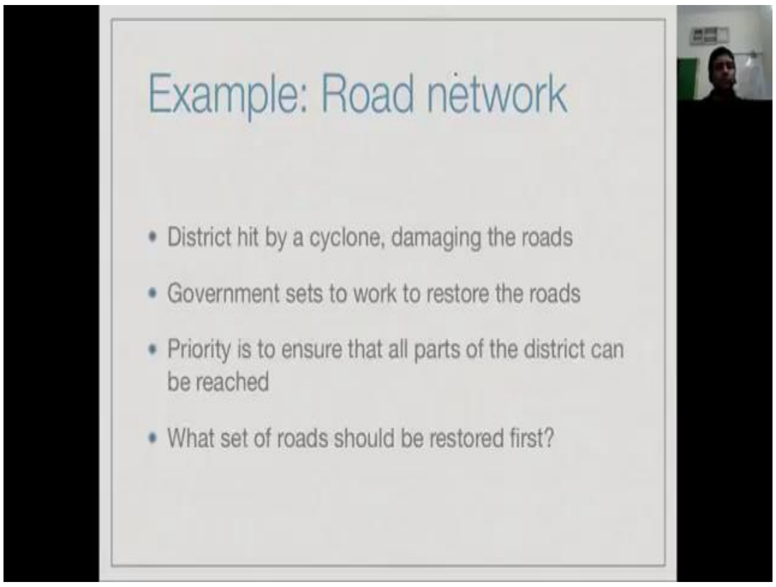
Design and Analysis of Algorithms, Chennai Mathematical Institute
Prof. Madhavan Mukund
Department of Computer Science and Engineering,

Module – 05
Lecture - 29
Minimum Cost Spanning Trees

Having seen a variety of algorithms for shortest paths on weighted graphs, we now move to a completely different problem that of computing, what is called Minimum Cost Spanning Tree.

(Refer Slide Time: 00:12)

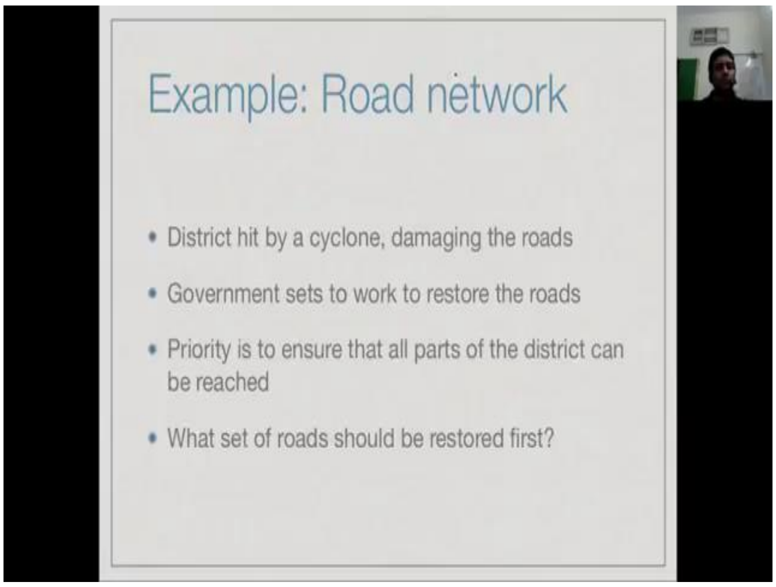


Example: Road network

- District hit by a cyclone, damaging the roads
- Government sets to work to restore the roads
- Priority is to ensure that all parts of the district can be reached
- What set of roads should be restored first?

So, to motivate the problem, let us consider the following example. Suppose, we are in a district which has a road network and after a bad cyclone, the roads are all being damaged. So, the first priority of the government is to restore the roads, so that relief can be sent to various parts of the district and also people can start moving around again. So, the priority is to restore enough roads, so that everybody can move around. So, the first criteria for the government to restore road is to ensure connectivity. So, given this which set of roads should the government restore first?

(Refer Slide Time: 00:48)



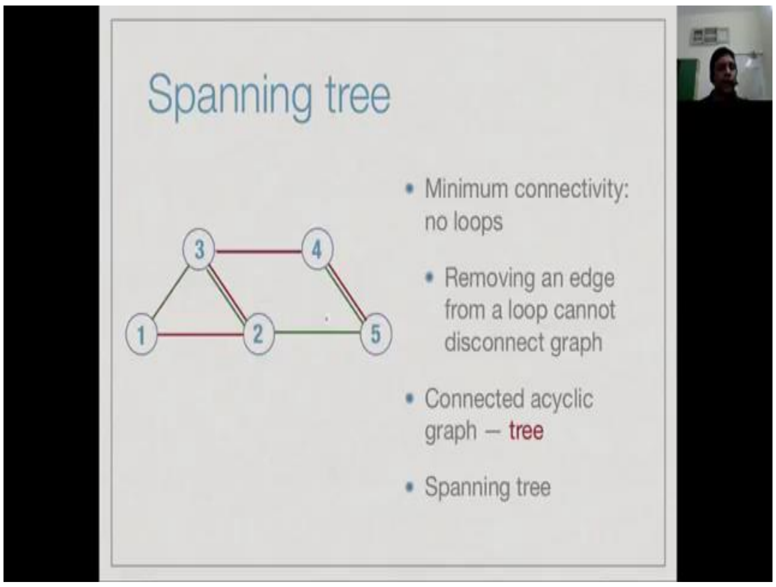
Example: Road network

- District hit by a cyclone, damaging the roads
- Government sets to work to restore the roads
- Priority is to ensure that all parts of the district can be reached
- What set of roads should be restored first?

So, if the main criterion is minimum connectivity, then it should be clear that there is no point in restoring roads which find a loop. For instance, supposing we restore all these four roads, then we could have deleted any one of these roads, say 3 to 4 or 2 to 3 and still one can get from any of these four towns to any four other towns in the district. So, removing an edge from a loop cannot disconnect a graph and our aim is to find some sub set of edges within this graph which are connected in such a way that this is a minimal such set of edges.

So, what we want is a connected sub graph of this original graph which does not have any loops which is acyclic and this is precisely what is called a tree. So, tree by definition is a connected acyclic graph. And in particular, we start to the arbitrary graph and we are looking for a tree which sits inside the graph, which is a sub graph in terms of the number of the edges, which connects all the vertices in the original graph. So, such a tree is called a spanning tree, it spans the vertices of the original graph, but it forms a tree outer the sub set of three edges.

(Refer Slide Time: 02:00)

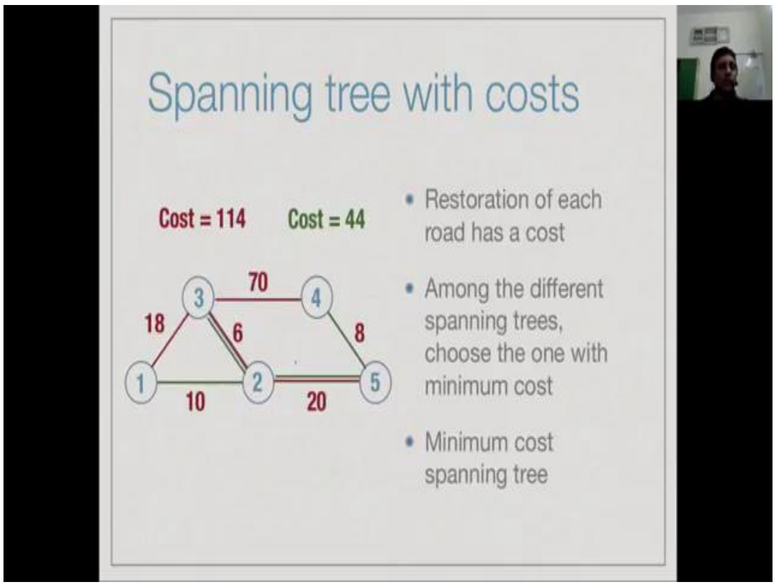


Spanning tree

- Minimum connectivity: no loops
  - Removing an edge from a loop cannot disconnect graph
- Connected acyclic graph – tree
- Spanning tree

So, in this graph for instance, one spanning tree we could form, other red edges shown here, 1 to 2, 2 to 3, 3 to 4 and 4 to 5. Of course, we could form other spanning tree for instance this is green one, this is 1 to 3, 2 to 3, 2 to 5 and 4 to 5. So, there are many possible spanning trees that one can construct on a given graph.

(Refer Slide Time: 02:22)



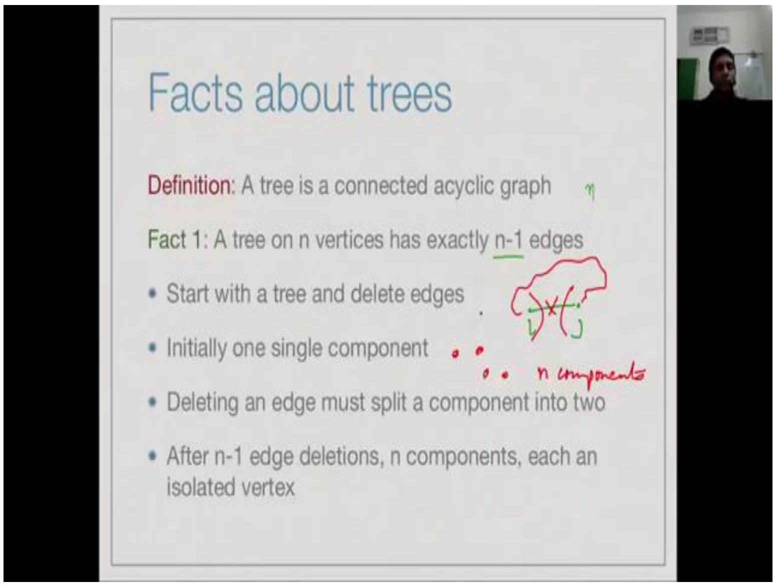
Spanning tree with costs

- Restoration of each road has a cost
- Among the different spanning trees, choose the one with minimum cost
- Minimum cost spanning tree

Now, suppose that the graph also has weights. In this example, the weight for instance could be the cost of repairing a road. So, supposing restoring the road has a cost and now the government would like to not only restore connectivity, but do it I mean at minimum cost. So, if for instance, the government chose to repair this tree of roads, then the total cost is 18 plus 6, 24 plus 70, 94 plus 20, 114. So, it could incur cost of 114 to the store, this spanning tree.

On the other hand, if the government chooses green spanning tree, then the cost reduces to 10 plus 6 is16 plus 20 is 36 plus 8 is 44. So, different spanning trees now will come at different cost and the goal would be to reduce the cost per minimum. In this particular example, you can check this green tree, which has cost 44 is actually the minimum cost spanning tree on this particular graph.

(Refer Slide Time: 03:22)



Facts about trees

Definition: A tree is a connected acyclic graph

Fact 1: A tree on n vertices has exactly n-1 edges

- Start with a tree and delete edges
- Initially one single component
- Deleting an edge must split a component into two
- After n-1 edge deletions, n components, each an isolated vertex

So, before we move ahead to algorithms to compute minimum cost spanning trees, let us look at some basic facts about trees. So, remember that by definition, a tree is a connected acyclic graph. So, the graph in general will have n vertices, so the claim is that any tree has exactly n minus 1 edges. So, this is very easy to prove, there are many different ways of proving it, here is one way of thinking about it.

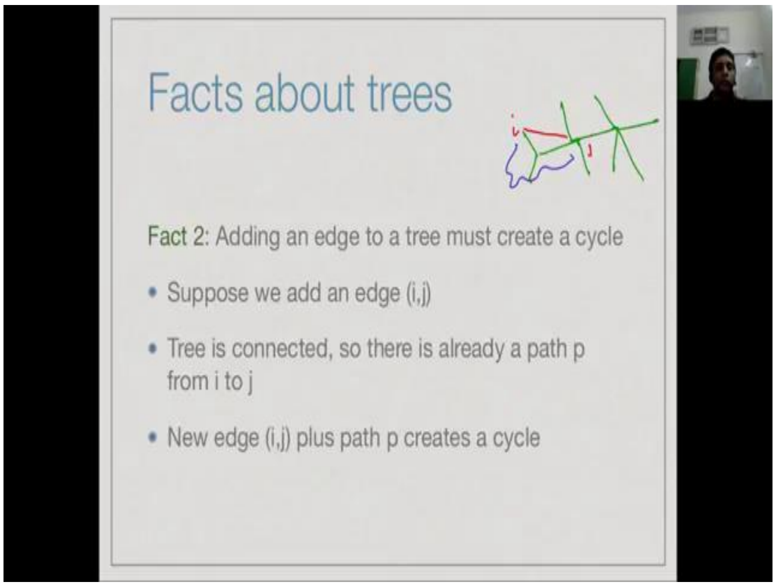
So, supposing we have a tree, so initially the tree is connected by a definition, so the entire graph forms one connected component. Remember, that when we read breadth first and depth first search, we said that we can take a node and look at everything connected to it and it forms a connected (Refer time: 04:10). So, this tree defines one single connected component, if you look at it as a graph in isolation.

Now, because it is a tree, if I have an edge from i to j, there cannot be any other path going from i to j by some other edges, because if not that path plus this edge would form a cyclic. So, if there is an edge from i to j and I remove it, then by definition this component containing i and component containing j must get disconnected. So, if I started with one component, now I get two components. So, I delete the first edge from the graph from my tree and I have one component more.

Now, I delete one more edge by the same argument whichever component that edge belongs will split again. So, each time I delete an edge, I increase a number of components I have, but then, I know that in the end, if I have n isolated vertices, I can have utmost n component. So, I cannot have more than n components, if I have n vertices. So, I can only do this deletion n minus one times.

So, I start with a tree, I keep deleting edges until everything is disconnected, I can only do this n minus 1 times and I must do it n minus 1 times get everything disconnected therefore, the tree must had exactly n minus 1 edges.

(Refer Slide Time: 05:22)



Facts about trees

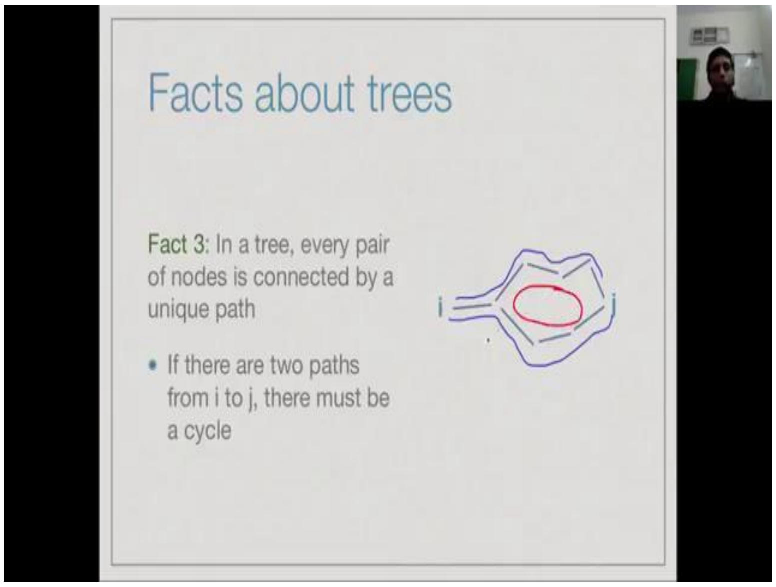
Fact 2: Adding an edge to a tree must create a cycle

- Suppose we add an edge (i,j)
- Tree is connected, so there is already a path p from i to j
- New edge (i,j) plus path p creates a cycle

Now, if I take a tree and then I add an edge, it must create a cycle, we already saw this in this previous argument that we said, so supposing I have a tree, so a tree in general looks something like this. So, it is a graph, it has a kind of more cycles, but many things branching out. Now, if anywhere if I create a tree, supposing I add them and supposing I take some i there and some j here, we may decide to add an edge.

So, we know that because it is a tree, there is already connection. So, there is some path which in this case to this vertex from i to j. So, therefore, that path p plus this edge forms acyclic. So, in a tree I have exactly n minus 1 edges and when I add any extra edge, no matter which edge I add, it will definitely form a cycle.

(Refer Slide Time: 06:06)



Facts about trees

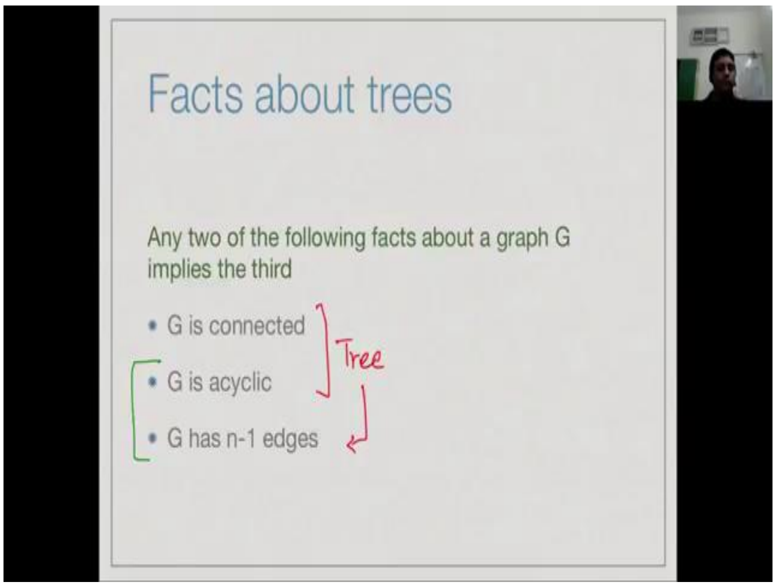
Fact 3: In a tree, every pair of nodes is connected by a unique path

- If there are two paths from i to j, there must be a cycle

So, another consequence of all these definitions is that between any two paths, any two vertices in a tree, there can only be one unique path. So, supposing there are actually two paths, so let us look at two vertices, here we have drawn them as i and j and suppose there are two parts. So, if I follow the two parts, then it is very clear that because there are two different ways are going there, there will be some loops somewhere in between.

So, notice that it need not to be a loop including i and j, it could be somewhere in between i and j, but if you consider all the cases carefully, you can convince yourself, there is no way to have two distinct paths from i to j without creating a loop. And we have a loop, then the graph is no longer acyclic, so it is not a tree which is our assumption to be given.

(Refer Slide Time: 06:52)



Facts about trees

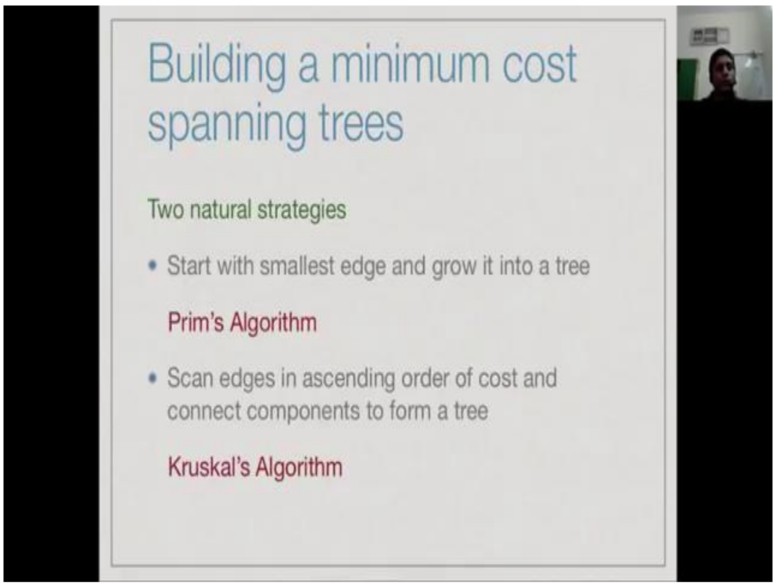
Any two of the following facts about a graph G implies the third

- G is connected
- G is acyclic
- G has n-1 edges

So, we are actually the following claim that we have these three properties, that G is connected, G is acyclic and G has n minus 1 edges, then any of these two implies the third. So, G is connected and G is acyclic by definition it is a tree, we have just shown the first arguments that any tree has n minus edges. So, the fact that the first two imply the third is what we have already shown.

Now, you can easily convince yourself to find a formal prove that G is acyclic and has n minus 1 edges then in fact, it must be connected, everything must be connected everything. And finally, if G is connected and it has n minus 1 edges, then it can be acyclic graph, it cannot have any graphs. So, these are varies ways of looking at trees and sometimes we might use one property or another property. So, it is useful to keep these things in the back of our mind when we talk about trees in general.

(Refer Slide Time: 07:45)



Building a minimum cost spanning trees

Two natural strategies

- Start with smallest edge and grow it into a tree

  Prim's Algorithm

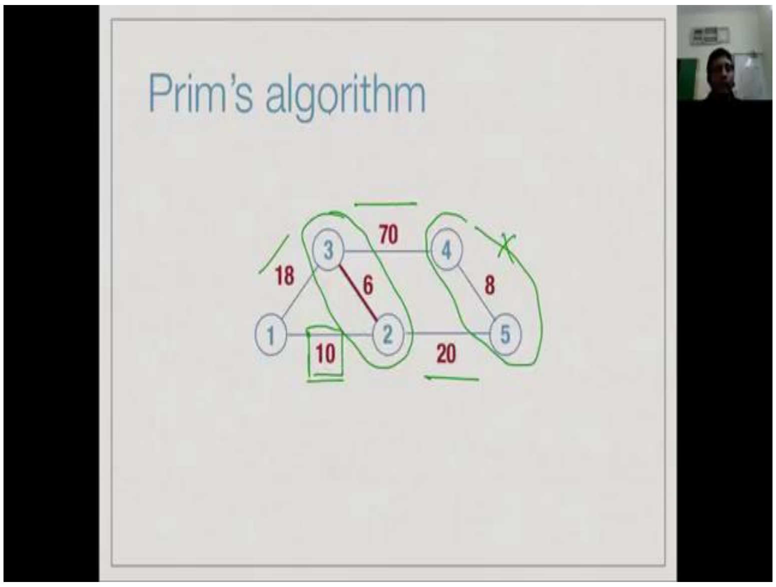
- Scan edges in ascending order of cost and connect components to form a tree

  Kruskal's Algorithm

So, our target right now is to build a minimum cost spanning tree. So, there are two naturally greedy strategies that one can think of. One is, since you are looking for a minimum cost tree to start with this smallest stage and incrementally build the tree. So, we keep adding edges to the existing tree, so that the new graph remains a tree and it grows as little as possible it terms of cost. This will give raise to an algorithm which is called prim's algorithm. It will also turn out to be very similar to Dijkstra's shortest path algorithm with a single source.

The other strategy we can have is to look at edges in ascending order of cost and keep adding them, so long as we do not violate the tree property. Now, this is different from Prim's Algorithm, because here we do not built a tree to start with, we keep adding edges, so that we do not create a cycle, but we could have disconnected groups of edges, but eventually there will all connected to form a tree. So, we will see these in more detail in the next 2 lectures, but let us just look at an initiative example of how these two strategies work.

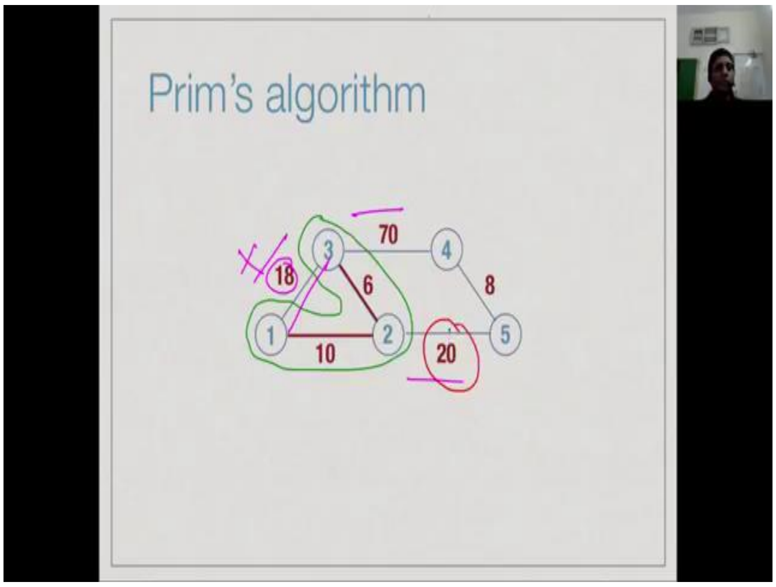
(Refer Slide Time: 08:54)



Prim's algorithm

So, let us look at these two algorithms intuitively will come to them more detail later, so let us start with Prim's Algorithm. Remember that the strategy in Prim's Algorithm is to start with the smallest weight edge and then incrementally grow a tree. So, we start with a smallest edge here which is the edge weight 6 between 2 and 3. Now, we have to look at the existing tree which consists of this order the graph and decide whether to add one of these four edges to extend it. We cannot add that edge over here, we cannot add this edge, because it would not form a tree, it would be disconnected from this edges. So, we can add any of these, but we choose the smallest one. So, in this case, we choose the edge with weight 10.

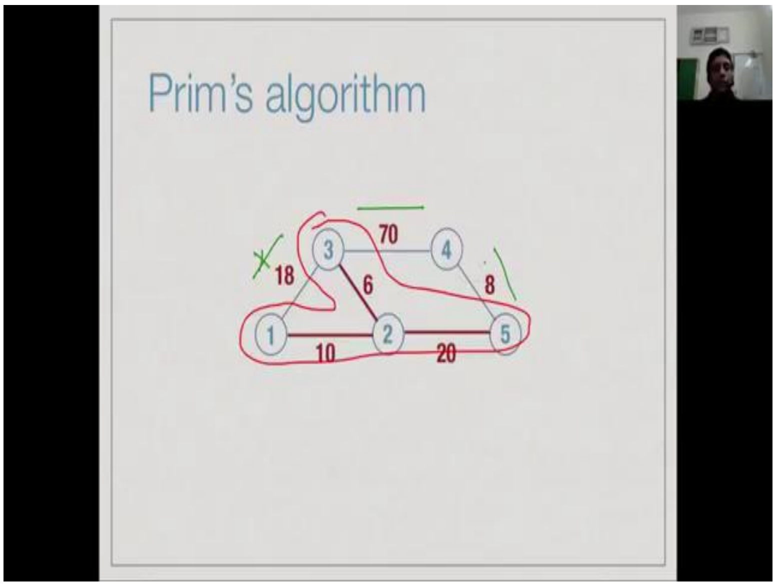
(Refer Slide Time: 09:37)



Prim's algorithm

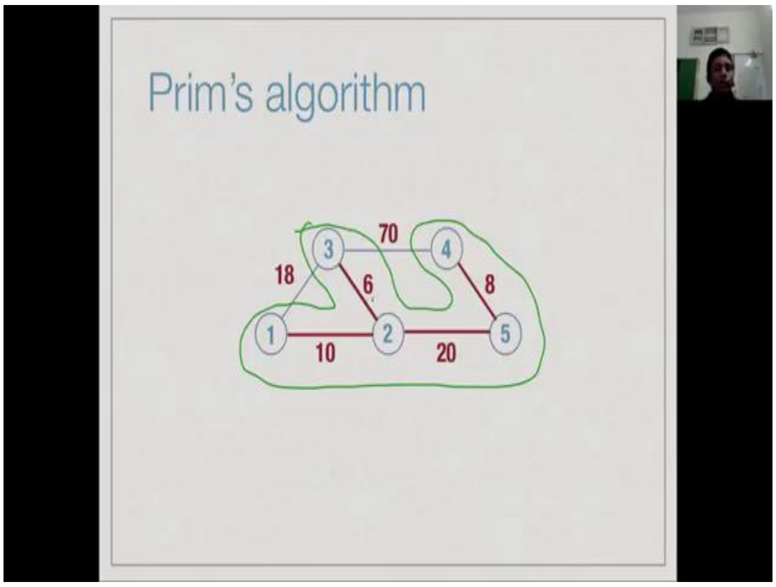
So, the next step in the tree is to add the edges 1, 2 and now, we have this tree. Now, if we look at possible edges that we can add, we have this edge, we have this edge and we have this edge. Now, the smallest among these is the edges with wait 18, but if I had that we get a cycle. So, this is not a good edge to add. So, therefore, we must add one of the other 2, again to pick the smaller one, which in this case is the edge labeled with weight 20.

(Refer Slide Time: 10:06)


Prim's algorithm

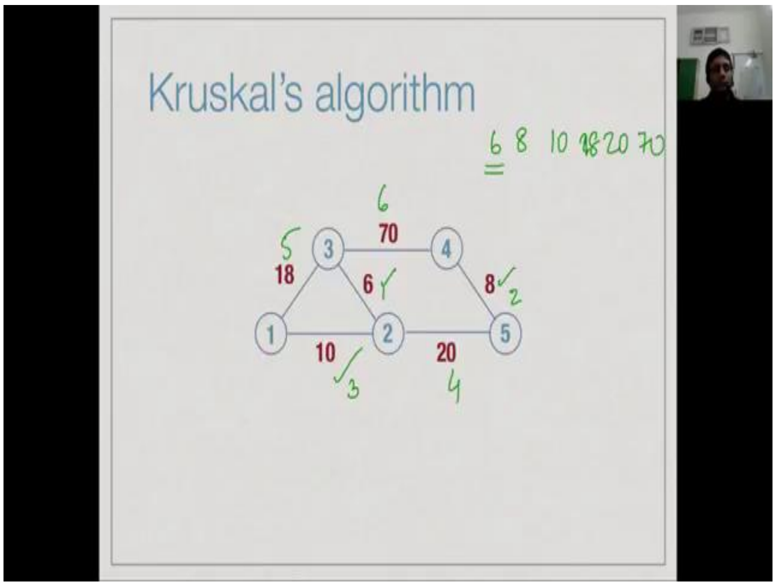
So, then we get this tree which has now this shape, this is a given tree. Now, we can add, we cannot add this is we know. So, we can either add the edges 70 or the edges labeled with weight 8 and obviously 8 is smaller.

(Refer Slide Time: 10:23)


Prim's algorithm

So, finally, we add that in this is a tree that we can get. So, this is the final tree, we get from Prim's Algorithm, let us starting from the smallest edge and incrementally growing the tree.

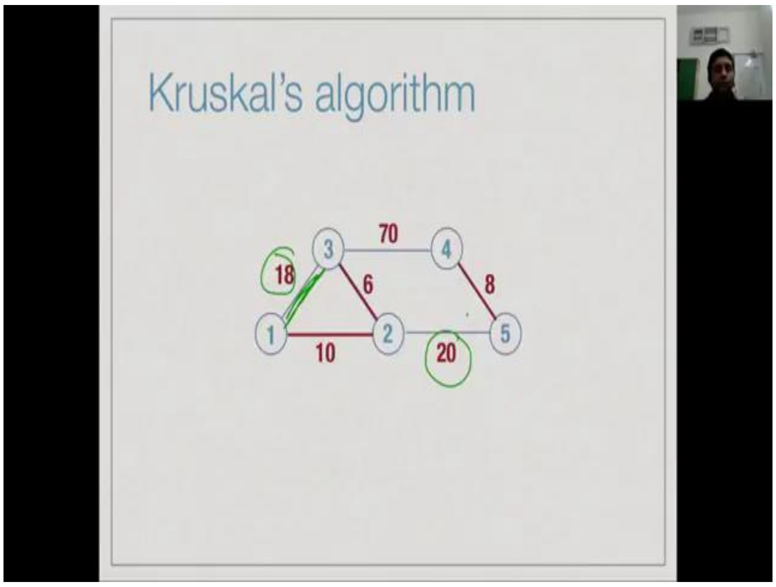
(Refer Slide Time: 10:35)



Kruskal's algorithm

The other strategy we said was to start with the edges in ascending order. So, we start with edges with 18, then 8, then there mean. So, we have this is the first one, this is the second one and this is third one and so on. So, we consider the edges in this order, 1, 2, 3, 4, 5 and 6. So, we have the weights in 18, 20, then kindly 70. So, every 6 edges when we consider them in this order, so among these of course, if small f 6, so we add.

(Refer Slide Time: 11:07)



Kruskal's algorithm

So, this is the starter of our tree, now the next one is 8 and that does not form a cycle, it does not violate a tree property. So, we add that, notice now the crucial difference between Prim's and Kruskal's algorithm at this point, we do not have a tree, we have two separate trees in some sense. So, we have two different acyclic component within this graph which are not connected each other, but we are just going a order of an edges.

So, next we will see 10 is the next edges that we can add, this does not form a cycle. So, we add that, so in some sense, we are grown this component and left that component grown. Now, the next one would be 18, but if you add 18, it would form a cycle. So, we skip 18, we move to the next one, this is 20, 20 is fine and 20 will in fact, connect the two component to form a tree. So, we add 20, now we are done, because we have added n minus 1 edges, there are five vertices, we got 4 edges and therefore, we are definitely got a tree.