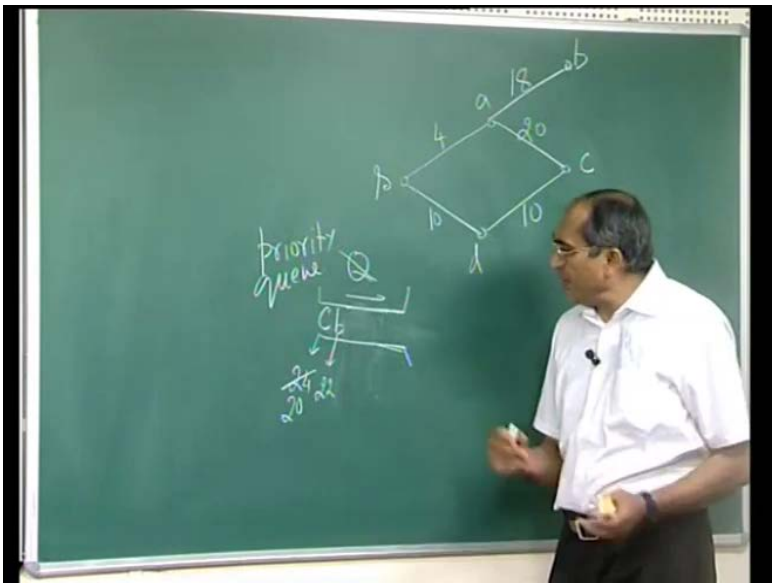
Computer Algorithms - 2
Prof. Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 3
Dijkstra Algo

Hello, so today's agenda is to compute the weighted shortest path tree in a graph in which the edges are weighted. So, let me begin with a small example.

(Refer Slide Time: 00:31)



Consider a graph, now our intension is to use the same breadth first search technique on this graph and see what happens. So, suppose I start with this, so in the beginning I have in the q, this is my q, I put s inside this and then when I take out, I visit a and d and set their states to be current. So, this q takes s out and then puts a and d, the d value of a is 4 and that of d is 10, so let me just write down 4 and 10. Then you take a out, so let us take a out and you are going to place b and c, so let us put b and c. So, the corresponding values will be 18 plus 4 is 22 and 24.

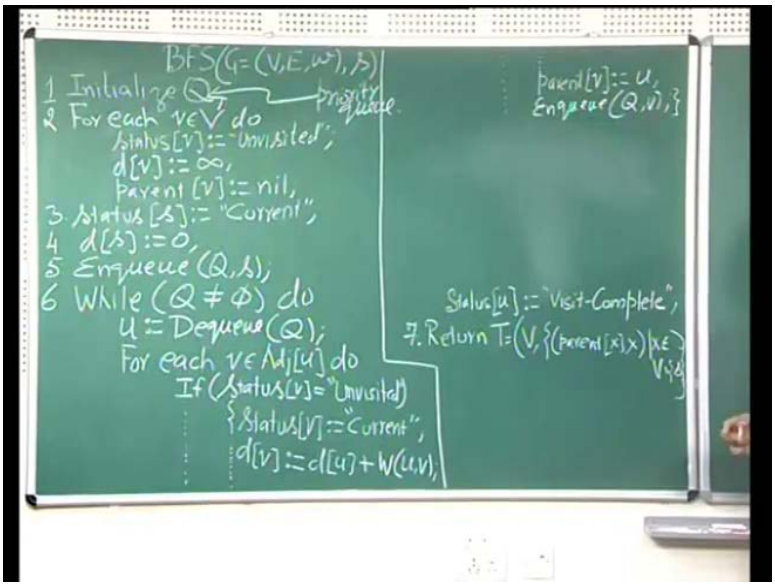
Then we take d, so d is taken off, once d is taken out we see that c is again a neighbor. At that time you notice that the potential d value of this would have been 10 plus 10 which means 20, the d value of c has already been set as 24. So, in the breadth first search algorithm that we have discussed in the last lecture, what would have happened is that

we would have ignored that value a new value of c, but that would have been wrong. What we have found is that in the second visit to c we have found a better or shorter or less weighted path to c.

So, it is no longer possible to ignore that. In other words in each case we find a new value for c which is better we will have to update it, so this should be changed and made 20. Now, in our algorithm we will have to make provision for this change is another thing that is happening. In the previous version of the algorithm, the d values were monotonically non-decreasing, which means that the vertex later on or vertex further away from the head had greater than or equal to d value.

Then the vertex, which was closer to the head, but now you notice that opposite is happening. So, we can no longer work with this kind of a q data structure, now onwards we will have to change this to something called a priority queue. This is a data structure in which we in normal sense insert a new element, but every time when we extract an element we extract the element with the least d value. So, let us do these changes in the algorithm.

(Refer Slide Time: 05:30)

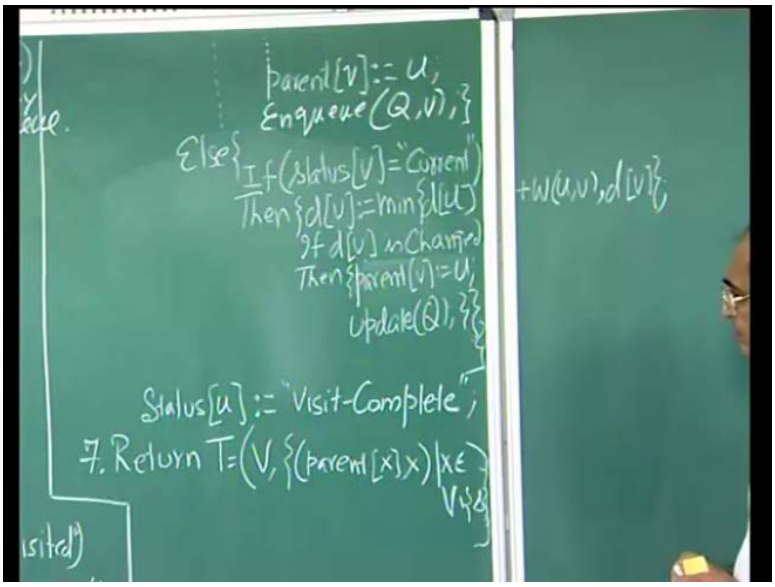


So, here I have the old algorithm as it was stated in the last lecture it is exactly the same, but now I am going to say that this is a priority queue. Now, if you notice here we extract the vertex with the least d value from the q and visit all its neighbors. There are three possibilities, there are three kinds of status values, either the current status value is

unvisited for that vertex, for that neighbor v or it is current or it is possible that it is visit complete status.

So, what we did was when it was unvisited we were normally processing, we were just setting the value to be current, we are setting the d value to be d value of u plus the weight of the edge u v. Earlier, it was 1 now it is whatever the corresponding weight is and then we were setting the parent of v to be u. Then we put that vertex into the queue, but now as we noticed in the example we cannot ignore any other situations.

(Refer Slide Time: 07:25)



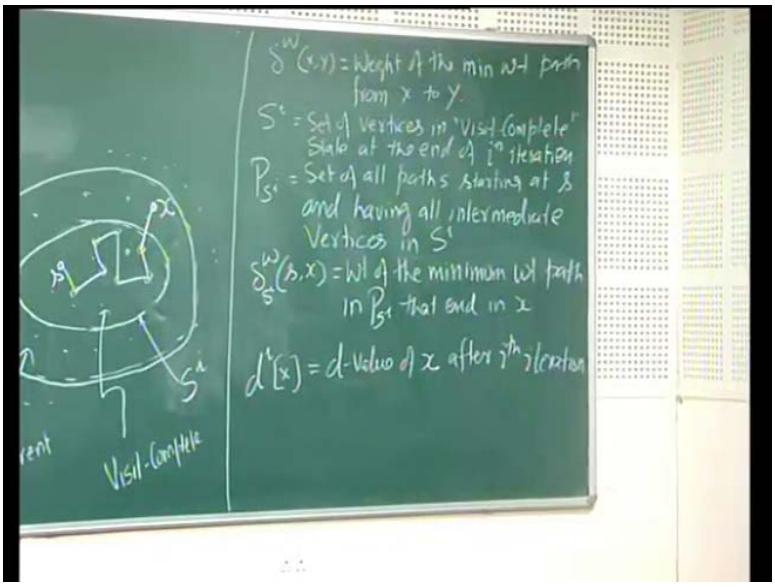
We will do the following what we will say is else, if the status of v is current, this is a situation when the v is in current status. That means it is already in the queue, it has d value assigned which is other than infinity of course, the d value is assigned infinity earlier. So, in this case we will just update the d value, so we will set d of v, what is the value that we will set? We will set the d value to be the smaller of the current value and the value that we would have gotten through u. So, we will set to be min of d u plus w, u, v and the other possible value is the old d value so will just set it to be d v. So, possibly we have improved the value, then we will pick this value otherwise will stay with the old value.

Now, in case we have changed the value, so if d v is changed then we need to do a few book keeping steps, we will have to set the parent of v to be the new vertex u. Because now we have found a better value through u while approaching through u, sorry the d

value, the parent of d value. So, the parent of v is u, we do not have to change the status it is staying current, we do not have to en queue the structure q, vertex v into structure q, but we will have to update the q. Because one of the vertices which was presently in q its value, d value has changed so certain internal changes has to be done, so will just say update q.

So, we have added this piece to accommodate a better value of d, v, what happens when the status of v is visit complete? We do nothing in this case and then we just normally complete the thing. Vertex u, which was being expanded in this iteration gets into visit complete status and we return this structure. Well it seems, it should work, but of course we have to prove the correctness. Now, we are going to prove that indeed this computes the desired weighted shortest path tree with respect to vertex s. So let us start with the simple observation here.

(Refer Slide Time: 12:10)



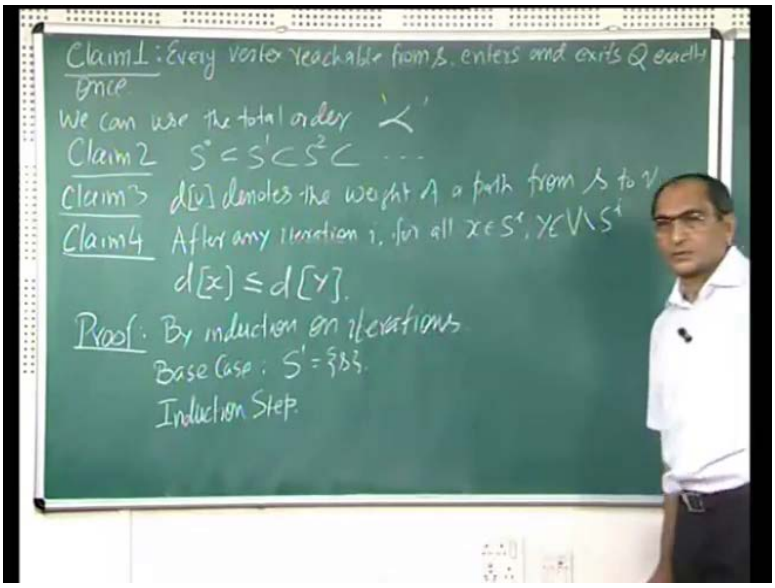
Let us notice that all the vertices, at any given time, at any iteration i, at any the set of vertices, which are in state visit complete are say denoted by these dots inside this. Now, what we notice is that all the vertices in state current must be adjacent to some vertex inside this, because only these vertices were expanded and all the neighbors were visited. So, any vertex which is in current state can have edges like this, so I may have edges like this or I may have edges like this. So, let us put another boundary and think that all the, so this is visit complete and these are current and outside this are unvisited vertices.

So, I will just call these unvisited, maybe I can just draw a solid line, there are these three sets of vertices. So, all the v partition into three, these three sets, edges from here can run within it or can go here, but no edge can go from here to here. Because if there were one then while expanding this I would have visited that vertex, I would have set its status to be current, I would have set its d value to be anything computed through this and so on, so this is the situation, will keep this in mind. Now, first thing I want to do is to define a few notations. So, let us just denote by del w, x comma y as the weight of the minimum weight path from x to y. Note that in case of directed graphs it makes sense to specify this, in case of undirected graph x to y and y to x would be same.

Let us denote by s i the set of vertices in visit complete state at the end of i th iteration, that is this set of vertices at the end of the i th iteration, I will denote them by s i. I will denote by p subscript s i is the set of all paths, this is the set of all paths starting at s and having all intermediate vertices in s i. So, let us take situation so if I happened to have, so s is of course here right from the from the first iteration, so there is a path which looks like this, which starts at s ends at x and all vertices except the last one, which means the first vertex is s, the intermediate vertices are these. This they are all inside s i, which means all, but the last vertex must be from s i, this says s i. Assuming that this is the picture after i th iteration, so such paths are member of this set del w, s i, s x.

Now, if I consider all such paths from which are from s end up in x, x could be here or could be here, you cannot go here of course and all the vertices except the last one are inside s i. Among all the paths, which end up in x what is the shortest path? What is the path with minimum weight? That is this weight, so the weight of all paths in p s i that weight of the sorry, the weight of the minimum weight path in p s i that end in x. So, if you happened to have more than one such paths, then pick the lowest weight and that weight is denoted by this. One more thing I am going to denote is d i, x is the d value of x after i th iteration, is one more point in case there is no path in p s i which ends in x, then this value will be infinity.

(Refer Slide Time: 20:30)



Now, let us make you claims, claim one; once again if you look at the algorithm what you notice is once we pick a vertex u from the queue we expand it we visit all its neighbors, we update the d values if necessary or set the new values. Once that happens you are setting it its status to visit complete, once that is done notice that it is never entered into the queue again. So, this claim again holds now namely that every vertex enters and every vertex i should be correct here, every vertex reachable from a s enters and exits q exactly once.

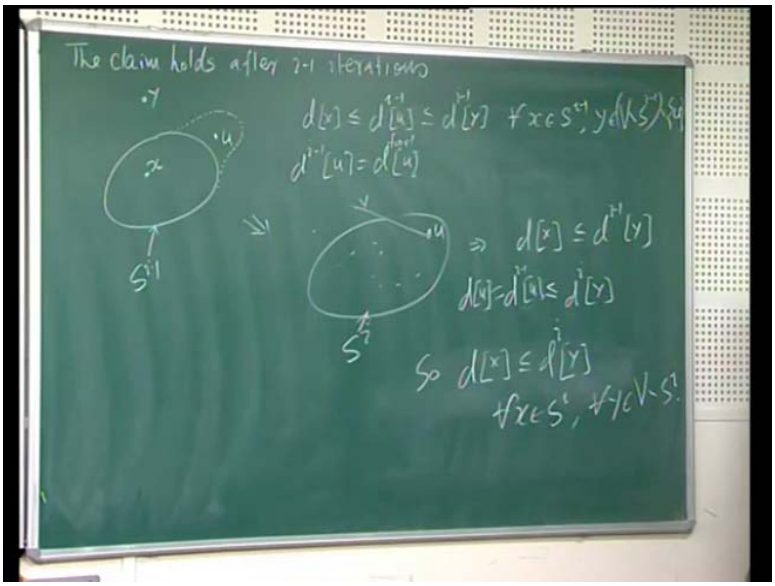
So, I can still use that notation, we can use the total order to indicate which vertex exited the q first, this total order is only over all the vertices which are reachable from the vertex s, those which are not reachable will never figure in this whole process. Another trivial claim is that s 0 is contained in s 1 is contained in s 2, this is empty set this contains only vertex s and then subsequently one vertex is added in each iteration. Another straight forward claim similar to the earlier case is that, d, v denotes the weight of a path from s to v. At any time the value of d v as means set as the value, the d value of its parent, say parent is u, then it is always equal to d, u plus weight of u v.

So, you can always trace it back you go back, this is w, u, v then trace back to its parent and just keep going you reach s. This is the only vertex which does not have a parent, so this path will end up in s and the d value is precisely the sum of these weights, hence it is a path not necessarily the best path, but it is a path whose weight is d v, so that holds.

Now, I would like to make a slightly non-trivial claim 4, so to describe this let me just go back to the picture, we have s i here, this is the picture after i th iteration. For any vertex x inside s i and any vertex outside s i, we claim that the d value of x is always less than equal to the d value of y. So, after any iteration i, for all x in s i and y in v minus s i outside s i, the d of x is less than equal to the d of y. Notice that a earlier stronger claim was made in this regard, what we said was that the q was a normal q and the d values of the things a head were less than equal to d values that was left behind. Now, the structure is not that simple it is a priority q, we cannot say much because by definition what is a head is one, which has got less d value which is inherently true. But what we are claiming is that one something goes out of the q, if it goes inside set s, then every subsequent d value will be greater than equal to that value.

So, how do we prove this? We will try to show this by induction on iterations, so by induction on iterations. Well in the first iteration there is only s inside or let us start with a second iteration, so there is only s at the end of the first iteration, base case is that s 1 is only s, its d value is 0 and the d value of everything else is going to be non negative. So, clearly such a claim is true because there is only one guy in s 1, so x is s, everything else will be automatically 0 or more, so this is trivially true.

(Refer Slide Time: 27:35)



Now, let us try to prove the induction step, so our induction hypothesis is that the claim holds after i minus 1 iterations. So, let us take a picture, this is s i minus 1, let us say the

vertex that we selected the start of i eth iteration is u, so after one iteration this will be our s i. Pick any x, any y here, then from this claim d, x, remember the d values of these never change. So I am going to denote d, x as the final value of x d value of x. This is less than equal to the d i minus value of all these, but by definition this is the smallest d i minus 1 value, this is how it is selected.

So, this is less than equal to d, i minus 1 u which has to be less than equal to d i minus 1 y, for all x in s i minus 1 and y in v minus s, i minus 1 minus u everything outside, s i minus 1 and u. This inequality is because of the choice of u and this inequality is due to the induction hypothesis. Now, that we have this, note that the d value of u because u has been selected in the algorithm never changes. So, d i minus 1 u is also the d value of u, which for clarity I would say final. So, this picture now turns into s I, these are the values including u and all y is here. So, what we have here is that d of x is less than equal to d of i minus 1 y, now s includes u as well, so far so good.

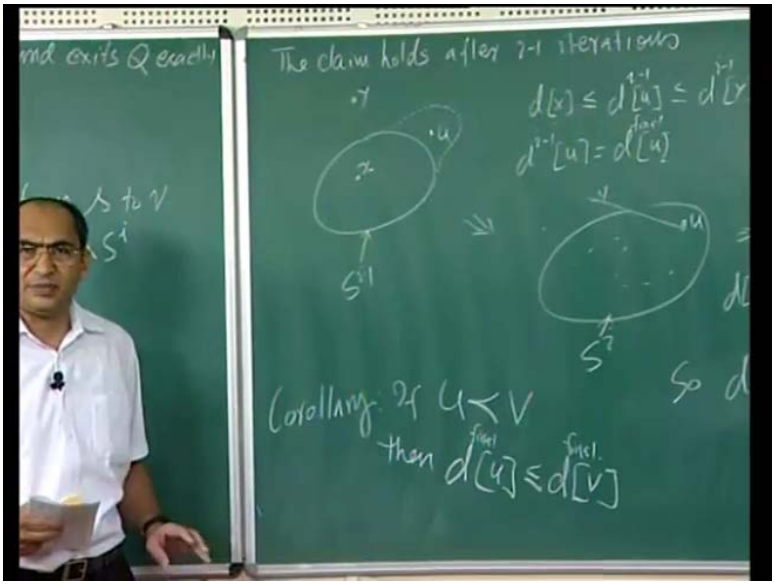
But to prove our claim we need to show that this is also less than equal to the i of y. So, let us pick any y here and look at what happens in this i th iteration. Well the two possibilities either the d value of y does not change in our algorithm, we set its value in case it is a neighbor of u to the minimum of the old d value of y and the value d u plus w u y. In both cases notice that that the d y remains greater than equal to d u, it never decrease, in case y is not a neighbor of u then anyway this do not change. So, what we notice is that this value namely d i of y is still less greater than equal to the d value i minus 1 of u which is same as d of u, the final value.

Once again, let me just quickly run through this point, that all y that are outside s i. Then we split it into two sets, those which are not neighbors of it the d value d i minus 1 is same as the d i. They were always greater than equal to this value, so no change there, this inequality holds. In case a y is neighbor of u, then there are two possibilities, we compare the d i minus 1 of y with d of u or i minus 1 of u which is same plus weight u y. If this is greater than this it do nothing but then this is greater than equal to this value, otherwise we set it to equal this still it is greater than equal to this value.

So, in all situations the new value of y remains greater than equal to this and this value was greater than equal to the d value everybody else here that was the story here. So, still what happens is that, so the d value of every x is less than equal to d value of i eth value

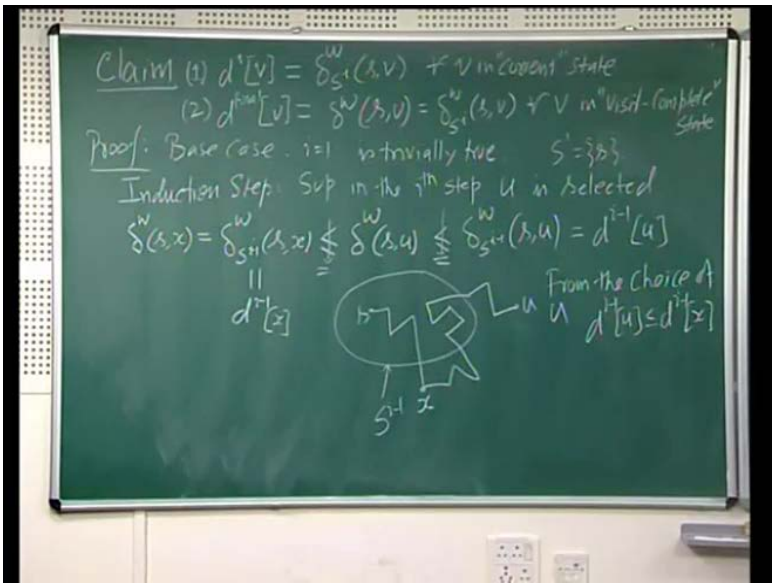
of y, for all x in s i and for all y in v minus s i. Now, one direct consequence of this, what this means is that every time have vertex comes in its d value has to be greater than equal to the d value of everything present in this.

(Refer Slide Time: 35:09)



So, a corollary is that, if u enters s before v. Remember if u exists q it enters s, so if u exists before v or u enters s before v enters. Then the d value of u the final value of this has to be less than equal to v value. When I do not put a superscript, I mean is the final value in this case or maybe I should put that. So, this is essentially a similar kind of result that we had in the previous claim.

(Refer Slide Time: 36:05)



So, the last claim now, the claim that we want to make is that the final d value, what we want to show is that d final of every vertex is its shortest path value, the weight of the shortest path from s to u, we want to prove this. This is our final objective. But we will not be able to directly prove it, we will prove this along with another claim and will try to prove both simultaneously. So, let me state the claim which has two parts that d i of v is del w s i, s to v for all v in current state and 2. The d final of v is del w of s v which is also del w s i of s v for all v in visit complete state.

There is something hidden in this claim let me just look at this, what we are saying is this claim that the final value is same as the shortest weight from s to v, but it is also this. Which means this says that, that there is a path of the shortest weight from s to v, v is already in visit complete all inter visit vertices for such thing must be in, which means the entire path is contained in s i. There is a shortest path and it is also, there is at least one such shortest path is inside s i, this is the full meaning of this claim. Here we are saying something that the current d value that is after i th iteration.

It is the best among the p s i paths, remember the p s i paths were those paths which had all, but the last vertices inside s i. So, we will try to prove these two claims simultaneously. Now, the claim again by induction is valid in the base case, what is happening in the base case I am again taking my base case as case is when i is 1 after the first i th iteration. In that case, you have your s 1 containing only s only vertex in visit

complete state is, this is at the end of first iteration. So, the only vertex qualifies to be an in s 1 is s, so the vertex for which we have to check this is s, d final of s is 0, this is 0, this is 0, this is 0 because the path is empty path which starts here and does not go anywhere just is there. As far these are concerned, the current vertices are those which are directly incident on s, the neighbors of s and the d value is nothing but the corresponding weights.

So, the there cannot be a second path, these are the only possible paths to x 1, x 2 etcetera, which are in current state. So, the delta w s 1 will be the weight of these edges, with these are the only paths for corresponding vertices, there is only one path. So this is the delta value and that is exactly what we have said the d values of the d values of these are set to be precisely d of s which is 0 plus this is weight. So, this is trivially true, remember that s 1 is s, so there is nothing complicated about this claim, for the base case. So, let us now come to the induction step.

Let me assume that in the i eth, suppose in the i th step u is selected, so my claim holds for first i minus 1 i iterations and the i eth iteration just begins. This is s i minus 1, here is u we have just selected, there is s inside this. Select a anyone shortest paths that go from s to u, so it might go like any number of times it might exit the set, it might enter the set and just picking some picture here. So, the weight of this path is what? Delta w s u this is in absolute sense one of the shortest paths without any restrictions, so consider the path and start from s and look at the first vertex that goes out of s i minus 1, remember there is u which is outside this, this is the picture with respect to i minus 1.

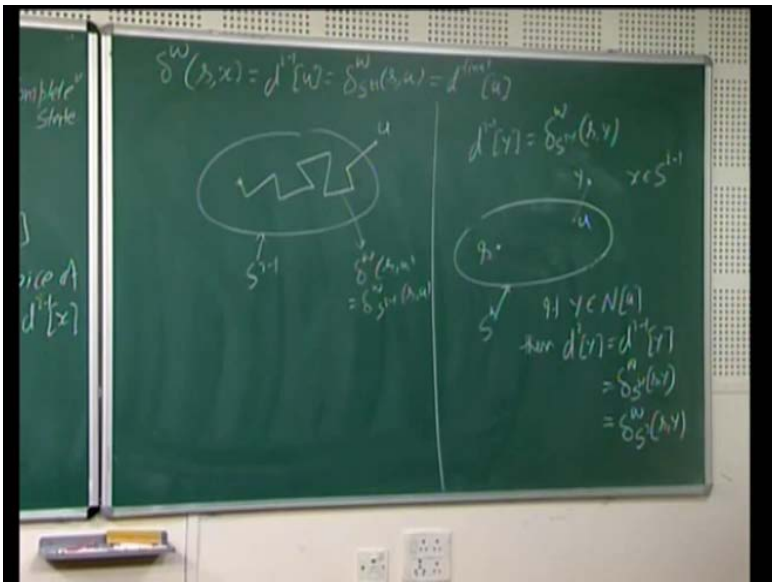
So, certainly there is one vertex which is outside this is path has to go outside s i minus 1, say the first vertex hits is x, let me first assume that x is different from u, then a few inequalities I can now write down. So, the delta w of s x is equal to delta w s i of s x, the reason is if this is a shortest path from s to u, then this has to be the shortest path from s to x, so the weight of this path is absolute minimum weight. But this happens to be one of the p s i paths, because all vertices except this are inside s i minus 1, i minus 1 i should say s i minus 1. So, this qualifies for also this, they are equal, from induction hypothesis this is also d i minus 1 x.

Notice that induction hypothesis applies for i minus 1, so this is equal to this all right. Now, let us look at u, note that the path goes further up to u, so the delta w of s u must be

less than equal to this value, because it goes further, may be the weights are 0 here, so they could be equal, but that is about all. Then let me just draw it somewhere here by definition this has to be s minus 1 of s u. Because this is the best path with respect to a restriction, this is the best path absolute sense and this has to be d i minus 1 u, this comes from induction hypothesis.

Now, notice that there was a certain conditions subject to u was selected, the u was that vertex outside s i minus 1 which had the least d value. So, from the choice from the choice of u, d i minus 1 u has to be less than equal to d i minus 1 x. So, what happens is that this is equal to this, forcing these inequalities to turn into equalities. So, that this thing is now equal this thing is equal, all of them are equal all right they must be equal.

(Refer Slide Time: 47:47)



Now, that we have the equality of these entities what we notice is that del w s x is d i minus 1 u and this is also del w s i s u. What this means is s i minus 1 my mistake, i minus 1, so what we have is that in s i minus 1 there is a path which is completely inside except on the last vertex, this is and this is also the absolutely shortest path one of the absolutely shortest paths. This has weight w s u which is also del s i minus 1 s u, but this is the final value of u, this is also d final of u. So, what we have concluded is that the next enter into inside as, has its d value the final value equal to its optimum distance or the distance actually the minimum shortest path.

So, let us come back to this and look at this claim, the second claim. What we wanted to say is that for all vertices with visit complete state, which means those are which are in s, there optimum value is there d value. What we did is we proved that this is also true for the new vertex that entered into set s. So, induction says therefore, this holds, but this is not complete unless we also prove this claim. So, now we will try to show let everything else, will have everything outside, everything in current state will have this thing true, so let us now prove that claim.

Now, what happens now is that this is your s i with u included in it and pick any vertex y out there. What we know is that d of d i minus 1 y is del w s i minus 1 s comma y, this from the induction hypothesis, this is what happened after i minus 1 iteration. Let me just put y here and I want to show the same thing at the end of the i th iteration. Well what happens when y is a vertex which is not adjacent to u, if it is not, then the path, well let just assume irrespective whether it is this into or not. Let us consider a path now let me directly to that, let us suppose we have a path like this.

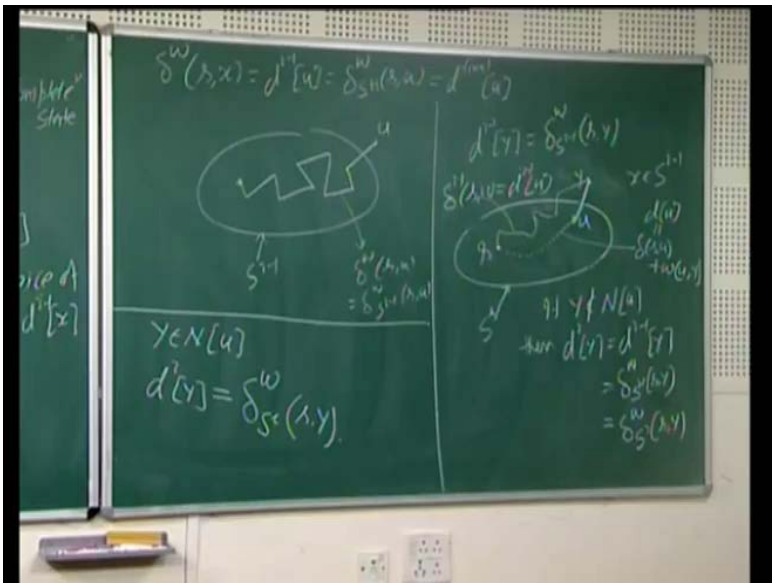
These are the only new paths that we need to worry about, those paths which do not contain u are already in this, they are accounted for. So, the only possibility is that there is p s i path which contains u and it is better, but I am assuming that u is not the penalty met vertex, the second last vertex on this path, say it is something before that. Let us say this is x, well then we know from first claim that the shortest path from s to x, x was present in s i minus 1, x belongs to s i minus 1 because the last vertex was u.

So, there is a shortest path from s to x contained in s i minus 1, so I can take that path, so let me pick that path, let us say that path, this is that path. This is something which does not use u, so the weight of this path being optimum will be same as that weight or less than equal to this weight. So, this pink path followed by this edge must be the, must have the same weight as this or less. So, I have an alternative path from s to y which does not even use u, so I do not need to even consider such a path its weight is not going to be better. So, I need to only worry about those paths which contain u and u is the second last vertex.

So, let me just consider that, here is this s I, this is s, this is y, if y is not incident on u, if y is not in the neighborhood of u, this denotes all the neighbors of u, then there is no such path. Then d i of y is same as d i minus 1 of y, which is same as del, what? Del w s i

minus 1 of s comma y, but since no better path is found including u, then this is also yes, y not in oh sorry, not in my mistake y when it is not a neighbor, this was the case when it was a neighbor, there was no change. Thank you, so in that case the claim that d i y is del w s I, s y.

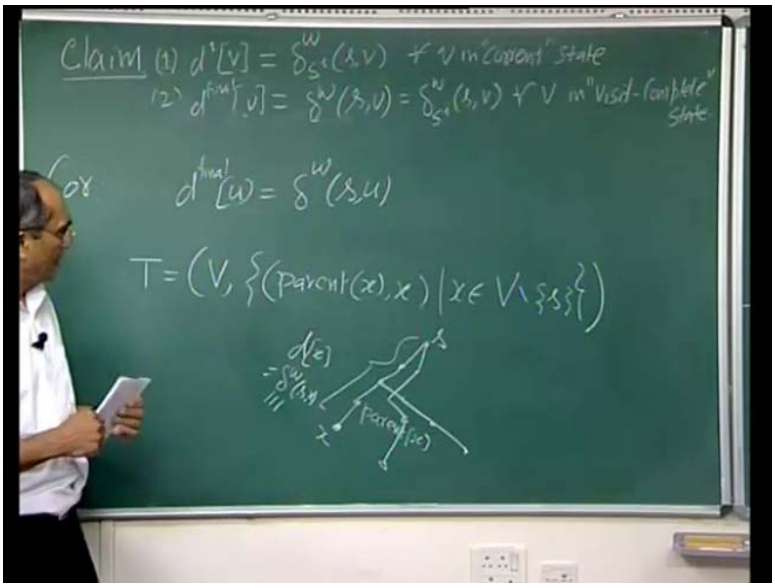
(Refer Slide Time: 55:31)



Second case, when next is when y is in n u, so there is an edge from this, so there are two types of paths either they do not use u, p s i paths or there is a path which goes like this and goes through this these are the two paths. Well, the weight of this path is clearly del s u plus weight of u y, and this thing is equal to d of u. So it is a d u plus w, u y and the weight of this path, the best path is same as d i would say delta i minus 1 s u and which is same as d i minus 1 u. So, the best path of this kind and this weight best of this kind is d u plus w u y.

In our algorithm all we do is compare these two values and pick the best, so whichever is smaller that will be the new value of d y. So, in that case also we notice that d i y is now the delta w s i, s y this is the best path of this kind, so we have established the first claim as well. Now, that we have proven the both statements hold for i th iteration the induction argument is complete.

(Refer Slide Time: 57:49)



So, what we showed is that for every visit complete vertex the d value is its optimum value that is d final for every vertex, this is a corollary. Let d final value of every vertex is its best value in the graph, because finally when the algorithm is over every vertex ends up in s, that value is the d final value, so this is always true. Finally, the graph that we output the graph we output is parent of x, x all these staples x in v minus s is indeed the weighted shortest path tree. The reason is that for any vertex x if we trace it back parent of x and so on, we reach s, these edges are part of this graph.

This is the set of edges, set of vertices are all that I point out that those which are reachable only will figure in this otherwise they are isolated. So, this path is a path in t and the weight of this path is precisely d x and that is same as delta w s x, which is same as. Now, we can say that this is the optimum, this is the optimum distance in the graph and this is also the optimum distance in t, in this sub graph and for every vertex we will basically will find may be ends here, some and then it goes up some vertex and so on.

So, what happens is that this tree, this will be a tree because there are only n minus 1vertices, no edges if everything is connected. So, if you ignore those unreachable vertices say there are n reachable vertices there are n minus 1 such edges, each has at least one edge incident on it, so it is a there is no isolated among them, so this is a tree and the path lengths in the tree are equal to their optimum distance in the graph. That is our definition of the weighted shortest path tree for s, so this is the end of the lecture.

Thank you.