular vertex, and then returns to this vertex. So for a few other edges, and then goes all the way to vk.

So if you talk about the path, then we really want all the difference vertices not to occur twice. So let's define this so the destination is that a path is actually a walk.

Is a walk where all edges, where all vertices, vi's, are different.

In this particular case-- well for example, if you strip off this part over here, you will get a path from v0 to here, to here, to here. And all these vertices on the path on this walk are different. And therefore it is a path.

This also shows us something. It is possible to construct from a walk a path. As you can see, we started off with this particular walk. And we just deleted, essentially, a part where we sort of came back to the same vertex again. And when we delete all those kinds of parts, you will end up with a path.

So can we formalize this a little bit better? Then we get the next lemma.

I call this lemma one. And let's see whether we can prove this. So we want to show that if there exists a walk from, say, a vertex u to-- well, maybe I should not use arrows. That's confusing-- from u to v. Then there also exists a path from u to v.

So how can we prove this? Do you have any ideas?

So what kind of proof techniques can we use here? And maybe one of the principles that we have seen?

AUDIENCE: [INAUDIBLE] but I have an idea about how you could show this. It's basically if you visit one vertex-- let's you go 1, 2, 3, 4, 5, 3, 6. You can walk from 1 to 6. Then we just take out the 4, you just go 1, 2, 3, 6.

PROFESSOR: Right. So what did we do there? We essentially had a walk. And then we cut out a smaller parts here that recurs in a sense. And we have shortened the path, we have shortened the walk to a smaller path.

So what we've used here is we can sort of take out parts of the walk just like we did over here until the walk gets shorter, and shorter, and shorter. So maybe one of the things that we may consider is a walk of shortest length.

And we know that this is possible. Oops, sorry about that.

So let's prove this. So for example, suppose there exists such a walk from u to v. Then we know by the ordering principle that there exists a walk of minimum length. So that's what we're going to use here.

And then we're going to essentially go in the same direction as what you were talking about because we're going to show that it's not possible to delete anything more. Because otherwise, if you could do that, the walk would get shorter. And that's not possible, because by the well ordering principle, we simply consider that there exists a walk of minimal length that we are interested in.

We will show that this particular walk is actually going to be a path. So let's see how we can do this.

So let's do walk be equal to-- well, it starts in v0. It's equal to u. There's an edge to, say, v1. It goes all the way up to vk, which is the last vertex, v.

So what are we going to prove? We're going to prove that this is actually a path. And since this is a walk of minimal length, well suppose it is not a path. So we are going to use contradiction.

Well if it's not a path, then by our definitions over here, there must be two vertices that are actually the same, just like in here. So we go from v0 to, say, this particular vertex, or this one. And then we come back to this one, and then all the way to vk.

So if it is not a path, then two of those must be equal to one another. And then we are going to use this trick. We're going to take out this part. We get the shorter walk. But that contradicts that our walk is minimal length. So that's sort of the proof technique that we're doing.

OK. Let's do this.

So first, let's consider a few cases. If k equals 0, then that's pretty obvious. We have only a single vertex u that is connected by a 0 length path to itself. So this is fine.

For k equals 1, we have just a single edge. And then we're done too. That's a path as well.

So let's consider the case where k is at least two. Well, suppose it's a walk. It's not a path. So we are going to assume the opposite.

If it's not a path, then we concluded that two of those vi's must be equal to one another. So there exits an i unequal to j such that vertex vi equals vj.

Ah, but now we have a walk that goes from v0 all the way to, say, vi, which is equal to vj. And then we go all the way up to vk which is equal to v.

So essentially, we have created a shorter walk. We have taken out this little triangle over here.

Now this is a shorter walk. And this contradicts our assumption of minimality.

So that means that our assumption over here is not true. So the walk is actually a path. So this concludes the proof.

So even though this lemma is pretty straightforward in the sense that we all feel that if you have a walk, then you can create a path, if you really want to write it down with all the principles that we have been taught, you can see a few things appearing here. So we first started with the well ordering principle. And also we used the method of contradiction in our proof.

So let's set talk about the next definition. So we talked about walks and paths. Let's talk about connectivity.

We actually defined two vertices, u and v to be connected if there is a path that connects u to v. So u and v are connected if there is a path from u to v.

And now we can talk about the graphs that are connected. That's a very important concept. A graph is connected if every pair of vertices is connected. So when every pair of vertices in the graph are connected.

So an example is the particular graph on the blackboard. You can see here that there's always a path to be found from this vertex, to this one, or to that one, to this. to that, to this, or that. You can do this for any two vertices. So this is a connected graph.

A graph that is not connected looks something like this for example. You may have a single edge plus, say, another triangle, maybe with another edge or something like that. So they are disconnected from this vertex. I do not have a graph that connects to this vertex over here.

So this is about connectivity. And now we can start talking about cycles and closed walks.

Together, these concepts will help us to define trees. And then we can start talking about this very special concept, spanning trees.

So let's see how that works.

So cycles and closed walks.

We first define a closed walk. And once we have done this, we are able to get into cycles.

So what's a closed walk? Well, it's actually a walk for which the start and the end vertex are the same.

So it starts and ends at exactly the same vertex.

So this is all pretty straightforward. So we have v0 connected to, say, v1, all the way up to vk. And vk itself is equal to v0. So you have a walk that goes all the way from v0 all the way back to vk.

For example, in this graph, you could have walked all the way back to v0. You would have a closed walk.

So what's a cycle? A cycle is a little bit more special in which all these vertices over here are actually different from one another. So if k is at least three-- so the walk is not just a simple edge, or just a single vertex. And if all v0, v1, all the way up to vk minus 1 are different, then we are saying that we have a cycle. Then it is called a cycle.

So these two concepts, connectivity and cycles, will help us to define trees.

So what are trees? A simple example is something that looks like this, for example.

The idea is that a tree is both connected. And it has no cycles. So in this case, we can see that every vertex it's connected to every other vertex. But there is no cycle like a triangle or something that is connected all around.

So let's define this properly. A definition is that a connected and acyclic graph is called a tree.

And within trees, we also have something very special that we call leaves. And leaves are exactly those nodes that have degree one. So for example, this node has degree one. It has only this edge connected to it. So this is called a leaf.

This one also is a leaf because it has just one edge connected to it. But this lead over here has three edges. So it has degree three. This one has degree four. This is the leaf again. This one is a leaf, this one, and this one.

So this is very important concept. We often use leaves in our proofs. And we will get to one.

So a lead is a node with degree one in a tree.

OK great.

So now let's have a look at trees. Suppose I'm looking at a subgraph of a tree. What can we say about a subgraph? What kind of structure does this have?

So suppose for example I want to take some connected subgraphs. So that's what I'm actually interested in. So for example, I take this-- suppose I take this connected subgraph.

Then what does it look like? Do we see some structure in this? It's-- no?

AUDIENCE: It's a tree.

PROFESSOR: Yeah, it's a three. So are we able to prove this also?

So this lemma you actually lose later on. So any connected subgraph of the tree is a tree.

And to prove-- well, how can we start out a proof? Any suggestions?

So let's have a look here. So how may I think about this? Well, I may take a connected subgraph, and I want to show-- of a tree- and I want show that it is a tree.

A general method could be, for example, to--

AUDIENCE: [INAUDIBLE] You know, one of the things, one of the conditions is it has to be acyclical. So since you're not adding nodes and you're not starting within cycles, you can't create a new cycle.

PROFESSOR: Yeah. I think you're saying something like, if I have a connected subgraph, which is like a smaller part of the tree-- now suppose that would not be a tree. Say it would have a cycle. Then that cycle would also be present in the tree itself. And that's not really possible because a tree has no cycles. So that's sort of indeed how the proof goes. So we essentially use contradiction.

So let's suppose that this connected subgraph is actually not a tree.

So suppose it's a connected subgraph. It's not a tree. Well then, in that case, it must have a cycle, because a tree is something that is both connected and has a cycle and is acyclic. We have a connected subgraph, which is not a tree. So it must have a cycle.

So it has a cycle. But now since it is a subgraph of the bigger graph, we know that the cycle must also be in the whole graph. So the whole graph has this particular cycle.

Wait a minute. The whole graph is the tree. And a tree's acyclic. So we get a contradiction.

So that means that our original assumption that it's not a tree is wrong. So it is a tree. So this is, again, such a general kind of proof method that we repeatedly use here.

So this lemma, which we will call two, we will use in the next proof where we're going to talk about the relationship between vertices and edges within trees.

It's a very beautiful relationship. The lemma states if you have a tree that has n vertices, then it must have n minus 1 edges. So this is a very tight relationship here.

So a tree with n vertices has n minus 1 edges.

So let's see. How can we prove this one? Any suggestions?

So what did we use at the beginning of the course here? So we have n vertices. And we want to show that it has n minus 1 edges.

We can use induction, right? So let's use induction. on n.

So how do we proceed if you do this? Well, we always start out with an induction hypothesis. So the statement here would be something like, well, there are n minus 1 edges in any n vertex tree.

And if you start with induction, you always start with the base case, right? So we have the base case.

So how does this work? So for example, we want to consider P1. So there are zero edges in any one vertex tree. Well, that's true, right? If I have just one vertex, there's no edge. So this is definitely correct.

So that's easy. So let's prove the other part, which is the inductive step. And the inductive step is--

So to do the inductive step, we are going to always assume Pn. And then we want to prove Pn plus 1.

So let's do this. So we have the inductive step.

And we always start out the same way. So we suppose P of n. And now we want to prove Pn plus 1. So how do we do this?

Well, take a tree that has n plus 1 vertices vertices. And we want to show that it has n edges. So take any such tree. So let T be a tree that has n plus 1 vertices.

So how can we use the induction hypothesis here? So do you have any ideas over here? So I have a tree with n plus 1 vertices. And if I want to be able to use this induction at this induction hypothesis, then I can only apply it on trees that have n vertices. So somehow I have to delete one vertex. Right?

So what kind of vertex can I delete in such a way that we still have a tree?

And I need that because this induction step can only be applied to trees.

So what type of vertex can I remove from a tree and it still stays a tree? So for example, if you look at this example up here, what kind of vertex can I remove here?

AUDIENCE: A leaf.

PROFESSOR: Yeah, except. I can remove a leaf-- for example, this one. And why is that? Because if there's only one edge connected to the rest of the tree-- so if I delete it, I will actually keep a tree. So that's how the proof will continue.

So we take out one vertex. So let v be a leaf of the tree.

And let's remove this particular vertex.

So what do we get? Well, we again have a connected subgraph. So this creates a connected subgraph. And we also know that this is a tree. So how can we conclude that?

Well, we had another lemma somewhere that's behind this blackboard. It says that if you have connected subgraph, then we still have a tree.

So this is great. So this is also a tree.

So now we can apply Pn because we have n vertices, because we deleted one. So by Pn, we can conclude that this particular subgraph has n minus 1 edges. You simply apply our statement over here.

But we want to say something about the original tree. So how do we do this? Well, we have to somehow reconstruct this tree. Well, we have deleted v. So we have to reattach v again. So let's do this.

So we reattach v. And if I do this, well, let's look at this example over here. I reattach the leaf. I will add one more edge.

So if I reattach this particular leaf, I will get back the original T over here.

And how many edges does it have? Well, it has these n minus 1 edges plus the one edge that is needed to reattach the leaf. So that's all together exactly n edges.

So now we have shown that for every n plus 1 vertex tree, every such tree has n edges.

So now we're done, because this shows P n plus 1. So this is how the lemma is proved. And in the homework, you actually use this.

So let's now talk about very special trees, spanning trees.

I think this is a very exciting topic. So let's look at this particular graph over here.

Let's define spanning trees. So what's a spanning tree? A spanning tree is a subgraph of a graph that somehow spans all the vertices within this graph. So it's a tree that touches every single vertex.

So for example, we may have--

Maybe this is not such a bright color. Maybe this one.

So for example, we may have a tree that looks like this.

So in this particular graph, we can create a subgraph with the thickened green edges over here, which is a tree, and also covers all the different vertices of the original graph.

This is actually pretty amazing that you can do this, you can actually create such a kind of tree.

And what you want to prove, first of all, is that for every connected graph, we can construct such a spanning tree.

So let's first define this. And then we will prove this theorem.

And then after this, we're going to talk about weighted graphs. That will lead to minimum weight spanning trees. And then it gets really interesting because you want to figure out how we can actually construct those for any graph.

So let's define a spanning tree. A spanning tree--

And we appreciate this as SD-- of a connected graph is actually a subgraph that is a tree. So that's the first condition. And it also covers all the other vertices.

So it has the same vertices as the graph.

And over here, we have this example.

So now what you want to show is the following theorem that any connected graph actually has such a spanning tree. So the theorem is every connected graph has a spanning tree. And let's think about a proof together.

So for example, we may start with the idea to use a contradiction. Suppose there exists a connected graph that has no spanning tree. So how would we go ahead?

So how can we prove that we have a contradiction here? So the proof that we're going to propose here is by contradiction.

That means that we're going to assume the opposite. So assume that we have a connected graph, a connected graph G.

That has no spanning tree. So how can we use this? How could this be strange? Or what can we do here?

So one of the nice things about trees over here is-- so that maybe gives us some insight is that if we have a tree-- well, it seems like such a kind of spanning tree is like--

It's a solution to a subgraph that is connected with a minimum number of edges.

Like if I have any other type of subgraph-- for example, if it has cycles, I can always remove an edge of course. So we know a subgraph of G that touches all the different edges and has a cycle can get shortened by eliminating one edge of a cycle that's within such as a subgraph.

So maybe we have an idea here. And maybe we can use this. So maybe we can say, let's set consider a graph T. So let T be a connected subgraph of G, but with a property that it has a minimum number of edges.

So let this be a connected subgraph of G. And we assume that it has the same vertices of G of course.

In addition, it has the smallest number of edges possible.

So what do we know? Well, let's look at our assumption here. We said that a connected graph, G-- that's what we consider here-- for which there is no spanning tree.

So what do we know? We know that this particular T over here can definitely not be a spanning tree, because that's what we assume here.

Can we use this? So let's think about this.

So if it is not a spanning tree, then what must it have? So we already know that T is a connected subgraph of G with the same vertices of G. So the only difference between a spanning tree and T is that the spanning tree is a tree, and T is not a tree. So therefor, T must have a cycle.

So we use our assumption over here. We know that T is not a spanning tree. And now we can conclude that it must have a cycle.

So now we can start thinking about this cycle. And let's feel--

What can we do here? So we constructed T as the subgraph that has the minimum number of edges possible. So now if you have a cycle, the whole idea is that we're going to delete one of the edges of the cycle.

And if you do that, we will be able to construct a smaller subgraph T, essentially, with a smaller number of edges. And that contradicts this particular statement over here. So that's sort of how we go ahead.

So if it has a cycle, let's write it out. For example, here we have a cycle like this.

Well, let's prove that if we remove one of the cycle, that we still have a connected subgraph. So the whole graph T is much bigger, right? It has lots of connections and so on.

Suppose we will remove this particular edge over here. So we remove this one. Do we still have a connected subgraph of G that covers all the vertices of G?

Well, of course it cover all the vertices of G because we only removed an edge over here. So that's fine. All these parts are still connected to the rest.

If you can show that it is connected, then well we just-- well, we removed one edge. So we actually created this smaller graph. And then we get contradiction. And then we're done with our proof.

So let's see whether we can prove that if we remove this particular edge over here, that we still have a connected subgraph. So how can we do this?

Well, let's take-- so the first case would be if a vertex x is connected to y. So I want to take any pair x, y of vertices. And suppose that the path that connects x and y does not contain this particular edge e that I remove here. So suppose this does not contain e.

So I look at the graph T. I Know T is connected. So I take a pair of vertices x and y. I know there is a path between x and y in T. If this path does not contain the edge e, then I know that the same path still exists in the same graph where I've removed this particular edge.

So that's great, right? I have shown that x and y are still connected in the new graph where I've removed this e.

So now let's look at another case. For example, x is over here. And it is connected all the way through here over this particular edge all the way to, say, y over here. So this is case number two.

Well, are x ad y still connected? Yes they are, right? Because I have removed e from a cycle. So I can still go into the other direction of the cycle and connect back towards y.

So what we see here is that x can go all the way to here. But it can still go over the cycle back to here, and then to y. So we are still connected.

So this shows that in both of the cases, for any pair of vertices, even after removal of e, the graph T minus this edge e is still connected.

So let's write this out. So all vertices in G are still connected after removing e from T.

So now we are done. So what was that again? We now constructed a smaller graph-- smaller in the sense that it has one less edge-- that is still connected, and with the same vertices as G. But I assumed that T was already the smallest such graph, such subgraph. So we have a contradiction.

So what does this mean? This means that our original assumption that we have over here cannot be true because we started off with this assumption. Because of this, we could start to construct this. And then you could derive this whole argumentation. And in the end, we get a contradiction.

So our original assumption is wrong. So actually, this is not possible. There does not exist a connected graph G that has no ST. So the theorem is true.

So this is an important theorem. So now we can start talking about weighted minimum spanning trees.

So let's use this picture over here and assign some weights to this graph.

So for example, we have 1, 2, 2, 3, 3, 3, a 1 over here, another 1, 1, 4, and 7. And let's construct a spanning tree.

So let me give an example over here. Actually, this is an example that we have right her in the thick lines in green.

So what is the weight of the spanning tree that we have here? So we simply add the weights of all the edges. So we have 7, 3. That makes 10. 11, 14, another 2, 16, 17, 18, 19. So the weight of this particular spanning tree is 19.

So now the problem is-- and that's the rest of the lecture-- how can you construct a minimum weight spanning tree, one that has the minimum weight possible?

So we see another example. So could I swap some edges around or something like that such that I can get a lesser weight?

So for example, over here, I have-- for example, this node over here is connected by an edge that has weight 3. But I could also have connected it to this particular one over here, which is only weight 1.

If I do this, I actually improve the spanning tree construction. So I will get something that looks like--

Let's see-- that looks like this.

And now we replaced 3 by 1. So it gets 17.

So can we do anything less than this? That's maybe-- it's still a tree, right? Because there's no cycle. Here we have this, this, this, this part over here.

If you want to have connected subgraph, I always need the 7 in here.

Well, it seems like you cannot really replace anything by something cheaper. Like any edge in here seems to be pretty necessary.

Of course, what I could do is I could create other minimum spanning trees. So actually it turns out that 17 is the minimum weight. For example, I could, instead of this edge, I could use this edge. That's another solution. Or instead of this edge, I could use this edge.

So how do we construct such a minimum weight spanning tree? Can we somehow find an algorithm that creates this? And that's the big problem of today.

So let me write out one spanning tree that I will use for the rest of this class. And maybe I will use a little bit different color.

Let's use red over here.

I hope this is visible.

So we have red, this one, and this one, and this one, and this one, and this one. That's going to be-- this is going to be the spanning tree that I'm going to look at. This is also weight 17.

So let's first define what a minimum weight spanning tree is, just to be very precise. And then let's have an algorithm. Maybe you have already ideas of how this could be done.

It turns out actually that if you just start with a greedy approach, you just start to add sort of through the edges of minimum weight, that you can still add without creating any cycles. You will get an algorithm that is going to do the trick for you. So that's kind of surprising.

So let's talk about the definition.

So the minimum spanning tree of an edge weighted graph is defined as--

It's defined as the spanning tree of G such that it has the smallest possible sum of edge weights.

Well the algorithm that I'm thinking about here is very straightforward. Actually I'll use this blackboard because this algorithm will be the focus of the rest of the lecture.

So what's the algorithm?

We actually grow a subgraph step by step. So one edge at a time such that at each step we want to add the minimum weight edge that keeps the subgraph acyclic.

So this is our algorithm. Let's have a look how this works in this particular graph over here.

So suppose I would start off with-- well, if I start with this algorithm. I take, say, any minimum weight edge. Say I take this particular one. So this could be my first edge that I choose.

Then I may want to choose another minimum weight edge such that I do not create a cycle. Well, which one could I choose? I could choose this one. I could also choose this one. With this one, I could choose this part. With this one, I could choose that part. I could also choose this one.

So let's choose this one actually. This would be our second step and our second edge that we select in our algorithm. There are two disjoint edges. So there's definitely no graph.

Well, maybe now our third step, I can choose one of these edges, which also has just weight 1, which is minimum weight. So this could be my third possibility.

Then what's the fourth possibility? Well, I have to choose one of those actually. I cannot choose this weight 1 edge? Why not? Because that would create a cycle.

So I'm not allowed to choose this one anymore. So I choose one of these two.

So for example, I choose this one over here. This is the fourth edge that I add to the whole picture.

So note that in the meantime. I have two disconnected components. One other here, which is this arc, and another arc over here.

So which edge do I attach now in the algorithm? Well, I can still add this minimum weight edge over here of weight two, because it does not create any cycle. So this would be the fifth wonder that I add to the whole picture.

Well, now I can choose either one of those. So for example, this would be my sixth step. And then finally, I can choose none of these because those would create cycles. But I can choose this one. So this is going to be the seventh one that I add to the whole graph that I construct here and grow in this algorithm.

It turns out that this algorithm actually creates for you a minimum spanning tree. And the proof is much more complex than what you have seen before. So that's what we're going to do right now.

So when we think about this algorithm, then somehow I want to guarantee that at every single step, I will be able to grow into a minimum spanning tree. And that's the basic intuition that we have. And that's the lemma that we want to prove first.

So formalizing, we get the following. We want to prove the following statement.

So the lemma that will help us is let S be the set of edges that I have been selecting up to now in the algorithm. So let S consist, for example, of the first m edges.

What we want to show is that we can still extend this set of edges into a minimum spanning tree. We can still grow within the algorithm into a minimum spanning tree. That is what you want to show.

So we want to show that their exists a minimum spanning tree T that has the vertex set V and an edge set E.

So this is the minimum spanning tree for the graph G such that S is actually a subset of the edges in this minimum spanning tree. So this is a nice mathematical formulation that really precisely states that we can still keep on growing into a minimum spanning tree.

So how would this lemma help us in proving the theorem that we want to show? Actually, where is the theorem? I did not write down the theorem so far. I can put this over here.

So the theorem that's want to show is that for any connected weighted graph G, the algorithm creates a minimum spanning tree.

The algorithm produces a minimum spanning tree.

So how can we prove his theorem? Suppose we know that this lemma is true, which we will prove later. Then how do we know that this theorem holds?

Actually, I'm wiping out this particular theorem over here. We will use this in a moment.

So we know that every connected graph has a spanning tree. We already know that. But now we want to show that we can even construct a minimum spanning tree for a weighted graph.

So how are we going ahead? The proof of the theorem is as follows. I will suppose that the number of edges-- the number of vertices-- is actually equal to n. And if we want to show that we get to a minimum spanning tree, we want to first of all show that the algorithm is able to choose at least n minus 1 edges, right? Because a tree has n minus 1 edges. We have shown this before.

The algorithm should not get stuck. So let's first prove that.

So suppose I have chosen less than n minus 1 edges. So suppose we have less than n minus 1 edges picked. Well then what do we know?

Well, there exists as edge in E minus S-- well, let me first write it out-- that can be added without creating a cycle.

So why is this? Well, I know that E is the edge set of my minimum spanning tree. I know that S is a part of-- is a subset of all the edges. I know that a minimum spanning tree is a tree. It's a tree on n vertices.

We know that's a tree on n vertices has n minus 1 edges. I know that less than n minus 1 edges have been chosen so far. So that means that S has less than n minus 1 edges. E has successfully n minus 1 edges. So there's at least one edge that is contained, that it is an element in E, but not in S.

And for that particular edge-- well, since it's part of this tree, together with S, which is also part of all the edges within the tree-- well, we know that that edge, for that reason, will not create a cycle. Because otherwise, the original would have a cycle.

So we know that the algorithm can always select an edge such that's no cycle is created. And that means that we can keep on going in the algorithm until at least n minus 1 edges are selected.

So what happens if we have n minus 1 edges?

So once n minus 1 edges have been chosen, well, then S has exactly n minus 1 edges. E has exactly n minus 1 edges, because it's a tree. They are subsets of one another. So they must be equal to one another.

So essentially what we have here is that S exactly defines the minimum spanning tree that we are looking for. So the algorithm at the very end, when it has chosen n minus 1 edges, produces a minimum spanning tree. So we know that S defines the edges of a minimum weight spanning tree. So if this lemma really holds true, then this theorem can be shown.

So now let's have a look at this lemma. And we're going to use that picture over there to see how we can prove this. So this is a little bit tricky. And the way to go about this is actually to use induction. So let's check see whether we can do this together.

So the proof of the lemma is as follows.

We want to use induction. On what do you think we are going to induct here? Well, we want to prove this particular lemma, and we have this particular variable m. So that seems to be a good methods to continue. So we'll use induction on m.

Let's state the induction hypothesis. Well, it's exactly the same as in the lemma. So we say for all G and for all sets S that consists of the first m selected edges, there exists a minimum spanning tree of G such that S is a subset of E.

So this is how induction hypothesis. And as always, we start with the base case. And the base case is going to be pretty straightforward in this particular case.

We have to do a little bit of work for it. So let's see how that works.

So the base case is m equals 0. So let's reread this statement where m is equal to 0. Well, if we have zero selected edges, then S is the empty set.

So I have to prove that for all G, there exists a minimum spanning tree such that the empty set is a subset of E. Well, the empty set is always a subset of E, right? So that's no problem.

So what I need to proof is that for all G, there exists a minimum spanning tree of G. And this is where our previous theorem comes in, because we showed that for all graphs G, there exists a spanning tree. And if there exists a spanning tree, there exists a minimum weight spanning tree.

So that's the base case. So m equals 0 implies that S is equal to the empty set. So that means that S is definitely a subset for any set of edges of a minimum weight spanning tree.

And now we're going to use our theorem to show that there also exists a minimum spanning tree. So this in itself is not yet sufficient, right? You see that. I know that if S is the empty set, then I know that this always holds. But the statement is a little bit bigger. For all G, I still need to prove there exists a minimum spanning tree. And that's what our previous theorem told us. So we'll now write that part out. So we have to base case.

Let's see.

I think I will go and prove this over here. So let's look at the--

That's the inductive step. And then hopefully we can play a little bit with this picture up here to get some insight.

So how do we do this? Well, with an inductive step, we start as usual.

We assume that P of m holds.

Now how do we go ahead? So I want to prove Pm plus 1, which is stated over here. So I want to consider the set S that consists of the first n plus 1 selected edges.

So well, essentially what I'm interested in is what happens in the n plus 1 step. So let E denote the edge that I added in the m plus 1 step.

And let S be the first m selected edges. And we know that we can apply p of m for S, right? Because we assume this is our inductive step.

So we can apply something for S. We would like to show Pm plus 1. So we would like to show something for the union of those two. So let me repeat that.

Let S denote the first m selected edges.

So by our induction hypothesis, we can-- well, let's apply it. We know that there exists a minimum spanning tree such that S is a subset of the edges.

So let's just pick one such minimum spanning tree. So let T star be such a minimum spanning tree. It has edges E star.

And we know that S is actually a subset of E star.

Let's look at the very first case. So what are the kind of cases that we are interested in? So let's think again. What do we need to prove? We need to prove Pn plus 1.

So we need to prove something about the union of S and E. We want to show that there is a minimum spanning tree such that the edge set of this minimum spanning tree contains both S and also E.

So what kind of cases can we handle here? Well, what will be a really good first case? The first case could be that E is actually already part of E star. Well, if that's true, then of course, S together with E is also a subset of E star. And that means that we are done, in this particular case.

And why is this? Because this is an example of a minimum spanning tree that contains both S and E. So it contains the n plus 1 first selected edges. And so this is Pn plus 1.

Now the next case is the difficult part.

OK, let's wipe out-- actually, we do not really need the proof of the theorem anymore. So let's take this off the blackboard.

So the way to do this is to sort of see how we can use the tree, T star, and somehow transform it into another tree for this particular case where we assume that E is not in E star.

So let's have a look at that graph up there. Let me rewrite it a little bit. And let's look at what happens in the algorithm after two steps.

So I will redraw the graph. So we have, say, these edges.

This one-- let me see. I want to use some different colors to make this as clear as possible. We have this one. We have this one. We have this one. And we also have this one actually.

Sorry.

All right, so let's define the different edges. The straight ones from the set S. So after two steps over there, we have selected this edge first, and then this edge was the second edge that we selected in our algorithm. So these two together form the set S.

So what is T star? T star could be, for example, the spanning tree that contains both these edges plus the dotted ones.

So let's have a look here. This is indeed a tree, right? And if we look into this picture over here, you can see that the weight of this particular tree is 17. It's also a minimum weight spanning tree.

What is G? G is the complete graph and contains not only these two types of edges, but also these sort of more orange colored ones.

So this is so the situation that we are in. Now in our algorithm, we want to choose this particular third edge. So this is going to be our e, this particular edge. And as you can see, e is not part of T star. It's not one of those two kinds of edges. It's actually one of the orange edges.

So what can we do here? Well, the idea is that we want to recreate like a you tree in which we somehow exchange e-- exchange one of the edges in T star with e, and still sort of maintaining the minimum spanning tree property. So how can you prove this?

The way to do this is to figure out what the algorithm teaches us about e, and also what the tree property tells us.

So let's combine all our knowledge. So first of all, we know that the algorithm, from its definition, implies that as together with e, actually has no cycle. That's how we select the n plus 1 edge. There's no-- all selected edged together should not contain a cycle. That's the definition of the algorithm.

We know that T star is a tree. And this implies that if you look at the graph that has the edges, has the vertices V together with E star with edge e, then this must contain a cycle.

Now this property I will not prove right now, but it is pretty straightforward. It's a one line proof actually in the book. So you can look it up.

So it simply states that if you have a tree, if you add another edge, an edge that is not contained in E star, then we will create a cycle.

So now we can combine these two statements together and conclude that, well, if S with e has no cycle, but E star with e has a cycle, and S is contained in E star, well there's only one possibility. And that is that this cycle must have an edge, e prime, that is in E star minus S.

Because if it would not have such an edge, then all the edges of the cycle must be located in S-- in S together with E. But S together with E has no cycle. So that's not possible. So this cycle must have and edge e prime that is outside S, but still in E star.

So we can find one over here. e prime can be this particular edge. So e prime is in E star. But it is not already selected in S.

So now we are going to do a trick. The idea is to swap e and e prime. So that's the main idea.

So let's first write down what we know about the weight of e prime and e. So since the algorithm, well, could have selected e or e prime-- it did select e, right? But it could have also selected e prime, because e prime does not create a cycle with S. That's what we have seen here. So this really implies therefore that the weight of e is at most the weight of e prime.

So why is this? Because the algorithm always chooses the minimum weight. And it can choose between e or e prime, but it chose e. So the weight of e must be less than the weight of e prime. So that's what we know also.

So now let's swap e and e prime in T. And then we're almost done with this proof, because that will be the tree of which we will prove that it is a minimum spanning tree.

So the key idea is to swap e and e prime in T. How do we do it? Well, let T double star be equal to V with the edge set E double star where E double star is really equal to, well, the original set E star. But we take out of this e prime. So this one is taken out of the minimum spanning tree. And we add e in here.

Now we want to prove that this particular T double star is a minimum spanning tree that fits our lemma. So what do we know? We know that T double star is acyclic.

And why is this? Well, we actually removed e prime from the only cycle-- so look up here also-- the only cycle in E star together with e. So it's acyclic.

T double star is also connected. And why is that? Well, since e prime is removed-- but it was on a cycle. So it remains connected.

And finally, we also know that T double star actually contains all the vertices in G. So all these three together with prove that T double star is a spanning tree of G, because it's acyclic, connected, contains all the vertices.

Is it a minimum weight spanning tree? Now we are really almost done, because-- let me write it out here. We know that the weight of T double star is at most the weight of T star.

Why is this? Well, we showed just now that the weight of e is at most the weight of e prime. So we have exchanged e and e prime. So that means that the weight of T double star is at most that of the weight of T star.

We also know that T star is a minimum spanning tree. So if you combine these two together, then we know that the weight of T double star cannot get less than the minimum weight possibility. So it also has a minimum weight. So T double star is a minimum spanning tree.

So now we're done because in this case, we have constructed a T double star such that S together with E is a subset of E double star. And that means that we have shown that our induction step is true. We have shown that Pn plus 1 holds.

So now we finally have figured out that the lemma is true. And now we showed already how to use the lemma in our proof for this particular theorem.

So this is the end of this lecture. And tomorrow, you will actually go again over this proof because it's rather complex. And then hopefully you really get into all the different proof techniques.

OK. Thank you.