One of the multiple definitions of trees that we saw is that it's a minimum edge simple graph that connects up a bunch of vertices. And that leads to the idea of finding a spanning tree within a simple graph that maintains the same connections. So let's begin with a precise technical definition.

A spanning subgraph of a graph, G, is simply a subgraph that has all the vertices of G. So, again, a subgraph of a graph means it's got a subset of the vertices, and a subset of the edges. Spanning subgraph is that has all of the vertices, but a subset of the edges. And the definition of a spanning tree is a spanning subgraph that is a tree.

Now, not all graphs are going to have a spanning tree, because the tree has to be connected. If the original graph is not connected, there's no way you can find a spanning tree using only the edges that are there already. But it's going to turn out that if the graph is connected, it's guaranteed to have a spanning tree. Let's look at an example.

Here's a simple graph. And what I want, then, is a spanning tree, a selection of edges, that connect up all the vertices such that we're only using edges in the original graph, and they form a tree. There it is.

So if you check on these magenta edges that I've highlighted, they define a tree. I haven't used three of the edges in the original graph.

Now this particular choice of spanning tree is kind of arbitrary. In general, there's lots of spanning trees. Here's another one, this time with green edges. Again, I'm using only edges from the original graph-- I've left out three different ones, and used a different set of edges to form the tree. But there it is.

It's got no cycles, and it spans the graph because every vertex in the graph is part of it. And of course, it's connected since it's a tree. There's actually some lovely combinatorial theory, which enables you to calculate the number of spanning trees in a simple graph without too much difficulty, just given its adjacency matrix. But we're not going to go into that for now.

First remark is that every connected graph is going to have a spanning tree, and the reason is, you just pick a minimal edge tree-- a minimal edge connected spanning subgraph, rather. So G, itself, if it's connected, is by definition, a spanning graph of itself, because it's got all its own vertices. That means by the well-ordering principle, there's going to be a connected spanning subgraph with a minimum number of edges. And that one, given that it has a minimum number of edges, it's guaranteed to be a spanning tree.

Now the problem gets more interesting when it has a little more structure-- instead of just trying to find a spanning tree that has a minimum number of edges, it's quite typical in applications that the edges have weights, and we want to find a minimum weight spanning tree. So here's an example where we have a simple graph with a bunch

of edges, and a bunch of vertices. And the edges all are assigned, in this case, an integer weight.

Now the motivation for this kind of graph, as you could think of, these weights on the graph as indicating the cost of transporting some quantity commodity from this vertex to that vertex, directly by a road. Or the time it takes to transmit a signal over this channel. Lots of ways that simple graphs are used to model issues of communication among various locations.

And it's quite typical that the channels and connections between them have different costs. And it's a natural question to say, OK, what's the minimum cost overall tree structure that will enable me to have everything connected to everything else in the same way, but that I can tolerate some of my edges going down? And I still would like to have the cheapest kind of tree that spans them all.

Well, there's a fairly simple way to construct such a minimum weight spanning tree, and that's what we're going to talk about now. How do you find it? Well, the idea is to build it using grey edges. So what that means is that starting off with the vertices, we're going to start building a tree. And at any point, we will have a bunch of edges that are going to be part of our spanning tree-- that means that the edges don't have any cycles among them, they're a so-called forest, but they're not yet connected.

And at each stage in this procedure, we're going to look at the connected components of the graph that we have at this moment, and color them black or white. And then look at the gray edges. So a grey edge is defined to be an edge where one end point is black, and the other end point is white. And what I'm going to do, at any stage in the procedure as I'm growing my minimum weight spanning tree, is I'm going to look at all the gray edges and pick a minimum weight gray edge.

Well, let's do an example to get this clear. So to begin with, I don't have any edges. All I have are the isolated vertices. So it means that I have six connected components, each of which is a single vertex with no edges. That says that I'm allowed to color them black and white in any way I choose, and I will do that. The only constraint on the coloring is there has to be at least one black component, and one white component. So there's an arbitrary coloring-- I've colored two of the vertices white, and the other four black.

Now, in this particular coloring-- I could've chosen any one, but I chose this one-- where are the gray edges? Well, I've highlighted them by thickening them. So this is a gray edge, because it's black and white. This is a gray edge because it's black and white-- black and white, black and white.

This is not a gray edge, because it's white and white. This is not a gray edge, because it's black and black. So that's a simple enough idea.

Now what I'm supposed to do is among my gray edges, pick the one with the minimum weight. Well, if you look at the weights of the gray edges, I got a four, a four, a nine, a seven, and a two. The two is the minimum weight gray edge. I'm going to choose that to start building my tree.

So at this moment, once I've committed to that magenta edge, what I now have is a graph with five components-- namely the component defined by this edge, with two vertices. And the other four isolated vertices, which still don't have any edges connecting them in the structure of magenta edges that I'm building to be my minimum spanning tree.

So according to the rules now, with these five components, I can recolor them. And as long as I recolor them in a way that this component gets the same color-- there's a recoloring. I've made both of those vertices in this component black, and the other four vertices-- which are isolated components-- I can color arbitrarily. So here's my new coloring.

Now, again, once I have this coloring, I can proceed to identify the gray edges. There they are. And this time there are only two gray edges, because I chose to have only one white vertex. There's a gray edge and there's a gray edge. And of course, the minimum weight among the two gray edges is three. So that's going to be my next edge in my minimum weight spanning tree that I'm growing.

What's next? Well now, I have four components left. Here's one component defined by that edge, here's another connected component defined by that edge. And these two vertices are isolated, still, so they're components all by themselves. And the rule is, recolor in such a way that both of these vertices in that component have the same color. All the vertices in this component have the same color-- I could switch them from black to white, in fact I will-- and those can be colored arbitrarily. Let's do that.

There's a recoloring. Now this component is all white, that component is all white. These two separate components happen both to be black. I could have colored one of them white, and one of them black. I need to have one black, given the other commitment to colors.

So now, again, we could find a minimum weight edge, a grey edge I guess it would be. There are two ties for minimum, both of those ones. And I proceed in this way, and I wind up with this minimum weight spanning tree. That's the procedure.

Now I haven't discussed why it works yet, and that is explained in the notes. But we're going to hold off on that and just examine applying this algorithm. So there are a bunch of ways, now, to grow a minimum weight spanning tree. One way is to start at any vertex, and then keep building around that vertex. So you start with that vertex and color it black, and everything else white. That means that all the gray edges are going to be connected to that

vertex.

So pick a minimum weight. Now you have a component with two vertices. Color it black and everything else white, and in that way, you keep working on one component that you're going to grow by always coloring it one color, and everything else the other color. This is a method known as Prim's algorithm for growing a minimum weight spanning tree.

Another one is to globally, among all the different connected components, find a minimum weight edge among them. So what that means is that you find the minimum weight edge among all the connected components, and then having identified where that minimum weight edge is, you can color one of its components black, and the other one white, and that will have to conform to our procedure for picking a minimum weight edge between different colored components. That's Kruskal's algorithm.

And finally, you can grow the trees in parallel. You can just start choosing the minimum weight edge around each connected component, because you can always take a connected component, color it one color, and color all the other edges another color. And so all of the edges touching a given component will be gray in that color, and you can choose the minimum one and grow that component.

And if they're not too close to each other-- so that your choice of edges doesn't conflict-- you can grow these trees in parallel. So I call that, jokingly, Myer's procedure. And that is the application of this coloring approach to finding minimum weight spanning trees.