Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture – 34

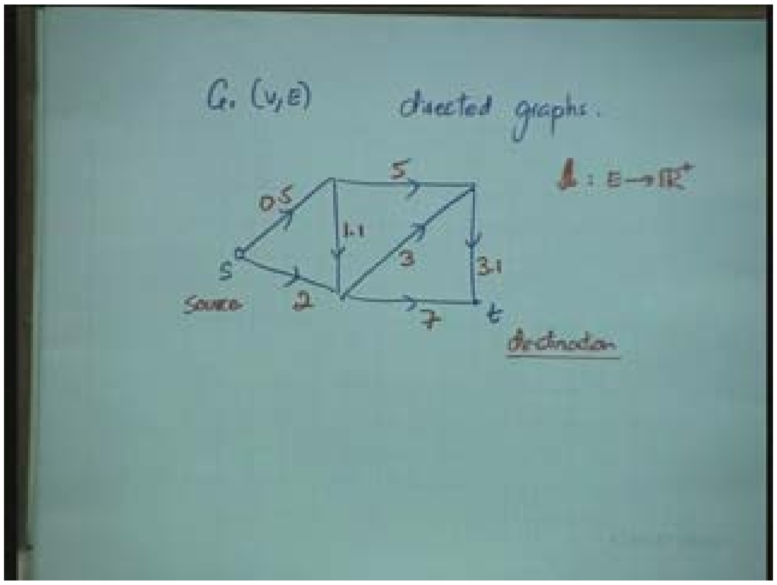
Today we are going to be talking about the single source shortest path. In the last class, we looked at the minimum spanning trees, Prim's algorithm for computing minimum spanning trees. This is a different problem and I will soon tell you what the problem is? What we will, how you will find is that the algorithm is strikingly similar to Prim's algorithm. So, let see what is the problem is and what the algorithm in this problem does.

So, once again you are given a graph and today we can assume that the graph is directed. So, this algorithm or this problem is well defined even for directed graph. I am going to assume that the graph is directed. Let us take a graph, each edge has a length l, is a function from the edges to positive … non negative, we can have the edges of length 0 also and let us give … Let say, this edge has length 2, this is length 3, this is length 3.1, this is length 7, this is length 0.5, this is length 1.1, this is length 5.

These this could be a network of let say, the roads in a city. So, each length so the length corresponds to the distance from this end point to this end point and let see these are 1 way roads. I will, at the end of this class I will tell you other applications of this problem, the ton and tons of that. What is the problem? The problem is to find the shortest path.

Let say, I want to go from this place S, S is also the, also called the source; I want to go from S to t, t stands for termination, no destination. Why should t stand for destination? You will figure out yourself.

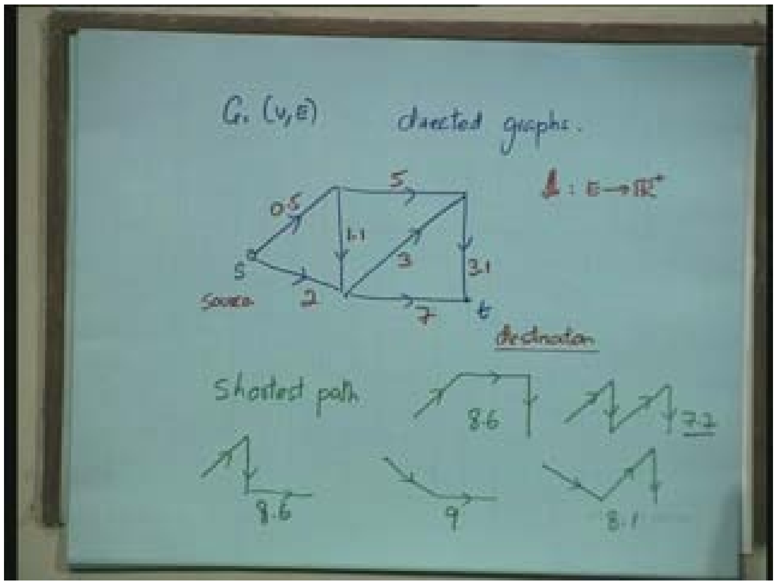
(Refer Slide Time: 4:03)



I want to go from S to t and clearly I would be interested in taking the shortest path. How many different paths are there in this graph between S and t? 4 only? So, there is 1 path I go like this, then I go like this, then I come back here, that is 1 path. The other path is I go like this, I come down, then I go like this and then I come down, this is a second option. The third option is that I go like this, come down, I go straight, this is third option. Fourth option is that I go 2 and then 7 and then 1 other option is this, this and this.

Is there 1 more? 0.53? 3, I cannot come back, I cannot come back on this. This is not a path because that this edge is directed in that manner. It would have been a path if this was an undirected graph but not in this graph. So actually, this graph has only 5 different paths. So, it is a simple problem to compute the length of each path and take the minimum. So, what is the length of this path? 8.6. Length of this? 7.7. What is the length of this? 2,3 8.1. This is 9, this is 8.6. If these were your GPA's, you would have preferred this. But clearly, we are interested in the shortest path. So, we would like to take this path, 7.7 because it is the shortest path, shortest way to get to a destination.

(Refer Slide Time: 5:49)



So, can we always do this thing, as in list out all the paths in the graph and just take whichever is the shortest, compute the length of each part and then compute the shortest? Not feasible? Why is not feasible? Not practical, why is not practical? What is impractical about it? High time complexity, why high time complexity? Why should there be, may be a graph on n vertices never has more than n paths. Then it is simple matter. Just list out all your n paths and just compute the length of … If it has a cycle? Will you ever go along a cycle? You are looking for a shortest path. Why would you want to cycle?

So, you can ignore … not when you are listing out your paths you will try ignore cycles. So, it is not feasible but that is because, the number of paths can be very large. What do you mean by very large? Let us look at an example. Very large means exponentially varying. This is my graph, edges are all directed left to right. You get the idea, this is S and this is t. Sorry, there are more such diamonds, this is t.

If there are n vertices, how many such diamonds are there? How many such diamonds are there if there are n vertices? Or let us do it the other way round. If there are k such diamonds, then there are, how many vertices are there? k diamonds would count as 3 k plus 1. Just do your count correctly. With each diamond I associate this vertex, this vertex and let say, the left end. I cannot associate both because … So, it will be something like, 3 k plus 1 vertex. How many path are there between S and t?

2 to the k, why? because for each diamond, I can either take the top part or the less bottom part. So, there are 2 options in each diamond. So, I can have 1 path which just takes the top part always, 1 path that takes (hindi) 2 to the k different paths between S and t. Everyone follows? (hindi) so, k is roughly n by 3. So, this is 2 to the n by 3 paths roughly.

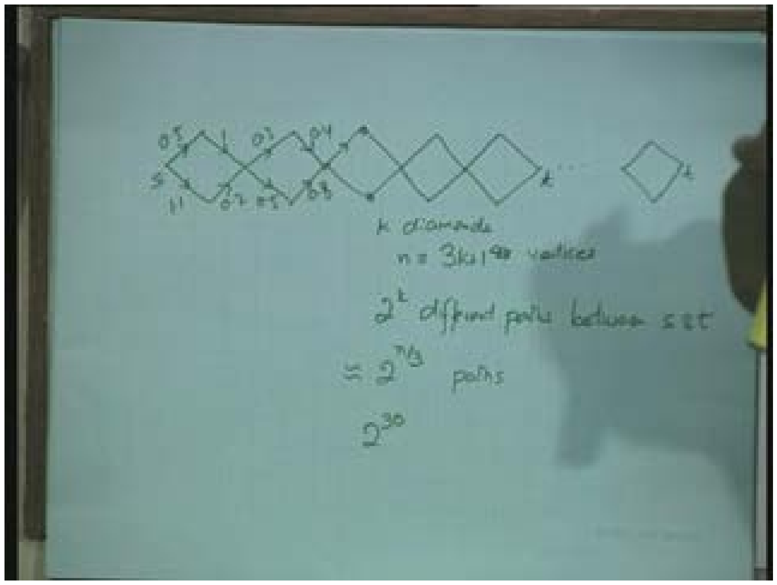
How large is this number (hindi) 2 to the 30 is the number of particles in this universe. (hindi) that is the number of particles, roughly. Of course, no one has counted it, this is an estimate and 30 would be achieved for a graph of size 90, n equals 90 which is not a, which is a very small number, actually. These graphs are typically huge when you are trying to compute shortest path system. So, it does not make any sense to do this caliber, to follow such an approach to compute shortest path.

What should we do? How can we solve this particular example? Can you think of a way of solving this example? Of course, there are lengths on all of these edges which I have not written down but you can imagine; 0.5, 1, 1.1, 0.2, 0.3, 0.4, 0.8, 0.5, I have just taken some arbitrary numbers. What is it that you can do in this example to figure out shortest path between S and t?

Compare, so what you are saying is here, I am going to see, 1.5, if I take this term, thing and 1.3, if I take this one, so I will go bottom and so I would have reached this point now. And then from here, I have an option of going 0.7 up or 1.3 down. So, I will take this one, now I would have reached this point.

So, what you were essential saying is that to reach this destination, I will first have to get to this, then I will have to get to this, then I will have get to this and so on and on. So, first I will have to go from here to here. So, I can choose the, there are 2 options and let I can choose the better option of going from here to here. And then, I have to go from here to here, there is no other possibility in this graph. So, I can choose to better option and so on and on.

(Refer Slide Time: 12:12)

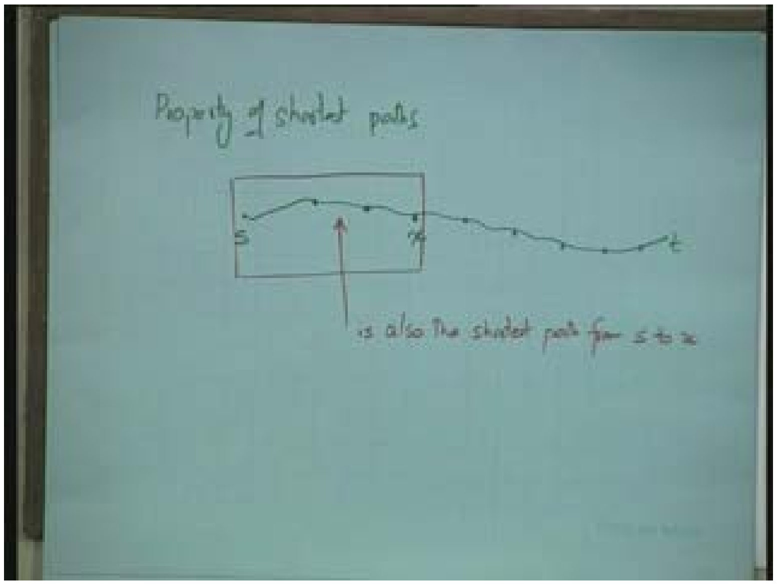


Now, this works in this example, not to always. I have to get to this vertex and I am staring from here. So, but I do not know which is the first vertex I should get to from here. There is no such clear demarcation. As in saying that if I have to go from here to here, I have to clearly go from this vertex to this vertex. We cannot say such a thing in … So, if I have to go from here to here, I cannot say, which is the first vertex have to get through. May be, I have to go through this vertex, may be not: cannot say. Everyone with me? So, that is the part of the problem. So any ideas, how will we do this?

(a to be must contain all shortest path but never part of the part if x is the part of the part from a to b then a to x is the part of the part form the to b then a to x also be a shortest path we can you some shortest heap improving the path)

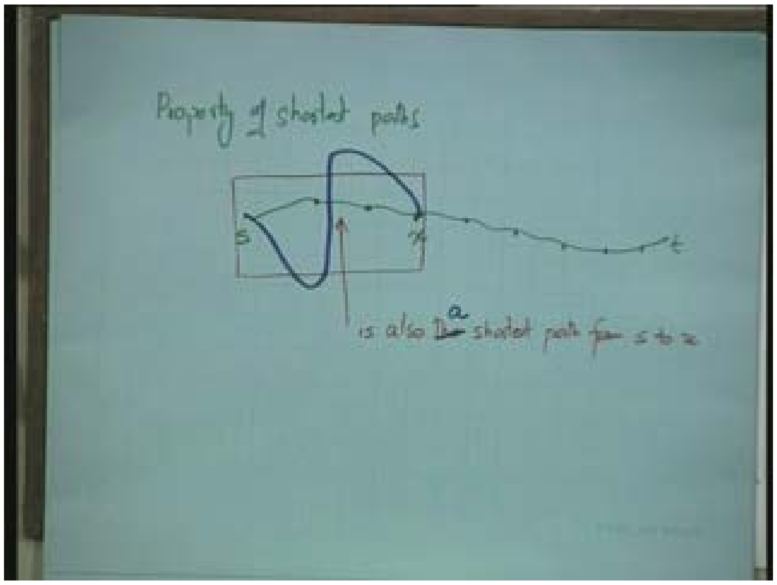
That is an important property of a shortest path. So, this is 1 property we are going to be using. So, what he saying is, suppose this is the shortest path from S to t, this is some path in the graph. There are some intermediate vertices, I have not drawn. Let say, x is one of these vertices. Then the part of this path from S to x, that means this part of the path. What I am referring to is, this part of the path, is the shortest path from S to x. This is also the shortest path from S to x.

(Refer Slide Time: 16:22)


Property of shortest paths
S
x
t
is also the shortest path from s to x

Why? This is, we are saying, this is the shortest path from S to t. Why should it be also the shortest path from S to x? x is any arbitrary vertex. We are trying to arrive at the contradiction, you are saying. Suppose, there were a better path between S and x, (hindi) So, then the claim is that the shortest path between S and t is this blue path from S to x followed by this green path from S to t and there was nothing particular about x. Any vertex any vertex on this path, the shortest path from S to that vertex is this path. (hindi) By the way, the blue path (hindi) so, this is also a shortest path from S to x. So, that is all you are saying, this is also (hindi) this part of the path is also a shortest path from S to x. (hindi)

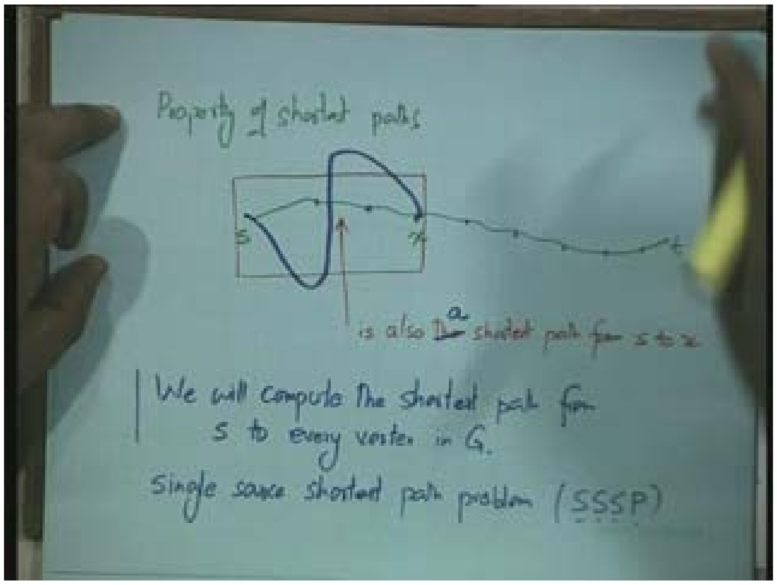
(Refer Slide Time: 17:44)


Property of shortest paths
S
x
t
is also ~~the~~ a shortest path from s to x

So, what does this suggest? This you know, this is giving you the following idea (hindi) we will end up having to compute the shortest path from S to many other vertices, all the vertices on the path. So, what the algorithm we are going to discuss today is actually going to compute the shortest path from S to every vertex in the graph. Not just t, in fact, what we are going to do is, forget t. First step of the algorithm, forget t and just compute the shortest path from S to every vertex of the graph.

Once you computed the shortest path from S to every vertex the graph, you also got the shortest path S to t. (hindi) So, we will compute the shortest path from S to every vertex in G. (hindi) So, this problem is also called as single source shortest path problem. Why? Because, we are starting from 1 particular source and from that source we are computing the shortest path to every other vertex in the graph. Sometimes this would be abbreviated as triple s p. (SSSP) single source shortest path.

(Refer Slide Time: 19:43)


Property of shortest paths

is also a shortest path from s to x

We will compute the shortest path from s to every vertex in G.

Single source shortest path problem (SSSP)

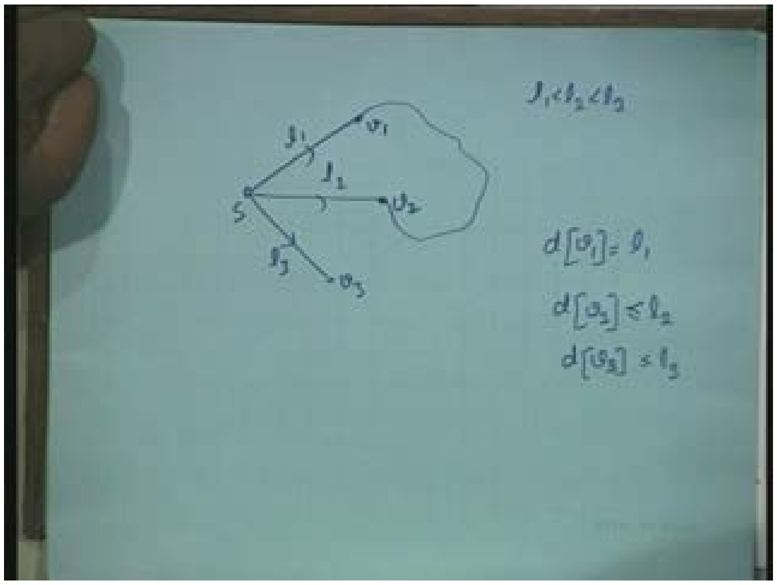
Now, given that we are not just interested in the shortest path from S to t but, the shortest path from S to every vertex in the graph. Let see, what we can say about this example. We are interested in the shortest path from S to every vertex in the graph. (hindi) and without using your head too much, which vertex, you know which vertex or the shortest path from S to which vertex is immediate? Adjacent ones? minimum of the adjacent ones.

We know by looking at this, we know that the shortest path from S to this vertex is 0.5. But this, the shortest path from S to this vertex is not 2. So, we cannot say that I know the shortest path from S to every adjacent vertex. But we know, for a fact that the shortest path from S, so if these are adjacent edges or the 3 adjacent vertices and suppose these lengths were l_1 l_2 l_3. Let us assume l_1 less than l_2 less than l_3, then I know for a fact that shortest path from S to V_1 is l_1. Why? Why do I know this for a fact?

Any other path, since it has to start from S, it has to take some edge out of S, either it takes this edge or this edge or this edge. If it takes this edge then the length of the path will be at least l_2. Why? because edge lengths are all positive. Why negative (hindi) has to be at least l_2 because this part of the length, path is only non negative. It cannot be negative. Similarly, (hindi) so the length of the path has to be at least l_3.

So, what does that mean? The shortest path from S to V_1 is l_1 (hindi) so just by looking at this I can say that, so I am going to use d to denote distance. So, distance of vertex V_1 I know is l_1. What can I say about the distance of vertex V_2? d of V_2, what can I say about that? greater than? greater than l_1? Greater than l_1 and less than or equal to l_2. Let us that put down and similarly distance of V_3 is less than or equal to l_3 (hindi) we do not know the correct distance. But, we have an upper bound on the actual distance. This is an upper bound that this actual distance is only less than this value.

(Refer Slide Time: 23:44)



So, what are you going to do now? We know that actual distance of V_1. Now, how can we extend it? How can we find out the, we know I have to find out the distance from S to every vertex. (hindi)

(initially the directed length we have each vertex have vertices the shortest path have length of the shortest path found nay point if you at vertex found with visited vertex then compare to the are in the length already stored in the vertex to the some of the path) (hindi)

I am just taking an example and soon we will concretize this algorithm. So, do not worry too much about it. Let say, V_1 V_2 V_3 (hindi) l_1 l_2 l_3 edge lengths (hindi) Let me now replace them by numbers because … otherwise, we will keep having lots of ls. So, let me put down numbers, 0.7 and say this is 1.1 and this is 2.3. So, l_1 is the smallest, let us put down more numbers. (hindi)

Let us try and understand what I have done here? Just to separate this out. So, we know this, let me circle it. We know the correct distance of vertex V and the correct distance of vertex V_1 is 0.7. Now, looking at the picture can I say, so this is the only, you know there might be other edges and so on between theses vertices, I have not shown those edges. I have just looked at the edges going out of V_1 now and there are 5 edges, 4 edges going out of V_1.

Now, looking at these edges can I say for certain, what the shortest path to some other vertex? I am not saying to this specific vertex. What is the shortest path, I just want to know the correct shortest path to some 1 vertex. No, you are going into details. Let us understand what the idea is. Let us look at vertex V_2 or let us start by looking at vertex V_3 (hindi) we had seen that (hindi) we had seen only this edge and we had said that the distance of V_3 from S is less than 2.3. That is what we had so far, it less than 2.3. Now, can I say something better? Because, now I see this edge from V_1 to V_3, I know the shortest path from S to V_1, what is it? 0.7. I see that the length of this edge is 0.3. So, 0.7 plus 0.3 means 1. So, I have found another path from S to V_3 of length 1. I can say that the distance of V_3 from S is less than or equal to 1.

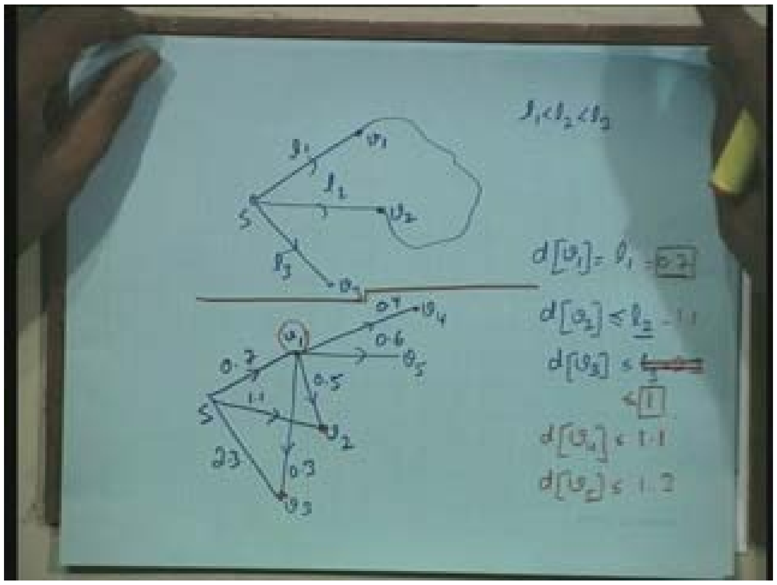
Now, let us look at vertex V_2. 1, we will come to why it is exactly, 1 step at a time. All I will say is V_3 has a, the distance of V_3 from S is less than or equal to 1. That is all I can say at this point, till I bring in some additional argument. So, the distance of V_3 from S is less than equal to 1. That is all I can say. What can I say about the distance of V_2 from S? Well, I have found another path to V_2. That is the path which goes from S to V_1 and then follows the edge V_1, V_2. But this path, any such path is going to have a length of at least 1.3. In particular there is only 1 path at this point but this has a length of 1.2 and since earlier I had a length of 1.1, well, I have not found a better path to V_2.

What can I say about V_4 now? Well, V_4 (hindi) there was no path from S to V_4. Now, I am seeing a path from S to V_4 whose length is; well, this is an edge from V_1 to V_4, so first I get to V_1. I know the shortest path is from S to V_1 that is of length 0.7. So, this is 1.1 and distance to V_5 is 1.3, 0.7 plus 0.6.

So, these are my new upper bounds in the distance. I call them tentative distances. This of course is fixed, final. This will not change. This was the shortest distance, this can never reduce but the other distance is might reduce as the algorithm proceeds. For instance, this distance, this tentative distance which was earlier 2.3 has now reduced to 1.1, 1 sorry.

This did not reduce. This earlier was not reachable at all. Let say, it was infinite earlier. Now, it has 1.1 and this also became, came down from infinite to 1.1. Now, what I am claiming is that let me look at the smallest of these which is 1 and this is the correct distance of V_3 from S. (hindi) At least 3 edges? No, we cannot say such a thing.

(Refer Slide Time: 32:48)



Conversation (hindi) This is what, so let us first say what the algorithm is and we will have to prove its correctness and we will have to look at its implementation and we have 3 lectures do it. So do not, let us not rush ourselves. This is not a straight forward algorithm but let us understand what the algorithm is first. So, once the algorithm in grained then we will see what the, why it is correct. So, what is my algorithm now? So, my algorithm is going to have a set S of vertices (hindi)

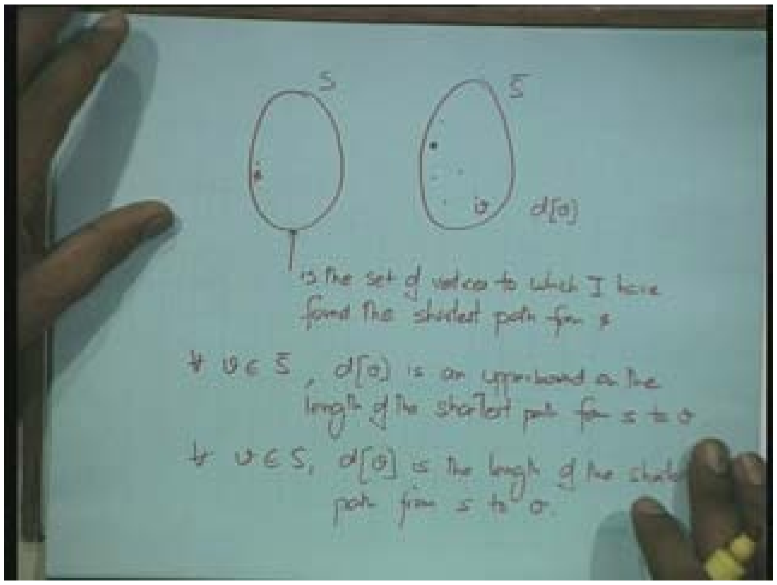
What does this set denote? This set is the set of vertices to which I have found the shortest path from little s. (hindi) so this, what is going to happen in this algorithm is with every step of algorithm, I would have found the shortest path to 1 more vertex. When I say I found the shortest path; I mean these shortest path, not an upper bound or any such things. I have found the correct value of the shortest path from S to that vertex. (hindi)

I would include 1 vertex into the set. At every step I will include 1 vertex into the step. So at any point, this is the set of vertices which I have found the shortest path to and this is the set of vertices for which I have not yet found the shortest path. I just have an upper bound on the value of the shortest path. I know that the shortest path less than this number. But, I do not really know, what is the correct value of shortest path?

So, each of these vertices has an upper bound. This is the vertex V, let d of V, so, let me write down; for all V in S complement, d of V is an upper bound on the length of the shortest path from S to V. You understand what I mean by upper bound? It is a quantity which is only larger than the length of the shortest path. It can never be smaller than the the shortest path and for all V in S, d of V is the length of the shortest path. (hindi)

I am just telling you, what are the various invariance the algorithm is going to … (hindi) These quantities would be upper bound and this could be the actual values. Now, I will tell you how do I figure out, how do I move 1 vertex from here to there? Which is the vertex I move from here to there and what happens when I move that vertex? So first, which is the vertex I will move from here to here?

(Refer Slide Time: 37:27)



is the set of vertices to which I have found the shortest path from s

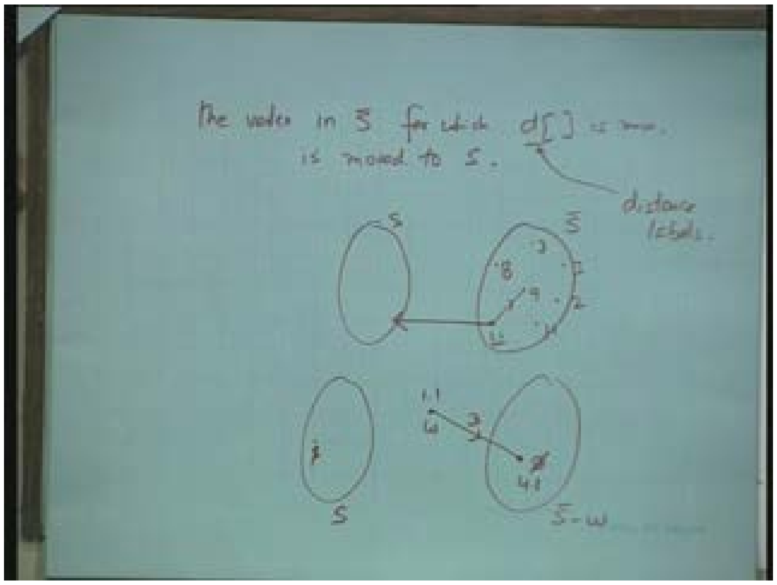
$\forall\ v \in \bar{S}$, $d[v]$ is an upper bound on the length of the shortest path from s to v

$\forall\ v \in S$, $d[v]$ is the length of the short
path from s to v.

So, I find the vertex in S complement for which d is minimum is moved to S (hindi) I have my S complement, I have my S, I have these tentative distance on all of these vertices. (hindi) I will move that vertex here (hindi) this is the vertex which will move from S complement to S (hindi) So understand, what moves from S complements. Now, what happens when this move? When this move is down? How do these numbers change? So, how do we update? That is what we have to determine now. (hindi) Let say, I am looking at this edge, (hindi) from what we have said, the right distance from little s to this vertex W will now become 1.1, will be 1.1.

So, I know that from S, I can reach W using the path of length 1.1 which means that I can reach this vertex by using the path of length 1.1 … 3, 4.1. So, I can change this 9 to 4.1 now and this is my new upper bound or the length of the shortest path from S to this vertex. So, this is how I am going to update these labels. These ds are also called distance labels. (hindi) I have not told you, you have a slide … of why we are doing? What we are doing? But, we have not discussed why it is not correct. We will do the correctness for this. So, do not lose heart. I first want to make sure that you understand what the algorithm is.(hindi)

(Refer Slide Time: 41:34)



Understanding will be complete after we write down the pseudo code. So, let us do that. So this is going to be almost exactly like Prim's algorithm with a very small modification. So, you can now start telling me what I should do? What should I do? What should my initial values of distance labels be? Infinite, distance label recall, we said is an upper bound on the length of the shortest path from S to that vertex. Since, initially I have no clue what those things are, I put every vertex, I give every vertex distance labels of d of V equals infinite and d of S is 0, distance of S is 0 because it is …

Now, what should I do? I should take vertex with the, at any step what do I do? I take the vertex to the smallest distance labels. So, all of these guys should be put into a heap. So, for all V belonging to V, dV equals infinity and H dot insert V comma dV. I have inserted the node V with priority, dV. This guy also I am going to insert. H dot insert S comma 0. So, the minimum (hindi) we have already inserted, you are saying there. Let say decrease, let us call it decrease priority. (hindi)

In my heap the minimum, the top element will be S. I should now remove this element. What should I do? I should remove the minimum. Did I also remove the minimum in the case of Prim's algorithm yesterday? or we just did the find min? That was a mistake. Correct that in your notes. We do not just have to do a find min, we also have to remove it from there. because recall, the only vertices that we are keeping in the heap all the vertices on the S complement side. So, let us look at this and you can then go back and look at your Prim's notes again.

So, what should I do first? I should remove the minimum element. So, H dot delete min and let say, this is vertex V. So, I get the minimum element, the vertex with the minimum priority and that say let say that vertex V. What should I do now? This I got as the minimum vertex, I should look at the edges, I should look at the edges going out of this vertex or the same as saying, let me look at the adjacent vertices.

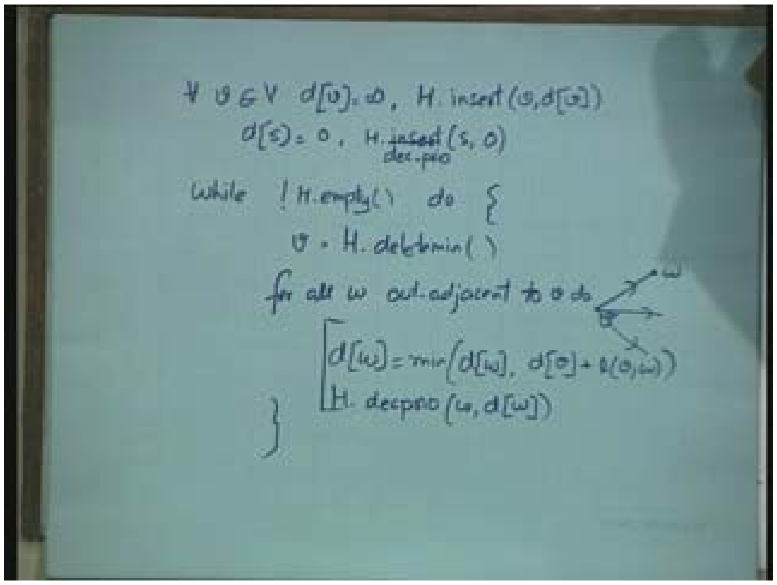
So, for all w, let me just say, out adjacent so it is complete clear what I am saying, out adjacent to V do. So, I am looking at this vertex W which is out adjacent to V. Now, I have to update its label. When should I update the label of W? W already has a label. What is that label? That label is an upper bound on the shortest path. Basically, that label is saying (hindi) now we are seeing a shorter path. But, what she is saying is that will we, we should update W only if it is in S path.

(hindi) even if I were to not to worry about that, the point is I have that I have already found the shortest path to W. If W were in S that means I have already found the shortest path to w. Then there is no way it is getting updated now. Because, if it is getting updated; there is something seriously wrong with what I am saying. Think about this. We do not really need to maintain this information in this particular case.

So, when should I update W or distance label of w? If I find a better path to W and what would be the better path? Through V, so d of W equals minimum, d of W comma, d of V plus l of V W. This is the other path I am saying: to W and if this is shorter then let us put this as the new level. Everyone understands (hindi) because S complement (hindi) the only vertices in the heap are the vertices in S complement (hindi) H dot decrease priority, W comma, this is together (hindi)

This statement would not be executed (hindi) you can do it more carefully this implementation, only if d W decreases, should you call this, but waste of time. You will have to follow a few pointers to get to be heap, make a comparison all of … (hindi) So, we have done. At the end, where are the shortest path? Sorry, what have we found out? We have only found out the length of the shortest path and they are all setting in d.

(Refer Slide Time: 49:54)



```
∀ v ∈ V  d[v] = ∞ ,  H.insert(v, d[v])
     d[s] = 0 ,  H.~~insert~~ dec-pro (s, 0)
While  ! H.empty( )  do  {
       v = H.deletemin( )
     for all w  out-adjacent to v do
          [ d[w] = min(d[w], d[v] + l(v,w))
          [ H.decpro (w, d[w])
     }
```

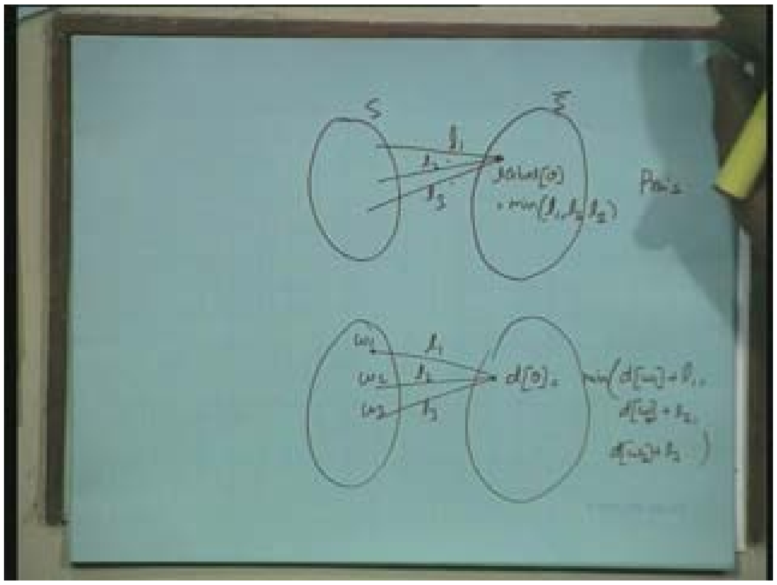
We will come to that, we will come to that (hindi). It is not very different from Prim's. (hindi) We do not have to maintain S and basically (hindi) this is similar to the version of the Prim's, the second version of Prim's that I talked about. (hindi) You recall the second version of Prim's. What was that?

We had maintained S, S complement and for every vertex in S complement, I had 1 label. What was that label saying? Look at all the edges going from this vertex to S, the minimum of these edges would be the label here. Now the label is saying, completely something completely different; in the case of Dijkstra's algorithm. This was Prim's algorithm.

This label, label V equals minimum of (hindi) $l_{1,}$ $l_{2,}$ $l_{3,}$ minimum of these 3 area. Now again, we have a similar thing. We have, each of these vertices has a label. We are calling that label d now. But d is not the minimum of these 3. It is not that, d is not the minimum of $l_{1,}$ $l_{2,}$ l_3 (hindi) d has some other semantic which we will see later. (hindi)

Can someone hazard a guess here? What did you think is d? Minimum of, min of S, the value of d V is min of d W_1 plus l_1 comma d W_2 plus l_2 comma d W_3 plus l_3 (hindi) All these vertices are part of a heap. Priority is this label here and the priority is this d value here (hindi) We will see why this is true. This is we will see when we are doing the correctness of the procedure.

(Refer Slide Time: 54:22)



Who can tell me what the running time is? V square log V? Why V square? I understand, but why n square log n? Second summation degree log n (hindi) we are spending time proportional to the degree of this vertex or the number of times we are calling this loop. Time will be log n times that because of the decrease priority operation. Because, each decrease priority operation takes time at most order log n and (hindi) because up this operation corresponds to removing the vertex from S complement (hindi) each time it takes log n time, so it will be n minus 1 time log n.

This is the now realize, this is the mistake I made in the Prim's algorithm. (hindi) because the semantic is the same. The vertices in S complement are the ones in the heap and so when I am moving a vertex from S complement to S, I have to remove it from the heap. So, you have to do a delete min operation. We just said, find min. If you do not do, it will be an infinite because the same thing will coming as min, min, min, min. (hindi)

So, this total running time, let me put it down. No, not n, but m. Not n, but m why because, we have the total time, the total number of times this operation is being executed is order to m (hindi) With that we will end today's discussion on shortest path, but we will continue this because, we have to prove the correctness of this algorithm. We also have to see, how to find the shortest path because, here we have only computed length of the shortest path from S to the root.