Advanced Operations Research

Prof. G. Srinivasan

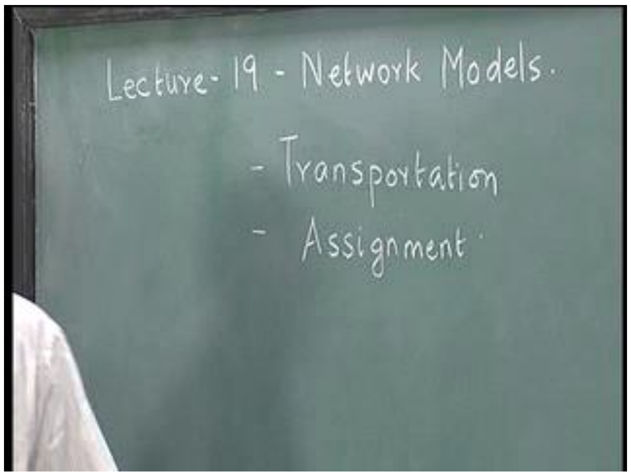
Department of Management Studies

Indian Institute of Technology, Madras

Lecture- 19
Network Models

In this lecture, we will discuss network models.

(Refer Slide Time: 00:16)



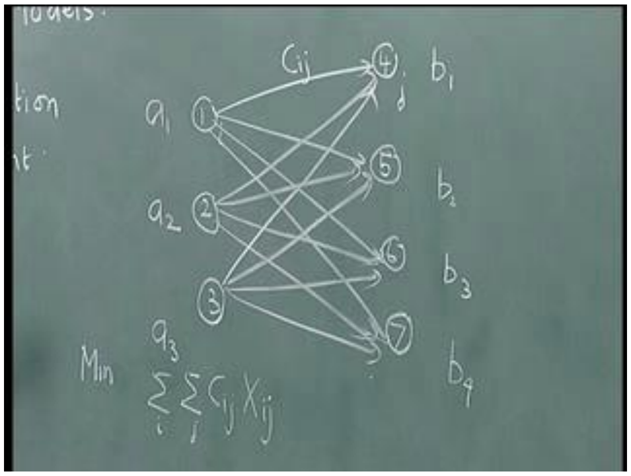
Lecture- 19 - Network Models.

- Transportation
- Assignment.

What is a network model?

We have already seen the transportation problem and the assignment problem. Let us take the transportation problem, first.

(Refer Slide Time: 00:46)



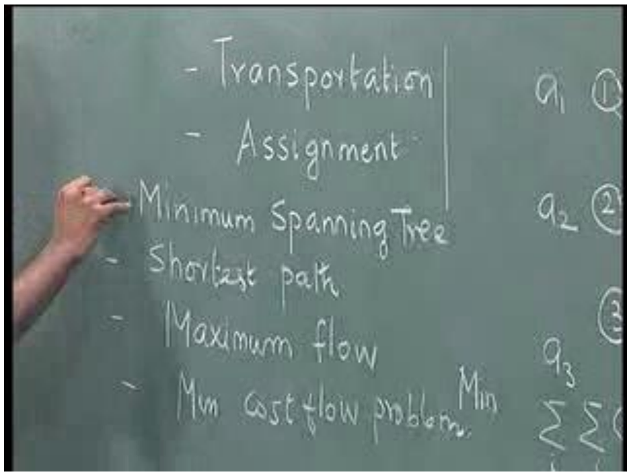
A simple transportation problem would mean we may have three supply points and four demand points or destination points. In a transportation problem, we say that the availabilities are a_1, a_2, a_3 in these three supply points and b_1, b_2, b_3, b_4 in these four demand points. We also have a cost of transportation, which is given by C_{ij}, which is the cost of unit transportation from i to j.

The problem is: Given the a_1, a_2 and a_3 which are the supplies which have to be transported to b_1, b_2, b_3 and b_4 how much do we send from i to j, which is X_{ij} such that the total cost of transportation which is sigma $C_{ij}X_{ij}$ summed over i and summed over j is minimized. This is the transportation problem and this is an example of a network problem because we have represented the problem on a network.

In this transportation problem we assume that each of the supply points is actually connected to each of the requirement or destination point. This is the graph or the network that represents the transportation problem. In fact, we have already seen this earlier in the fundamentals course. We have also seen the similar representation of the assignment problem.

What are the network models? Are there network problems other than the transportation and assignment problems? What are the problems we are going to see in this advanced operational research course?

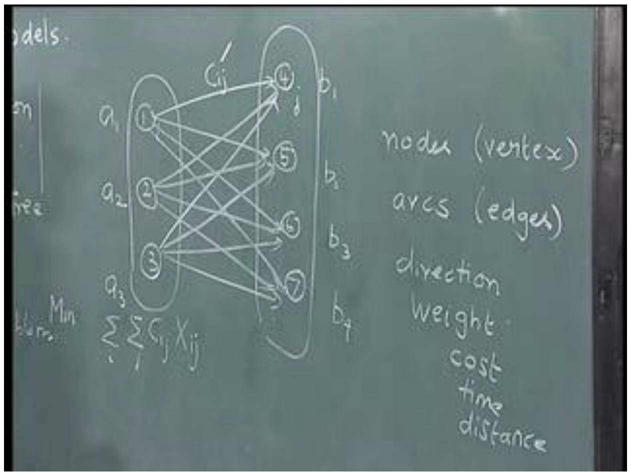
(Refer Slide Time: 03:12)


- Transportation
- Assignment
- Minimum Spanning Tree
- Shortest path
- Maximum flow
- Min cost flow problem.

a_1
a_2
a_3
Min
$\sum\sum$

Since we have already seen the transportation and assignment problems we will not go deeper into them. We will look at some more network problems namely: one is a minimum spanning tree problem, we will see the shortest path problem; we will see the maximum flow problem; we will see the minimum cost float problem. We will look at all these four new problems under the network models and we will see more than one algorithm or more than one way of solving each of these problems. Before we begin the minimum spanning tree which is the first one that we consider? We also understand that for every network problem there is an underlying graph or a network. A graph from basic definitions is a collection of nodes and arcs.

(Refer Slide Time: 04:25)



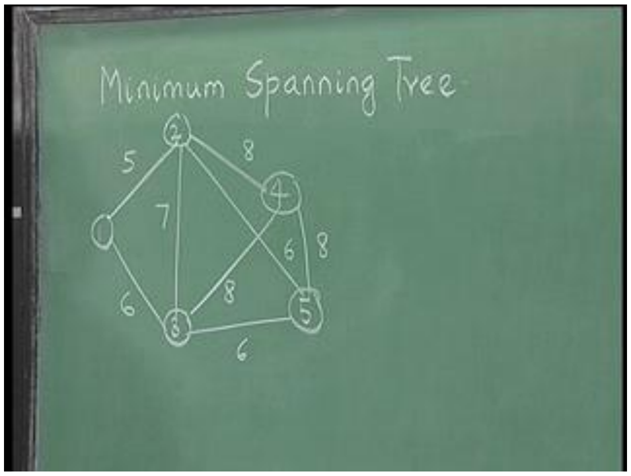
Each of these vertices is called a node. A node is also called the vertex or vertices. Arcs are the connectivities between the nodes and these are also called edges. This graph or a network is also characteriz~~s~~ed by two things: one is the direction that there is an arc from i to j, specific to the transportation problem, we do not have arcs from j to i but there could be other network problems where we could have an arc from i to j and another arc from j to ~~I~~i; another thing that characteriz~~s~~es this network is this weight which in the context of the transportation problem, represents the unit cost of transportation from i to j where there is an arc from i to j.

This weight can be a cost, it can be time, it can be distance or it can be any other number that represents some form of relationship among these nodes and are represented as weights of the arcs. The network problems that we will be looking at will be based on graphs which have nodes and arcs. Most of the network problems will be directed network problems. Only one of them that we will see is the minimum spanning tree is not directed. There is a weight associated with every arc in the network.

There are also some connectivity issues. For example, in the transportation problem you can always separate the transportation problem into two sets: set of supplies and set of destination. There are arcs only from a supply node to a destination node. You do not find an arc from a destination node to a supply node. You will not find arcs that connect the supply nodes. You will not find arcs that connect the destination nodes and so on. This is a special type of a graph which is called a bipartite. But the network problems that we are going to look at, in addition to transportation and assignment which we have already seen now, all four of them will have networks which are comprised of nodes and arcs. They will have weights, but for the minimum spanning tree; the rest of them will have directed arcs.

The first network problem that we will look at is called the minimum spanning tree. We will look at the minimum spanning tree using a particular example. I will also try and describe what a minimum spanning tree is.

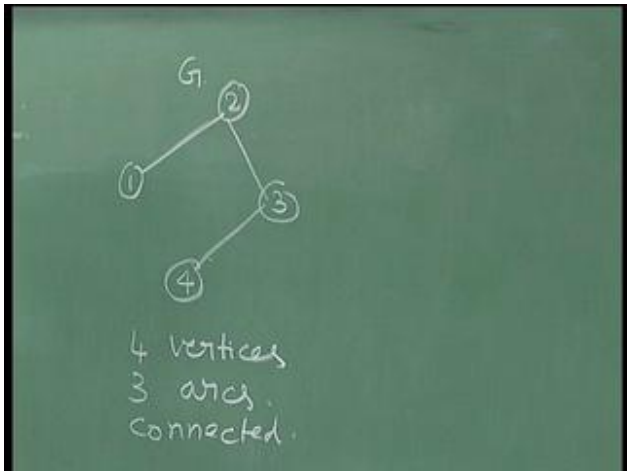
(Refer Slide Time: 07:35)



Let us consider this network. This network has five nodes or vertices which are 1, 2, 3, 4 and 5 and it has several arcs that connect these nodes or vertices. For example, 1 to 2 is an arc. Each arc has a weight, for example: arc 1- 2 has weight equal to 5, arc 2-3 has weight equal to 7, and so on. This network is not directed. There is no specific direction saying that the arc is from 1 to 2. Now, this 5 would mean that 1 and 2 are connected, so it would represent both 1 to 2, as well as 2 to 1. What is a minimum spanning tree problem? Before we get into the definition of what is a minimum spanning tree, I will also have to define, what a tree is? What is a spanning tree and then what is a minimum spanning tree?

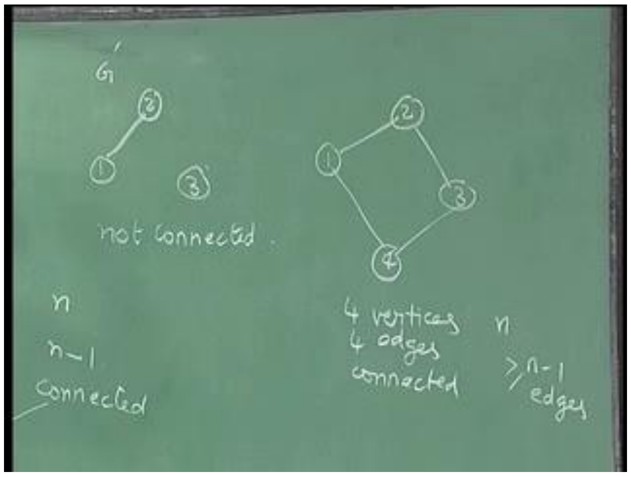
Let us consider a few simple graphs and try to understand the concept of a tree.

(Refer Slide Time: 09:32)



Let us look at this graph which I call as G. This graph has four vertices, three arcs and it is connected. Now, I have to define what connected graph is? If I take any i j, say 1- 2 or 1-3 or 3-4 and 3-4 there is a path, I can go from i to j. For example, if I look at 1-4, then there is a path 1to 2, 2 to 3, 3 to 4. For every i-j there is a path such a graph is called a connected graph. This has four vertices, three arcs or edges and is connected.

(Refer Slide Time: 10:25)



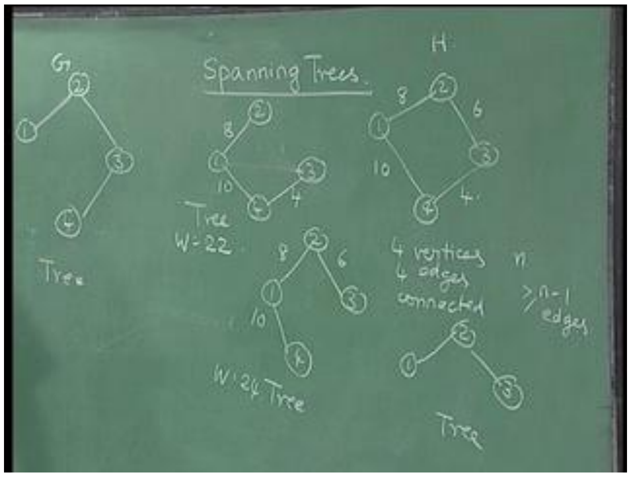
For example, if I consider another graph like this, which I may call G dash, there are three vertices 1, 2 and 3. There is only one edge which connects 1 and 2. This is not connected because there is no path between 1 and 3. So, this is not connected. If I consider another graph like this, 1 to 2, 2 to 3, 3 to 4 and again 4 to 1. This graph has four vertices, four edges and is connected. It is possible to find a path from every i, j on this. This one is called tree and this one is not a tree.

What ~~characterises~~ characterizes a tree?

In a given graph, if it has one vertex, it has n minus 1 edges and it is connected, then it becomes a tree. Now, this is a tree, this is not a tree, even though for one vertex it is connected but it has more than n minus 1edges. It is also obvious that for a graph to be connected and for the graph if it has n vertices, then for it to be connected it should have greater than or equal to (n minus 1) edges and so on. Given these graphs we understand that this is a tree. We get into other questions. Can the graph itself be a tree or can I get a subset from a graph which is actually a tree?

Let us go back to this particular graph.

(Refer Slide Time: 12:28)



The earlier one we showed here, this is the graph G and this itself is a tree. If I take a graph like this, say, I call this as H. Then, I may get a subgraph from this; now, this is a tree. From a given graph, I may get a subgraph which is a tree. From this same graph, I may get a subgraph, now this is also a tree. The difference is, this subgraph is a tree; it has four vertices; it has three edges and it is connected. This is also a tree. This has three vertices, two edges and it is connected. Both these graphs are subgraphs out of this. This is an example where the given graph itself is a tree. So we can ~~generalise~~ generalize this by saying that if the given graph is a tree, then there is only one tree; whereas, if the given graph is connected and it is not a tree, it is possible to get subgraphs from the given graph, which are trees.

From this there are four vertices. This is a tree with four vertices. This is a tree with three vertices. For example, this is a given graph, so this is also a tree with four vertices and three edges which is different from this tree. So, when the given graph is connected and it is not a tree, then it is possible to get trees more than one which has exactly the same number of vertices as the original graph. It is also possible to get trees more than one where, the set of nodes or vertices is a sub set of this. This is an example where this is a tree but, it does not contain all the vertices of the original graph. This is an example where it is a tree; it contains all the vertices of the original graph. Those trees which are taken from a given graph, which contain all the vertices are called spanning trees.

We take this graph H, this is a spanning tree, this is not a spanning tree, though it is a tree which is pulled out from this graph H. So given graph H, if it is connected, may have more than one spanning tree. This is one spanning tree, this is one spanning tree, this is not a spanning tree so we are not interested in looking at this because the emphasis is to find out a spanning and then describe what a minimum spanning tree is. Given a graph like this, we are right now interested in spanning trees, either this or this. We are not interested in this tree because this tree is not a spanning tree.

We can have more than one spanning tree from a given graph. Let us try and add some weights into this graph. Suppose I say, the weight here is 8, the weight here is 10, the weight here is 6 and the weight here is 4. If I look at this spanning tree and simply put the weights associated with this, this spanning tree has total weight equal to 10 plus 8 is equal to18 plus 4 is equal to 22. If I look at this spanning tree I get 8, 10 and 6. So, W is equal to 24, 10 plus 8 is equal to 18 plus 6 is equal to 24. If the graph is weighted and has weights, then every spanning tree that you can pull out, remember that we can pull out more than one spanning tree.

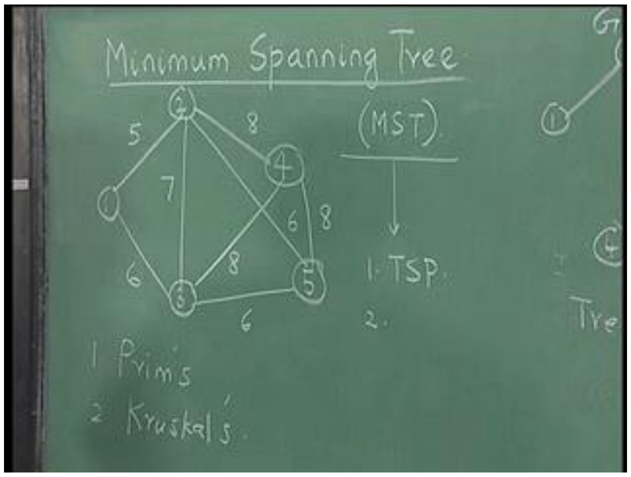
So, every spanning tree that we can pull out now has an associated weight. This spanning tree has weight equal to 22. This has weight equal to 24. Between these 2 spanning tress, this has minimum weight. What is a spanning tree that I can pull out of this that has minimum total weight? So, given a weighted graph that is connected, now what is a spanning tree which has minimum total weight is called the minimum spanning tree problem or more popularly called as MST problem. MST would mean minimum spanning tree.

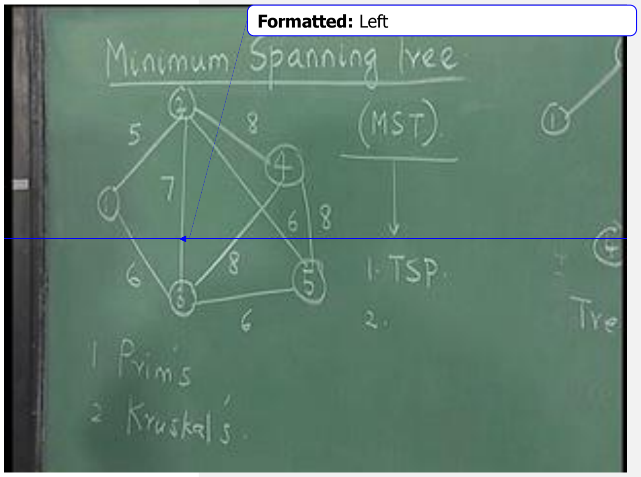
For this one, we just picked up, evaluated two spanning tress and evaluated their weights, and said this was better than this. But, we do not know or we have not shown at the moment that this is perhaps the best. Do we have algorithms which can give us the minimum spanning tree is a next question?

We now go back to this graph which has five nodes.

Formatted: Left, Space Before: 0 pt, After: 10 pt, Line spacing: Multiple 1.15 li, Tab stops: Not at 1.81"

(Refer Slide Time: 18:16)





Formatted: Left

The question is to get another subgraph or a tree or a spanning tree which has five nodes, which has four arcs and it is connected such that the sum total of the weights of the arcs is ~~minimised~~minimized. Such a spanning tree is called the minimum spanning tree.

Formatted: Left, Space Before: 0 pt, After: 10 pt, Line spacing: Multiple 1.15 li, Tab stops: Not at 1.81"

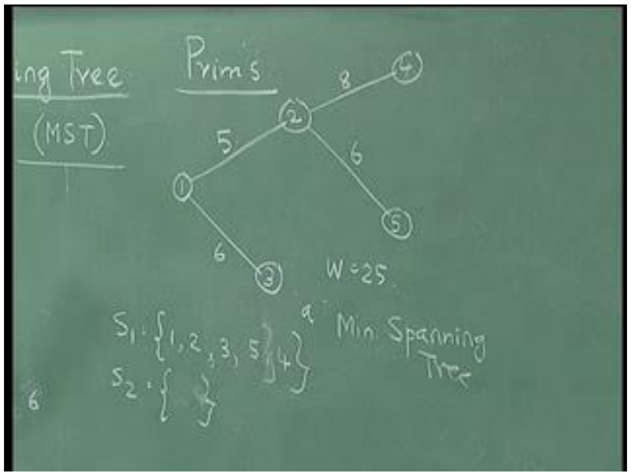
What we will do now is to look at two algorithms which are very popular, which are used to get minimum spanning trees out of a given network. These two algorithms are called the Prim's algorithm and the Kruskal's algorithm. First, we will describe the Prim's algorithm and then the Kruskal's algorithm. Before that we come back to the problem again.

Given a graph which is connected, find out a spanning tree which has least weight?

This graph has five nodes, so the spanning tree will have five nodes and four arcs. What is the set of four arcs which will span across the five nodes, retain the connectivity and make it a tree such that the total weight is minimized? That is a minimum spanning tree problem. This problem is important for more than one reason. The minimum spanning tree: In this course, we will find that this has a lot of use when we actually come and solve the Travelling Salesman Problem or the TSP.

In practice, the minimum spanning tree also kind of represents the minimum length of wire that is needed to provide connectivity in this network. For example, if we go back to this as a network, if we want to connect points 1, 2, 3 and 4 and if we choose this configuration for connectivity, then we may require 24 units of length of the wire; whereas, if we look at this configuration, we may require 22 units of the length of the wire. We would obviously be interested in finding out the spanning tree with minimum weight such that minimum length of wire is used to connect these nodes. Minimum spanning tree finds applications in many connectivity problems. It also has applications and use, particularly when we solve the Travelling Salesman Problem and other related problems. With this we go back and first try to do the Prim's algorithm and then the Kruskal's algorithm to find out the minimum spanning tree.

(Refer Slide Time: 21:08)



Let us explain the Prim’s algorithm. The first thing that we do in the Prim’s algorithm is to find out that arc which has minimum weight. Now, arc 1 - 2 has a minimum weight, so put 1 - 2 into the spanning tree with weight equal to 5. There are five nodes or vertices. We have already put 1 - 2 into the spanning tree with minimum weight. Having put 1 - 2 into the spanning tree, we now separate the nodes into two sets. Set S_1, which is a set of nodes 1 and 2, which are into the spanning tree and set S_2 which contain the rest of the nodes 3, 4 and 5, which have not yet come into the spanning tree.

When we start the Prim’s algorithm, we will always start with the cardinality of S_1 will be 2 because we start by putting that arc which has minimum weight into the spanning tree. That arc will connect two nodes always, because in any network problem an arc connects two nodes. This arc 1 - 2 has connected nodes 1 and 2. The set S_1 will have 1 and 2 and the set S_2 will have 3, 4 and 5.

Now, go back to the graph or go back to the data, try and find out the connectivity between this set and this set. If we look at 1, then we have to look at 1 to 3, 1 to 4 and 1 to 5. Now, 1 to 3 exists with weight equal to 6. 1 to 4 does not exist, 1 to 5 does not exist, so the minimum from 1 is 6. Similarly, look at 2: 2 to 3, 2 to 4, and 2 to 5. 2 to 3 exits with 7. 7 is bigger than 6, so keep 6. 2 to 4 is 8. 8 is bigger than 6, so keep 6 and 2 to 5 is 6, which is equal, so you could either take

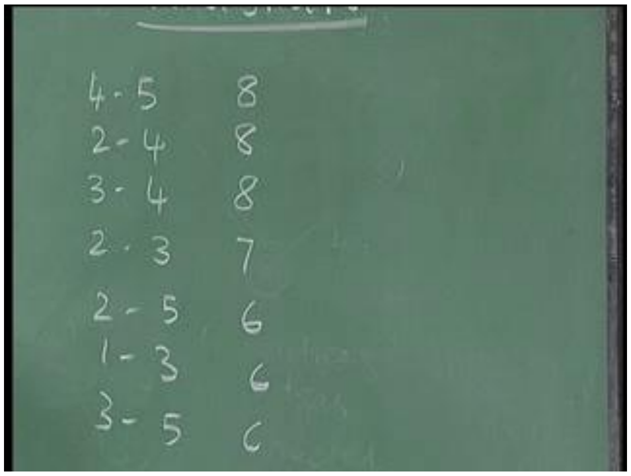
2 to 5 or you could take 1 to 3. Both 1 to 3 and 2 to 5 have weight equal to 6. Choose that which has minimum weight. There is a tie between 1 to 3 and 2 to 6 so, we could take either of them. Let us take 1 to 3, so add 1 to 3 with value 6. The moment you add 3 into the spanning tree, S_1 gets updated to 1, 2, 3 and S_2 is 4 and 5 vertices. Now vertices 1, 2 and 3 are into the spanning tree and vertices 4 and 5 are outside the spanning tree.

Repeat the algorithm. Look at 1 and find out the connectivity between 4 and 5; 1 to 4 there is no connectivity; 1 to 5 there is no direct connectivity, so leave it. Look at 2. 2 to 4 is 8 which is the minimum; 2 to 5 is 6, so 8 gets updated to 6; 3 to 4 is again 8; 3 to 5 is 6. So, we could either pick 2 to 5 which is 6 or pick 3 to 5 which is 6. Let us pick 2 to 5, which is 6. Break the tie arbitrarily, so 2 to 5 is chosen, 5 gets into the network with 6, which means that 5 gets added here so you have 4, which is, left out, 4 has to come into the network.

Again, repeat the algorithm 1 to 4, 2 to 4, 3 to 4 and 5 to 4. 1 to 4 does not exist; 2 to 4 the minimum is 8; 3 to 4 is also 8; 5 to 4 is also 8. We could either choose 3 to 4 which is 8 or 2 to 4 which is 8 or 5 to 4 which is 8. Let us say we choose 2 to 4 which is 8. We will do 2 to 4 which is 8. This means, this set becomes a null set and 4 comes in. All the nodes are in the spanning tree which is indicated by this, which is also indicated by the fact that this has become a null set, which means all the nodes have come into the spanning tree. So, the algorithm stops when all the nodes are into the spanning tree. We have got a spanning tree with weight W equal to 6 plus 5 is equal to 11 plus 6 is equal to 17 plus 8 is equal to– 25. This spanning tree is the minimum spanning tree. It is a minimum spanning tree because we realised that as we proceeded with the algorithm, we found several places where there was a tie. For example, the last one we entered 2 to 4 was entered with 8. We also knew that 3 to 4 was 8 and 5 to 4 was also 8. If we had entered 3 to 4 instead of 2 to 4, we would have got another spanning tree but with the same weight 25.

The Prim's algorithm guarantee a minimum spanning tree with minimum weight, W equals to 25. There are some proofs for the Prim's algorithm, but right now, we do not concentrate on the proofs. We concentrate more on the understanding and the learning the algorithm. I will also try and give you some results, whereby, we try and understand the proof of Prim's algorithm.

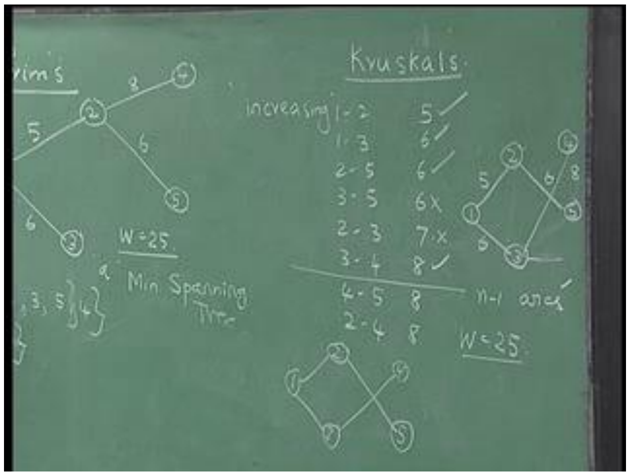
(Refer Slide Time: 27:55)



4 - 5	8
2 - 4	8
3 - 4	8
2 · 3	7
2 - 5	6
1 - 3	6
3 - 5	6

We go to the next one which is called the Kruskal’s algorithm. To do the Kruskal’s algorithm, the first thing we need to do is to arrange these weights in the decreasing order. We have three arcs with weight equal to 8. We could start with any of them. Let us say, we start with 4 to 5 which is 8, 2 to 4 which is 8, 3 to 4 which is 8, then we do 2 to 3 which is 7, then we do 2 to 5 which is 6, 1 to 3 which is 6 and 3 to 5 which is 6 and then we do 1 to 2 which is 5.

(Refer Slide Time: 29:21)



1 - 2	5 ✓
1 - 3	6 ✓
2 - 5	6 ✓
3 - 5	6 ✗
2 - 3	7 ✗
3 - 4	8 ✓
4 - 5	8
2 - 4	8

What we have to do here is we have to arrange them in increasing order. We start with 1 to 2 which is 5 and then we do 1 to 3 which is 6, 2 to 5 is also 6, 3 to 5 is also 6, 2 to 3 is 7, 3 to 5 is 8, 3 to 4 is 8, 4 to 5 is 8 and 2 to 4 is also 8. If we arrange them in increasing order then we start from the first and move down. If we arrange them in decreasing order then you start from the end. We arrange it in increasing order or non-decreasing order to be precise, such that these values are greater than or equal to the previous ones, respectively.

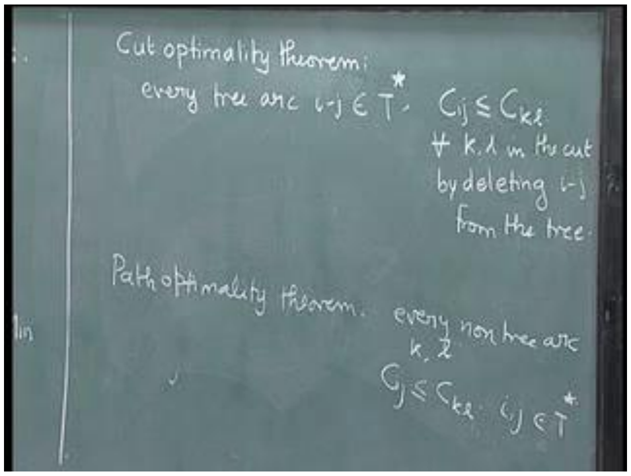
Start the Kruskal's with the first one which is 1 to 2. We start with 1 to 2 with weight equal to 5. Go to the next one which is 1 to 3, so put 1 to 3 with value 6. Now, 2 to 5 is 6, so create 2 to 5 which is 6. Look at the next one which is 3 to 5. If I put this 3 to 5 equal to 6, then what happens is that I am completing a circuit. Therefore I cannot have a tree. A tree should not have a circuit. Adding 3 to 5 is going to create a circuit, so do not add 3 to 5. Ignore this. Go to the next one which is 2 to 3.

Similarly, if I add 2 to 3, I create a circuit. Tree should not have a circuit therefore, avoid 2 to 3. Go to 3 to 4 which is fine, so you could have 4 somewhere, here. We can write 3 to 4, which is like this with value equal to 8, this is done. At this point, we have actually added four arcs into this and make sure that we do not create a closed circuit in the graph. When you do the Kruskal's, the moment we have added n minus 1 arcs, which is 4 arcs, in this case, the algorithm stops and this is a minimum spanning tree with weight equal to 6 plus 5 is equal to 11 plus 6 is equal to 17 plus 8 is equal to 25. We realise that the minimum spanning tree obtained by Prim's and by the Kruskal's algorithm both have the same minimum weight of 25. The difference, of course, is that the spanning tree is different here and the spanning tree is different here.

Here we have taken 3 - 4 with weight equal to 8. Here we have taken 2 - 4 with weight equal to 8. To go back to Kruskal's again, sort the arcs or adjust in increasing or non-decreasing order as shown here. Pick them one after another and put it into the spanning tree with the condition that the property of a tree is maintained as we put it along. To redo this, we could start with 1 - 2, which retains the tree property and 1 - 3 which retains the tree property. There are three nodes and two arcs and so it is a tree. 2 - 5 retains the property. There are four nodes and 3 arcs and it is a tree. 3 - 5 is not added because by doing so I do not get a tree.

I have four vertices and four edges, 2 to 3 is not done. Now, 3 to 4 is included. I have five vertices, four edges and it is a tree. At the end, when we do the Kruskal's the moment n minus 1, arcs or edges are added into the spanning tree, the algorithm terminates and gives us the minimum spanning tree. Both these methods are very useful. Both these methods are very important and used extensively in computation of the minimum spanning tree. Both of them are optimum but how do both of them work? Are there some underlining principles with which we can show that the Prim's algorithm and the Kruskal's algorithm are optimum?

(Refer Slide Time: 34:59)



Cut optimality theorem:

every tree arc i-j $\in T^*$. $C_{ij} \leq C_{kl}$

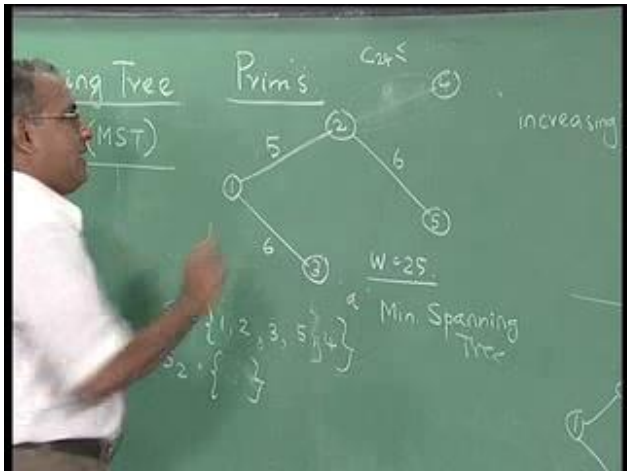
$\forall$ k, l in the cut by deleting i-j from the tree.

Path optimality theorem. every non tree arc k, l

$C_{ij} \leq C_{kl}$. $ij \in T^*$

We can also try and explain two important results: One is called the cut optimality theorem and the other is called the path optimality theorem or condition. What is the cut optimality condition say? It says, for every tree arc i-j belongs to T star, which is this: C_{ij} is less than or equal to C_{kl}, for every k,l in the cut obtained by deleting i-j from the tree. Let us apply this cut optimality condition into the spanning tree that we obtained here. For every i-j which belongs to T, which means, if I take this, this i-j, 2 - 4 belongs to T.

(Refer Slide Time: 36:36)

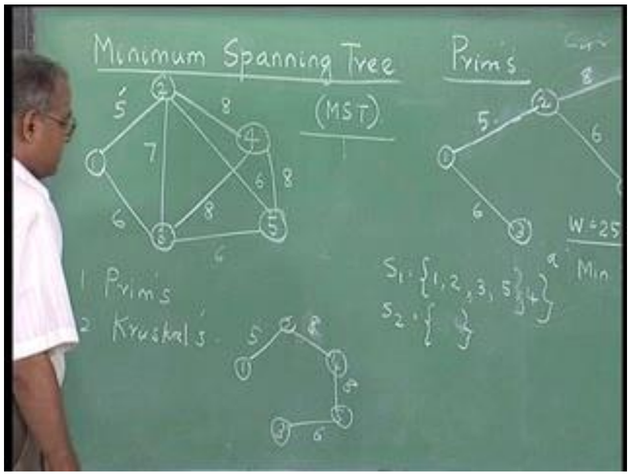


What does the cut optimality condition say?

If I remove this 2-4 and this 8, it means I have a cut which means the tree is cut into two portions or two pieces. One portion is this and the other is this. If I want to make a spanning tree from here, it simply means that, I have to either add 1 - 4 or 3 - 4 or 5 - 4 into it, because I have already removed 2 - 4. I might say even, I would like to add 3 - 4 as well. So, I can make a spanning tree from here by adding 2 - 4 or 5 - 4 or 3 - 4 or 1 - 4. The fact that I have added 2 - 4 is because C_{24} was less than or equal to any other thing that I could have added to make this. What are the other things I could have added? It is either 1 - 4 or 3 - 4 or 5 - 4. Let us go back. 1 - 4 does not exist. Therefore, $C1_4$ is infinity, so C_{24}, 8 is less than or equal to infinity; 3 - 4 was 8, so 8 is less than or equal to 8; 5 - 4 or 4 - 5 is 8, so 8 is less than or equal to 8. We observe that this 8 that we added actually satisfies the cut optimality condition.

For example, if we remove this, then you realise that the tree is not split into two trees. To make it a spanning tree, I have to add either 1 - 2 or 2 - 3 or 2 - 4; 2 - 4 is already there. So, we could either add 1 - 2 or 2 - 3 or 1 - 4 or 3 - 4 or 1-5 or 3 - 5. If you take this so 1 - 2 is 5, 3 - 2 is 7, which is more than 5, 1 - 4 does not exist, 1 - 5 does not exist, 3 - 4 is 8, 3 - 5 is 6 and all of them are more than this 5 or this 5 is less than or equal to all of them.

(Refer Slide Time: 39:16)

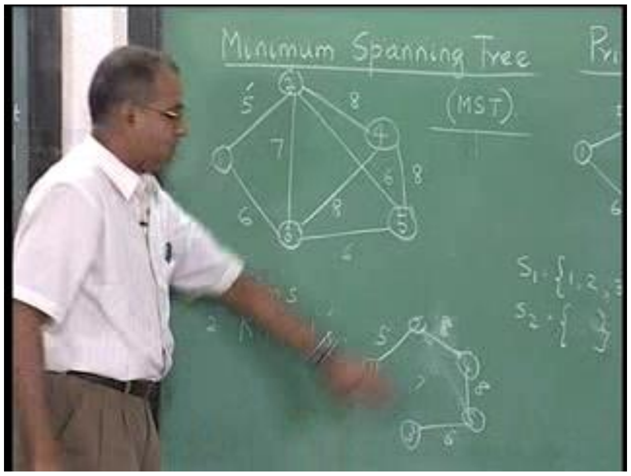


We realise that the optimal or a minimum spanning tree would satisfy the cut optimality condition. For example, if we had taken a spanning tree like this 1, 2, 4, 5 and 3; and, with 5, 6 and 8, we realise that if we do that and then if we remove this 8, it is possible to add 2-5, which is 6, which is less. This spanning tree is not the minimum spanning tree; whereas, if we take this spanning tree we realise that this satisfy the cut optimality condition. Therefore, it is a minimum spanning tree.

We look at another one which is called the path optimality condition. What does it say? It says, for every non-tree arc, k, l: C_{ij} is less than or equal to C_{kl}, for every i-j belongs to T star. What does this mean?

Let us try and explain the path optimality condition using either this or this.~~ Let us go back to this and try and explain~~. Let us consider a non-tree arc which is, say, 2 to 3. So, 2 to 3 is a non-tree arc with 7. For every non tree arc kl, which is 7, every tree arc that lies in the path between 2 and 3 on the spanning tree which has 5 and 6, both of them are less than equal to 7. You take another non-tree arc, which is 3 and 5 that is 8, but you see the other path is 3 to 1, 1 to 2, 2 to 5, all of them are less than or equal to 8. The same thing can be shown with the Kruskal's algorithm as well.

(Refer Slide Time: 41:51)



On the other hand, if we do this, there is a non tree arc which is 7. There is a path from 2 to 3 which has 8, 8 and 6; this 7 can replace one of these 8s. This does not satisfy the path optimality condition and therefore, it is not a minimum spanning tree. We ~~realise~~ realize that both the Prim's and the Kruskal's solutions satisfy the cut optimality conditions and the path optimality conditions and therefore are optimum.

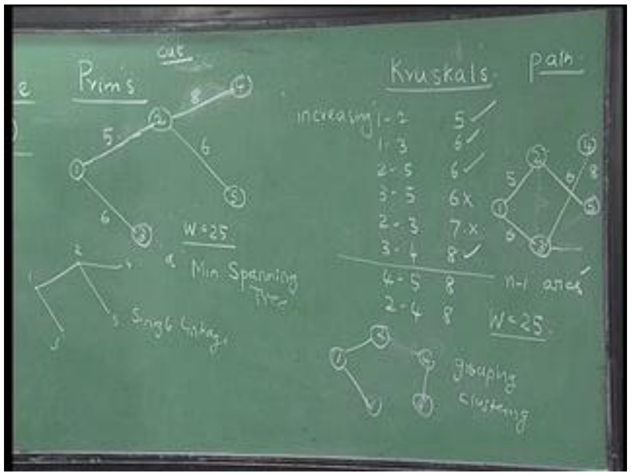
With a little bit of careful observation we may be able to show that the Prim's algorithm is actually a direct implementation of the cut optimality condition and the Kruskal's is a direct implementation of the path optimality condition. Thus, every time we added something it is like the same example that we said, when we added this 2 and 4, we pulled it out. If we pulled it out at some point we have to realize to put 2-4 or 1-4 or 3-4 or 5-4. This is what we did and we actually took the minimum which is 2-4. This is a direct application of the cut optimality condition.

Similarly, in the Kruskal's whenever we did that, we did not add this 7 here, because we had already added 5 and 6, which are individually less than the 7. So, the basic idea of sorting it in the increasing order and then applying the Kruskal's algorithm is a direct application of the path optimality condition. Both these very familiar algorithms are actually direct applications of the cut optimality condition and path optimality condition, respectively. There are still a couple of observations about the minimum spanning tree before we move to the next problem.

The observations are like this. Every time, we apply the Prim's, we always keep building trees. For example, you start with 1 and 2 and then we added 3. There was a tree and then we added 5, which is again a tree and then we added 4 which is again a tree. We keep adding arcs to existing trees such that the tree property is maintained throughout. Prim's algorithm is like a single linkage algorithm. At every stage, we add a link into the tree to make the tree slightly bigger.

The Kruskal's, on the other hand, is a slightly different from a computational point of view. In the Kruskal's, we have to first sort it in the increasing order. Secondly, every time you consider an arc, for example, when we consider this arc we added. When we considered this arc, we did not add. So, we have to check that every arc that we consider does not result in a circuit or in a close loop. The other thing that happens in the Kruskal's is that we may get into a situation where half way through the algorithms it may be possible to have something like this.

(Refer Slide Time: 45:14)



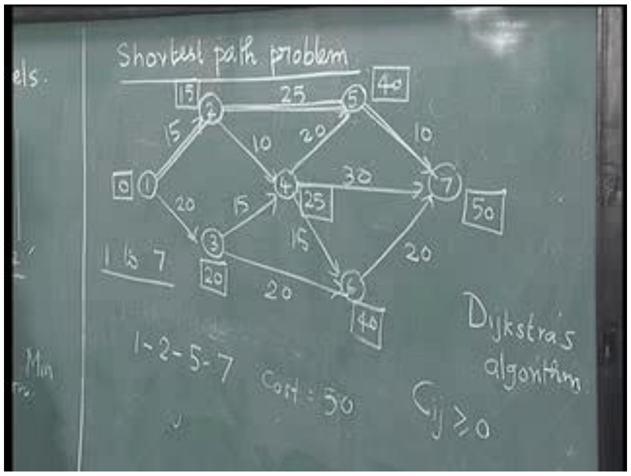
Not in our example though, in some other example it may be possible to have something like this. If we see this carefully, all the vertices are there but we have three arcs and then there are two trees which somewhere later will be again combining into a single spanning tree; half-way through, we may get a solution like this. This kind of a thing is called a spanning forest. All the nodes are here but you still have not got a spanning tree. You have fewer edges into a spanning tree and you have like a collection of trees. So, it is like a forest. This is useful particularly, when we solve the grouping problem or we solve a clustering problem. The Kruskal's way of doing the minimum spanning tree can lead us to a spanning forest which would actually give groups, if we are looking at minimum spanning tree as a way to solve a grouping. That brings us to the end of the discussion of the minimum spanning tree.

We defined what a tree is and then we defined what a spanning tree is and then we started defining what the minimum spanning tree is. Minimum spanning tree is given a network to find out a spanning tree, which means, another network which has the same number of vertices but (n minus 1) edges is a tree and is connected, and called as a spanning tree. Then, to find out that spanning tree which has minimum total weight is called the minimum spanning tree. We also saw two methods, the Prim's algorithm and the Kruskal's algorithm, both to get the minimum spanning tree. We also observed that for the same problem, we could have 2 or 3 different minimum spanning trees, but all of them should have the same total weight with 25.

We also saw the two important conditions which is the cut optimality condition or cut optimality theorem, and the path optimality theorem. We illustrated the theorem on both these solutions and also said that Prim's is a direct implementation of the cut optimality condition and Kruskal's is a direct implementation of the path optimality condition.

We get into the second network problem which is the shortest path problem.

(Refer Slide Time: 48:01)



Let us describe the shortest path problem with an example. Let us consider this network. This network has seven nodes; It has several arcs and we are interested in finding the shortest path from 1to 7. Ordinarily, the nodes will be numbered such that 1is a starting node and the highest node number will be the destination one. Or, there can be situations where explicitly the starting and the destination nodes can be provided.

For example, one might say that, I would like to find the shortest path from 1 to 5. In this case, we are interested in finding the shortest path from 1to 7. In a shortest path problem the network is given. The weights are given. These weights, as I said, can be cost of travel from i to j or time taken to travel from i to j or distance from i to j. Correspondingly, the problem will be the least cost path from 1 to 7 or the least distance path from 1 to 7 or the least time path from 1 to 7.

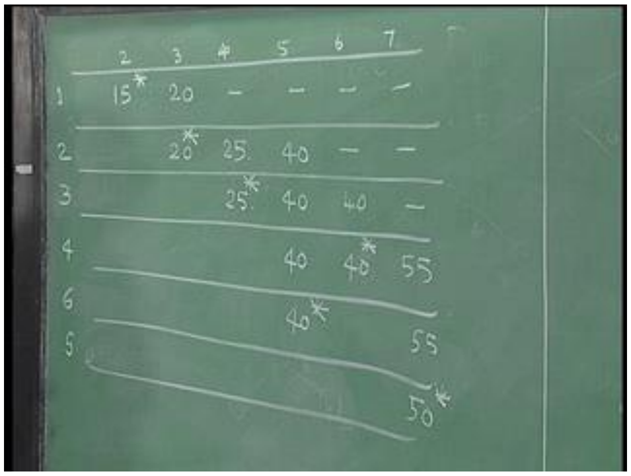
Let us try and first solve this with an algorithm; then, try and explain the goodness or the optimality of the algorithm by starting with a formulation of that problem. First, let me describe the algorithm and then do the formulation later to show that, whatever algorithm we see here is actually optimum. A very simple algorithm would be to say that we start here at time equal to 0. From node 1, we can reach node 2 with time equal to 15 and from node 1, we can reach node 3 with time equal to 20, 20 being more than 15 or 15 being less than 20, we may choose to reach 2 first at time equal to 15. From 1, I can reach 3 at 20; from 2, I can reach 4 at 15 plus 10 is equal to 25 or I can reach 5, 15 plus 25 is equal to 40. Now, come back. From 1, 1 to 3, it is 20. From 2, 2 to 4, it is 25 and 2 to 5 is 40; 25 comes as 15 plus 10 and 40 comes as 15 plus 25. The smallest among these being 20, we say that we reached 3 at time equal to 20. Now, go back and check. From 2, I can do 2 to 4 at 25, 2 to 5 at 40; and, from 3, I can do 3 to 4 at 35, 3 to 6 at 40; 25 being the smallest, I decide at this point to come from 2 to 4 with reaching 4 at time equal to 25. Again, we can repeat this. Now, we say that this is 15 plus 25 is 40, 25 plus 20 is 45, 20 plus 20 is 40 and 25 plus 15 is equal to 40; 40 is the minimum number that we look at. So, we could do 25 plus 15 is equal to 40 or 20 plus 20 is equal to 40 or 15 plus 25 is equal to 40.

Let us say, we do this first to get 40 here. Again we repeat, I can reach 6 as 20 plus 20 is equal to 40, 25 plus 15 i~~si~~s equal to 40; I can reach 7 as 40 plus 10 is equal to 50, 25 plus 30 is equal to 55. Minimum being 40, I do 25 plus 15 is equal to 40 or 20 plus 20 is equal to 40. I have to reach 7 so 40 plus 10 is equal to 50, 25 plus 30 is equal to 55, 40 plus 20 is equal to 60. So minimum is 40 plus 10 is equal to 50. 50 is the shortest distance from 1 to 7 and we also observe that this 50 was reached by 1 to 2, 2 to 5 and 5 to 7. The shortest path is 1 to 2, 2 to 5, 5 to 7 with cost or distance equal to 50.

This is a very rudimentary algorithm for the shortest path problem where the underlying principle is: you start a time equal to 0 and reach the next best destination in terms of minimum cost or minimum time. That is what we did. We moved from 1 to 2 and then we said we reached 3 from 1; then, we said, we reached 4 from 2; then reached 5 from this; then reached 6 and then reached 7 with cost equal to 50. Rudimentary algorithm is a very famous algorithm. It is called the Dijkstra's algorithm and very popular and commonly used algorithm to solve the shortest path problem.

There is an important assumption in the Dijkstra's algorithm which is necessary. The important assumption is that, all these weights that we have, all the C_{ij}s' are greater than or equal 0. When all the C_{ij}s are greater than or equal to 0, it is possible to use the Dijkstra's algorithm to get the solution to this problem. Let us also see how we implement the Dijkstra's algorithm in a tabular form, rather than in the form that we have actually described. Let us do the Dijkstra's algorithm in the tabular form now. We start with seven nodes.

(Refer Slide Time: 55:06)



	2	3	4	5	6	7
1	15*	20	—	—	—	—
2		20*	25	40	—	—
3			25*	40	40	—
4				40	40*	55
6				40*		55
5						50*

We start with 1, 1 to 2 is 15 and 1 to 3 is 20. The rest of them are not connected at all. Take the minimum of them and label it as 15. This is the labeled 1. So 2, from this 2, you could go to 4, 15 plus 10 is equal to 25 where 25 is less than infinity so put 25. 2 is labeled at 15, so you could reach 5; 15plus 25 is equal to 40 where 40 is less than infinity, so put this and the rest of them remain as dash and 1 to 3 remains as 20. If you go back and see that we could label this, as the one which has the unlabeled one with minimum weight so label 3 and get 20.

Come back to this, we are at 3. This is the labeled one. From 3, 3 to 4 is 20 plus 15, 35, where 35 is more than 25 so retain at 25; 3 to 6 is 20 plus 20 is equal to 40 where 40 is less than infinity; 3 is not connected to 5, so the same 40 will remain and this goes. We realise that out of these unlabeled values the one with minimum is 4, which is 25, so label 4 here. Having reached 4, you could do 4 to 5, 25 plus 20 is equal to 45, where 45, is more than 40, so keep it as 40. 4 to 7 is 55 and 4 to 6 is 40. You could label 5, 6 or 7. Let us say we labeled 6 and then put 6 here. From 6, 40 plus 20 is equal to 60 which is more than 55. Retain 55 and retain this 40 because 6 to 5 is not connected. Again, label this 40. Go back to 5 and 5 to 7 is 40 plus 10 is equal to 50 which is less than 55, so update this and label 7. This terminates the tabular version of the Dijkstra’s algorithm which we shall see further in the next class.