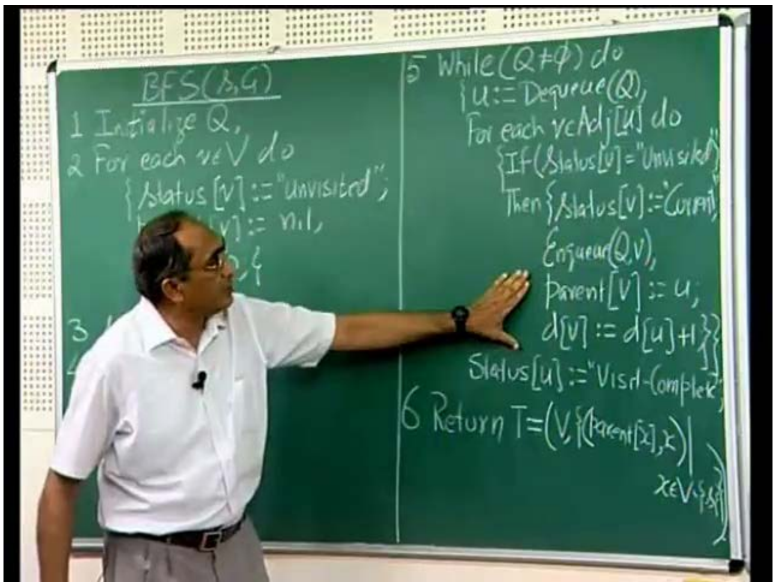
Computer Algorithms - 2
Prof: Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 2
Breadth First Search

Hello. In the last lecture, we described what is breadth first search; and based on that we developed an algorithm to compute the shortest path from a given vertex x to each vertex of the graph. The algorithm is reproduced here.

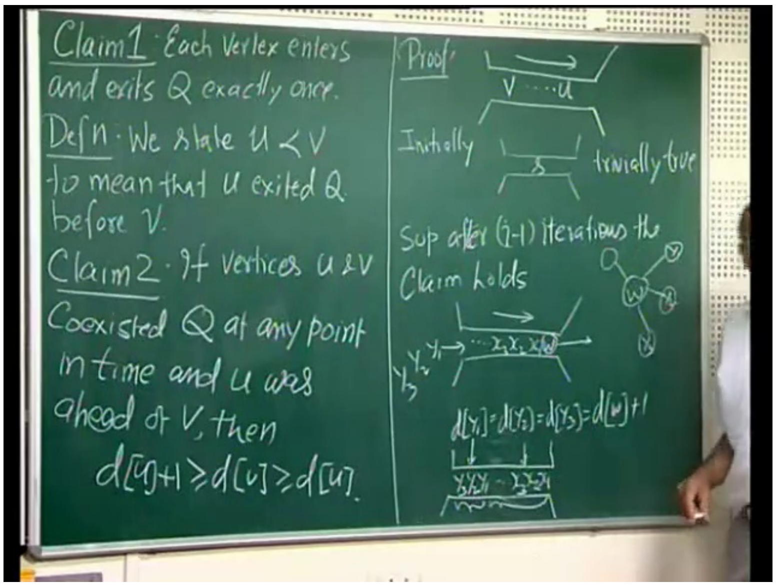
(Refer Slide Time: 00:36)



```
BFS(s,G)
1 Initialize Q,
2 For each v∈V do
      {Status[v] := "unvisited";
       [illegible][v] := nil,
       [illegible]{
3 [illegible]
4 [illegible]
5 While(Q≠φ) do
      {u := Dequeue(Q),
       For each v∈Adj[u] do
           {If(Status[v]="Unvisited")
            Then {Status[v] := "Current",
                  Enqueue(Q,v),
                  Parent[v] := u,
                  d[v] := d[u]+1}}
       Status[u] := "Visit-Complete";
6 Return T=(V,{(Parent[x],x)|
                x∈V\{s}})
```

The two important features of this is every vertex V for which we assigned the parent and the d Value is 1, which is unvisited. At that time, we reassigned the status to be visit to be current. So, every vertex has only one assignment to the d Value and to their parent value. We also stated one claim and proved it that every vertex enters and exits Q precisely once. Now, we will go on and prove the correctness of this algorithm, that is we will establish that indeed it computes the shortest path a tree for vertex s. So, let us define a notation.

(Refer Slide Time: 01:43)



Claim 1: Each vertex enters and exits Q exactly once.

Defn: We state $u \prec v$ to mean that u exited Q before v.

Claim 2: If vertices u & v Coexisted Q at any point in time and u was ahead of v, then $d[u]+1 \geq d[v] \geq d[u]$.

Proof:

Initially trivially true

Sup after (i-1) iterations the Claim holds

$d[y_1]=d[y_2]=d[y_3]=d[w]+1$

v ... u
s
y_1
y_2
y_3
$\cdots x_3\, x_2\, x_1\, w$
w
$y_3\, y_2\, y_1 \cdots x_3\, x_2\, x_1$

We state that vertex U precedes Vertex V to mean that U exited Q before V. Now, that is because every vertex actually enters and exits and that is exactly once. So, there is a natural order on which these vertices are existing from Q. So, this is a well defined total order on the vertices. Now, we are going to state an important claim that is if vertices U and V coexisted Q and at any point in time U was ahead of V, then d of U plus 1 is greater than equal to d of V which is greater than equal to d of U. So, let me explain what I mean by this claim.

So, let me write down here that this represents the cube where the elements exits from this end; and at some point in time U was somewhere here and V was after it. So, the claim says that the value of the d value of U and d value of V their difference cannot be more than 1. This cannot be less than the d value of this. So, we will prove this claim by induction on the iterations. So, initially let us take the state when we first enter s or V en-queue s into the queue. So, initially we have only one element namely s. There is no other element.

So, the claim is trivially true. There is nothing to compare with. So, let us suppose that it is initially trivially true. So, we will take now an induction step. Suppose, after i minus 1 iterations the claim holds, so I would like to prove that the same holds after ith iteration. So, let us suppose we have this situation at the head of this. This is the direction of Q. At the head of Q, is Vertex W and then we have other vertices.

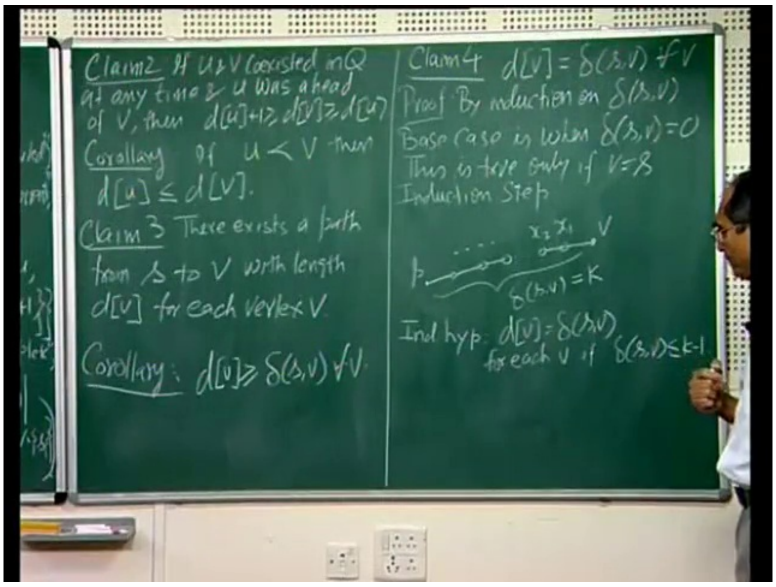
Let us call them x 1, x 2 and x 3 at the start of the iteration. As the algorithm does, it picks the elements at the head of the cube so that variables stated to be U is this vertex. It has value of this vertex W. So, W has been taken out this. It is taken out. Let us come to that piece of the code when you are taking the U out. So, W is this vertex right and now what you do is you look at all the adjacent vertices to the selected vertex W. What you do is you assign the d value of each of these neighbors to be 1 more than the d value of W, provided it is not already visited.

So, it is unvisited. So, when we take out and suppose this is vertex W and you have several neighbors to this, say y 1, y 2 and y 3 in the graph. If these are new unvisited vertices, then we enter them here. So, y 1, y 2 and y three start entering into this. Their d values will be 1 more than the d value of this. So, d of y 1, d of y 2, d of y 3, etcetera are all d of W plus 1. Now, from induction hypotheses, d value of all of these cannot be more than d W plus 1.

So, after this iteration, what you get here is x 1, x 2, x 3, etcetera and then y 1, y 2, y 3. Of course, those vertices of W, which are current, they are already present in this. So, we do not touch them, but the other vertices which are unvisited are entered into the cube. The d values are d W plus 1. Now the d values of all these vertices is between d W and d W plus 1 and the d value of these is exactly d W plus 1. So, our claim is that for any two vertices, if you take any two vertices, say you pick any vertex here and any vertex here, then the d value of this is greater than equal to this and the difference is at most 1.

Now, if both of them are from this collection that claim was already true, because the claim was true in i minus first iteration if the second vertex is among these ys, then we know that the d value of this is either d W or d W plus 1. This is d W plus 1. So, once again the difference is at most 1. This is greater than equal to this. So, if we take this of and both of the vertices are from this side, then their d values are same. So, the claim remains true in all the three situations. Hence, the claim has been established by induction. So, one of the consequence of our result is that the d value of every next vertex cannot be less than the d value of the previous vertex.

(Refer Slide Time: 10:54)



Claim2 If U & V coexisted in Q at any time & U was ahead of V, then $d[U]+1 \geq d[V] \geq d[U]$

Corollary If $U < V$ then $d[U] \leq d[V]$.

Claim 3 There exists a path from s to V with length $d[V]$ for each vertex V.

Corollary: $d[V] \geq \delta(s,V) \; \forall V$.

Claim4 $d[V] = \delta(s,V) \; \forall V$

Proof By induction on $\delta(s,V)$

Base Case is when $\delta(s,V)=0$

This is true only if $V=s$

Induction Step

x_2 x_1 V
s
$\delta(s,V)=K$

Ind hyp: $d[V]=\delta(s,V)$ for each V if $\delta(s,V) \leq k-1$

So, let us see the corollary. This claim is that if U came out of the Q before V, then the d Value of U cannot be greater than the d value of V. The second thing that I would like to establish is that every time we assign a d value to a vertex, which was in the neighborhood of U, we established U as its parent and d value is incremented by 1. So, what happens is that, suppose this is V and this is U, then this has some d value alpha. Then, this has d value alpha plus 1. Now, when U was assigned value alpha. It must have been assigned a parent.

Let us call it x 1. The parents d value must have been alpha minus 1. So, if we go backwards, every vertex except x has a parent. So, it will reach s and its d value is 0. So, what you notice is that, just by walking along the parents of the vertices going towards this, we find that we build a path from s to the vertex V. The d value has to be the length of that path. The reason is that every time we walk along one edge backwards, we reduce the d value and it reaches 0.

So, the d value is the length of that path. So, we can also have claim three. It is, where exists a path from S to V with length d V for each vertex V. This implies a corollary, which is d of V has to be greater than equal to delta S V for all V. Let me remind you that delta S V stands for the length of the shortest path from S to V. Among all paths from S to V the path which has got the minimum length, where length is this. This is the length of some path from S to V.

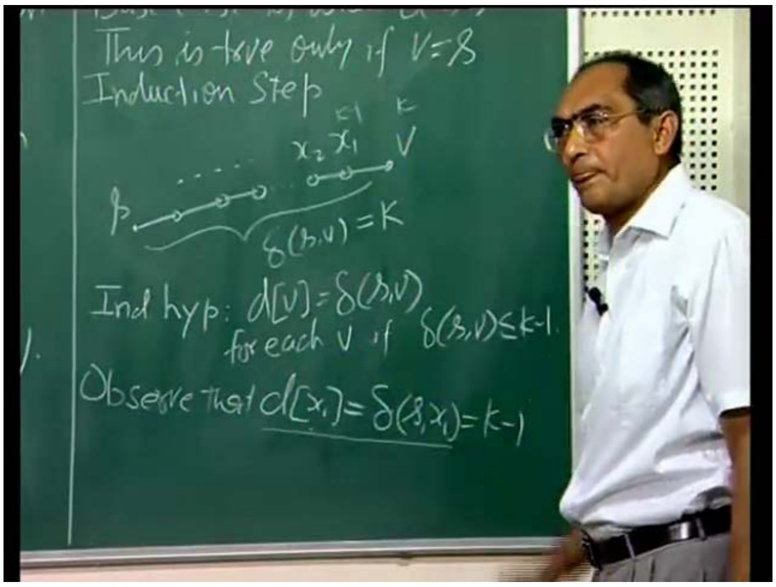
So, this has to be less than or equal to this. Now, we want to prove the main result that not only that this is greater than equal to this, but we will establish that this is equal to this. The d V actually computes the shortest path from S to V. So, let us try do that. The next claim four, is that for all V, how do we prove this? So, we will prove this again by induction, but this time we will do induction on the delta value of that vertex. Each vertex V has a delta S V value unique value.

We will prove this with the increasing value of delta. So, which vertex has got the minimum delta value, exactly the value of S, namely delta S. S is 0. The d values can never be negative because the only thing we do is we assign the d value of some vertex plus 1. So, there is no reason for d value to be negative. So, it is the least now. So, this is by induction on delta S V. So, the base case as we said is, when delta S V is 0. This is true only if V is equal to S. The d value of S is independently set to 0.

Well, it is set to 0 with every vertex. The d value of S is never changed in the entire algorithm. The d value of S is never changed. So, it remains 0. So, indeed for s this claim is true. The d of S is 0 and delta of S is 0. So, base case is correct. Now, we want to prove for other vertices. So, for induction step we will take any vertex V. We will take a shortest path. So, let us say this is a shortest path, from S to V. In general, there could be more than one shortest path.

So, I pick one of them. The length of this path has to be delta S V by definition. Let us say, this is K. So, induction hypothesis is that the claim is true for every vertex V for which delta S V is less than or equal to K minus 1. So, the induction hypothesis says d of V is equal to delta of S V, for each V. If delta of S V is less than or equal to K minus 1, this is the induction hypotheses. Now, I have a vertex, which has delta S V value equal to K. So, we would like to prove that d of V is K. Then, we are done. So, how do I show that?

(Refer Slide Time: 19:47)


This is true only if $V=S$

Induction Step

$x_2 \; x_1 \; V$

$k-1 \quad k$

$\delta(s,V)=K$

Ind hyp: $d[V]=\delta(s,V)$ for each V if $\delta(s,V)\le k-1$

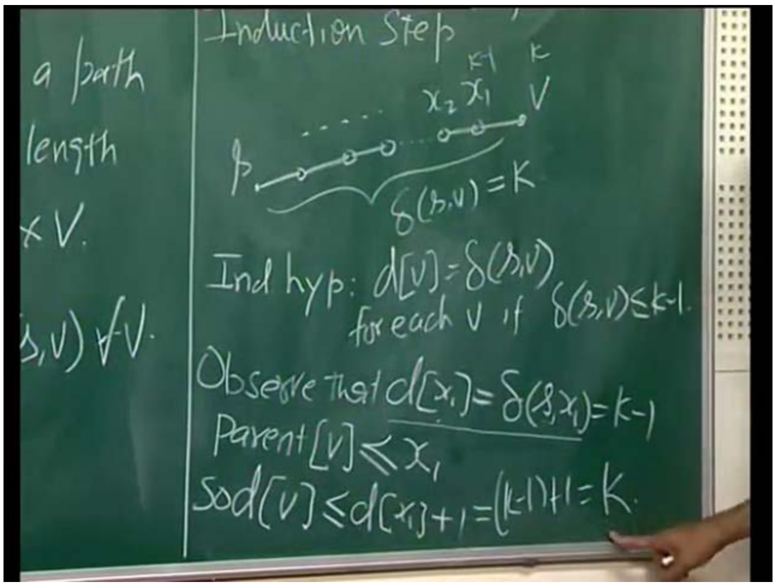
Observe that $\underline{d[x_1]=\delta(s,x_1)}=k-1$

Well, observe that d value of x 1, which has to be delta value of S x 1 is K minus 1. This is the first vertex from this end on the path namely x 1. This is x 1. This is K and this is K minus 1. Why is it K minus? If there was a shorter path from s to x 1 with length less than x 1, then I can replace this path by that shorter path and then I will get a path from S to V of length less than K.

That is not possible. This is the shortest path. So each sub path is the shortest, if delta of delta S x 1 is K minus 1. Then, from induction hypothesis the two are equal now, because the d value is equal to this. So, when we expand the neighbors of x 1, that is vertex U in the step 5.1 was x 1, then we were looking at the neighbors of x 1 here. If it was unvisited, that is among these. We had this V vertex here, which is one of the neighbors. If it was unvisited, then I would have assigned the d V to be d U plus 1, which is K minus 1 plus 1 is K.

Suppose, this was not unvisited, which means it was visited earlier, so the parent of V came before x 1 came. The second situation is while expanding at x 1, we visited V. We found that V was not unvisited. So, V must have been visited through some other vertex. So, the parent of V occurred before x 1. So, we know that the parent of V is either x 1 or it came before x 1.

(Refer Slide Time: 22:46)



a path

length

x V.

s, v) $\forall V$.

Induction Step

$k-1$ k

x_2 x_1 V

$\underbrace{s \;\cdots\; x_2 \; x_1 \; V}_{\delta(s,v) = K}$

Ind hyp: $d[v] = \delta(s,v)$ for each v if $\delta(s,v) \leq k-1$.

Observe that $d[x_1] = \delta(s, x_1) = k-1$

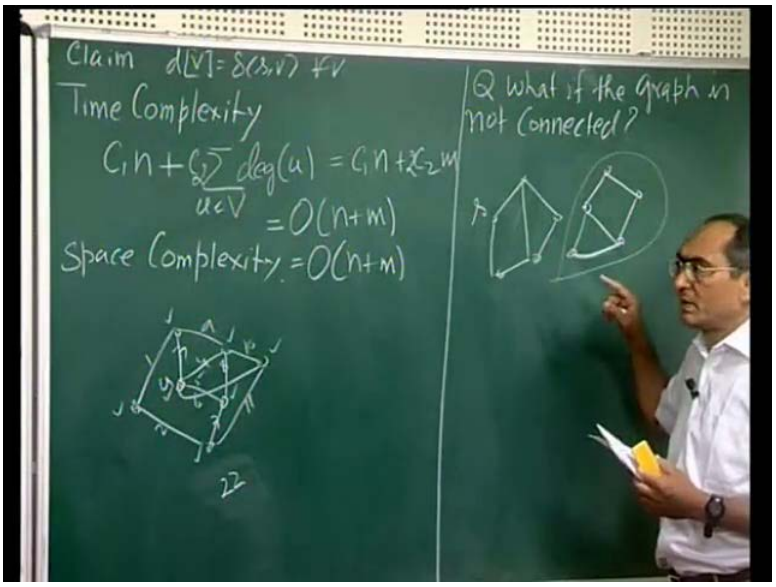
Parent$[V] \leq x_1$

So $d[v] \leq d[x_1] + 1 = (k-1)+1 = K$.

In that case, the d value of V is less than or equal to d value x 1 plus 1. This one we have proven. So, the vertex that comes earlier, its d value is lesser or equal to x 1. So, d of V is less than equal to d x, which is equal to K minus 1 plus 1, that is equal to K. So, we established that d V is less than or equal to K, but take a look at this claim. Here, d V can never be less than delta of S V. Thus, delta of S V is K. So, putting it together we conclude that this is say, d is less than equal to K. This is saying d V is greater than equal to K. Hence, d V is equal to K.

So, we have proven that d V is equal to delta of S V, for V as well. So, we just proved that d V is equal to delta S V, the shortest path length from S to V. This is also the distance of V from S for every vertex. We also proved that there is a path from S to vertex V of length of this. Hence, we can build by just tracing the parent going from vertex to its parent and so on. So, we can from this information, deduce that path as we output here the graph for this vertex. We set those edges, which are from vertex to its parent on for every vertex other than S. So, we have all information in this graph about the shortest path. Now, let us take a look at the time and space complexity right.

(Refer Slide Time: 25:24)


Claim d[v] = δ(s,v) ∀v
Time Complexity
C1 n + C2 Σ deg(u) = C1 n + 2C2 m
u∈V
= O(n+m)
Space Complexity = O(n+m)
Q What if the graph is not Connected?
22

So, in this algorithm has three steps and 1, 3 and 4 are all constant time. This one has got some constant number of operations. It runs over each vertex. So, this is of time order n, where n is the number of vertices. Now, let us take a look at this while loop. Well, this is a constant time, but it will dominate this for loop. In this loop we notice that for each vertex U, we do this once remember that each vertex enters and exits the Q this is the time when this U has exited. From this U, we do this process. So, what is the time complexity of this process?

Well, in this V, we visit each neighbor of V and then check what is the status, whether unvisited or it is otherwise. Depending on that, we do some constant time operation. These are four fixed operations. So, for every neighbor of U we will do once. So, what we have here is that some C 1 n for the step two and then for step five we perform for every vertex U in this thing, degree of U time operation some C 2 time. Here, degree stands for the number of vertices which are adjacent to U. For example, if you have a graph such as this and this is U, then one two and three and four vertices adjacent to U.

Hence, there are four vertices adjacent to u. The degree of U is 4, degree of this vertex is 2, degree of this Vertex is 3, the degree of this Vertex is 3, the degree of this Vertex is 4, and so on, so that what degree starts from this is dominating. These are two dominating terms. Everything else is of course, in constant time. So, well this is really not constant time. This is also order n. This is not even order n. This is the number of edges in this graph, that is the time complexity number of edges that we have, but the number of edges are what for each vertex there is one edges.

So, it is actually n minus 1 such edges. So, entire thing takes order n time which is already accounted for. What is this term? The sum of the degrees of the vertices of a graph, if I just sum this up, I will get 2 and 4, 9 and th3ree, 12 and 3, 15 and 3 edges. So, we have 18 number of edges. If this is 1, 2, 3, 4, 5, 5, 7, 8, 9, 10, and 11, so I have missed something. Let me add up again, 2 plus 3 5 and 4 9, 3 12, 3 15, 3 18 and 4 22.

So, this total sum is 22, which is twice the number of edges in it. In general, you can easily show that the sum of the degrees of the vertices of a graph is 2 times the number of edges. That is simply because, when you count the degree, you count for a given edge, you count A twice, one at this end and the other at this end. Hence, the sum turns out to be 2 times of the number of edges and m denotes the numbers of edges in the graph. So, this time complexity is of the order n plus m. This is the space complexity. Note that, the major data structure we have used here is the edges in the list in which what we do is for every vertex.

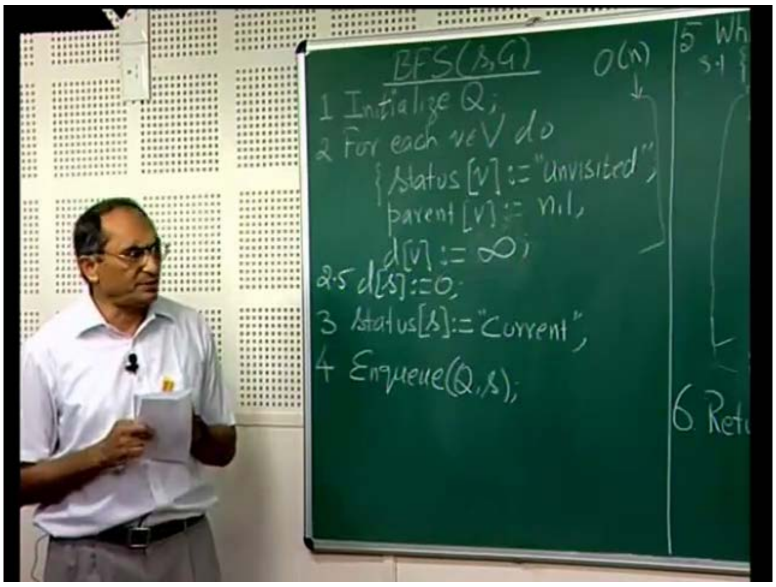
We store the list of all the vertices which are adjacent to U. So, for U we must have stored in a list this vertex, this vertex, this and this vertex. So, the length of the list for U is exactly the degree of U. Hence, the total space used is the sum of the degrees of the vertices which is m 2 times m. So, the data structure called adjacently takes order n time m time. We have certain list like parent and d and all that, which will take order n space. The status also takes order n space Q. The size of Q in the worst case, could be order n, that is the total number of vertices. So, all the set of list that we see here take order n space. So, this is also of order n plus m.

So, this completes the discussion of our algorithm. We have proven the correctness and we found the time complexity and the space complexity now. So, far what we have done is, we have assumed that we have a graph for which we can compute the shortest distance for every vertex from S. The output also gives you one path which gives you such a distance. Now, as we said if you take every vertex and go back to its parent and go on, you end up into S. May be some of them are already there. So, we start building this. What you build is a tree and the length of the paths in the tree from each vertex to S is precisely delta S V for that vertex V.

So, this is indeed the shortest path tree. But, there are certain things we have ignored. So, the question is, why is the graph not connected? So, what do we mean by that? So, what we say is, let suppose the graph looks like this. This is the graph. It turns out that the set of vertices here which have no edge to this set of vertices, means there is no way we can go from S to any of these vertices. Such graphs are known as not connected graphs. If I input such a graph to our algorithm, what will happen? Well, it is clear that after this algorithm is complete, none of these vertices will figure in this. The d values will never change.

The parent value will remain nil. So, the d values will be 0. This is incorrect. The d value of only S can be 0. In fact, what typically one states is that the d value of this should be infinite to indicate that there is no path from S to this. So, we should ideally initialize the d value to infinity rather than 0. So, we should actually set d V to be infinity. That way if any vertex is not accessible from S, then its d value will remain infinity. We will have to make one more addition to this, because d of S must be 0.

(Refer Slide Time: 36:20)



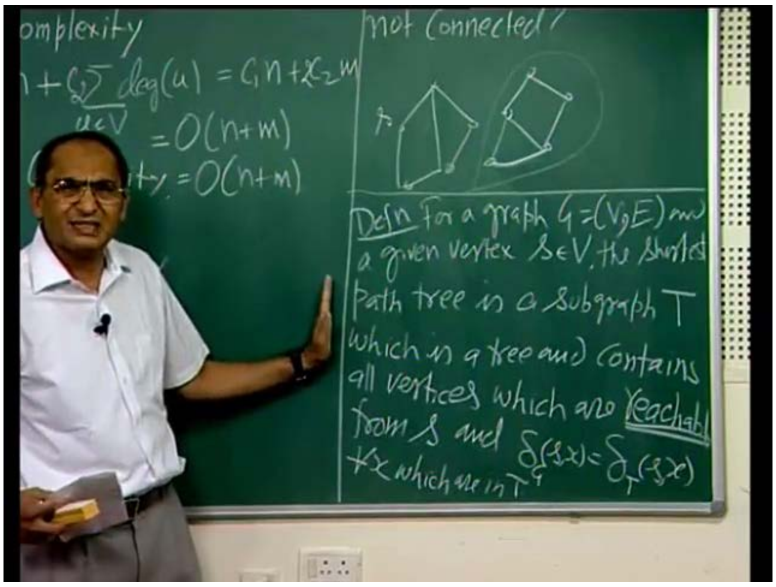
```
BFS(s,G)                          O(n)
1 Initialize Q;
2 For each v∈V do
    { Status[v] := "Unvisited";
      parent[v] := nil;
      d[v] := ∞;
2.5 d[s] := 0;
3 Status[s] := "Current";
4 Enqueue(Q, s);
```

```
5 Wh
  5.1 {

6 Ret
```

So, we should say that d of S is 0. Separately, that is the only change we need to do to handle the graphs which are not connected, but we will have to now change the definition of shortest path tree. Earlier, what we said is that it is a sub graph of the given graph which is a spanning tree. This means that every vertex is present on this tree and the shortest path from S to each vertex in the graph is also equal to the distance from that vertex S. In that tree there is precisely one path between any pair of vertices. So, what we notice here is, in general the graph is not connected. Such a thing is not going to happen there. So, what we will do is will redefine the notion of a shortest path tree as follows.

(Refer Slide Time: 37:44)

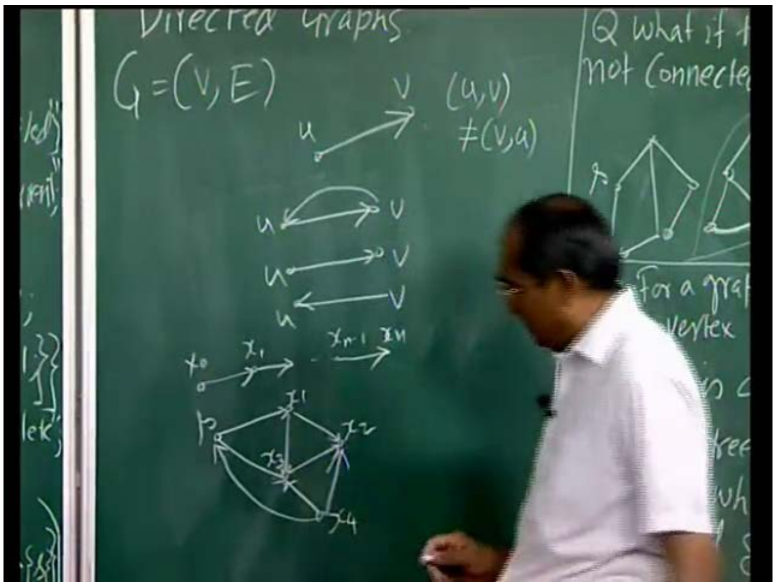


Defn For a graph $G=(V,E)$ and a given vertex $s \in V$, the shortest path tree is a subgraph T which is a tree and contains all vertices which are Reachable from s and $\delta_G(s,x) = \delta_T(s,x)$ $\forall x$ which are in T

So, the definition is that for a graph G equal to V comma E and a given vertex S and V, the shortest path tree is the sub graph T which is a tree and contains all vertices C. They are reachable from S and del G S x is equal to del T S x for all x. So, we defined the notion of reach ability. We say that a vertex is reachable, if there is at least one path reachable from S to that vertex. So, all these vertices are not reachable. All these vertices are reachable. So, our definition is that the shortest path tree is a tree, indeed it is a sub graph of a given graph.

This means its edges are from the edges of the original graph G, but its vertices are only those vertices of G which are reachable from S. So, it will be only restricted to this path. The other part is the same, namely that the distance is short. It means the length of the shortest path. The distance of each vertex from S is in G. It is same as the distance from S to that vertex in T. So, with this definition we can handle the non connected graphs as well. Now, we will define one more type of graph which are called directed graphs.

(Refer Slide Time: 40:56)



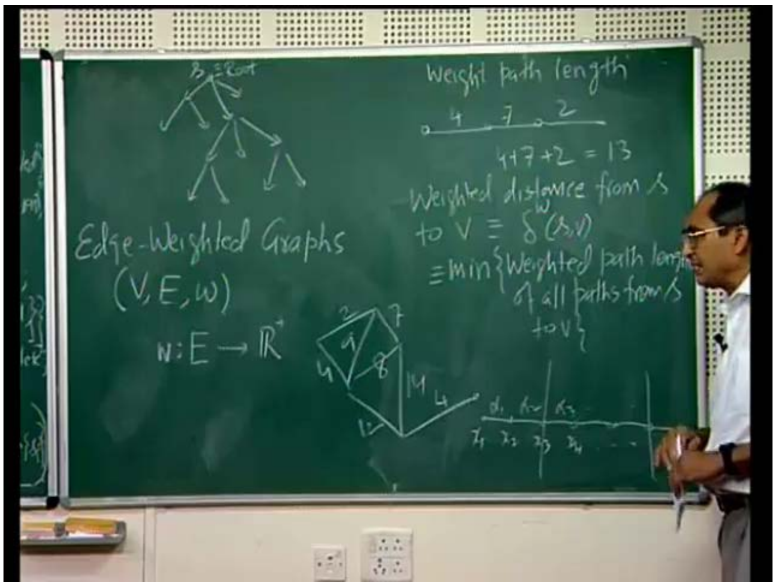
In these graphs, we have the same thing. The set of vertices, set of edges. But, these are directed edges. In other words, for any pair of vertices U V, if there is an edge, we also assign a direction. So, we say the edges U V to indicate that the edge is between U and V. Its direction is U to V. So, in general U V is not equal to V U. Besides, any of these can exists or may not exist. So, it is possible that you have a situation like this or you have just one edge in one direction or the other. All these are possibilities. This is the same vertex U V. In this situation, we modify the notion of path in the following sense.

We say there is a path from x naught, x 1, all the way to x n minus 1 and x n. But, now we also insist that the direction of each of these edges is going away from x 1. So, all of these are directed in one direction from x 1 to x n. So, you may have a path from x naught to x n. That path does not become a path backwards. So, this is not a path from x n to x naught. Now, let us take a simple graph. Now, the direction of the edges I think, it is not that clear. Let me just make it clean. Here, x 3 is this vertex. If you notice the paths from S, they manage to reach x 1.

They can reach x 2. They can also reach x 3, but there is no path from S to x 4. So, the notation of reachability now generalizes. Further, in this case we say that in the directed graphs a vertex V is reachable from x, if there is a path with appropriate direction. Here, the reachable set is S itself. Here, x 1, x 2 and x 3 are reachable, but x four is out. So, in the output, if I run this algorithm exactly as it is, in the output x 4 will not be figured.

The d value of x 4 will remain infinite. So, when we have a directed graph and input that graph into this algorithm, what will come out will be some structure such as this and so on.

(Refer Slide Time: 45:08)



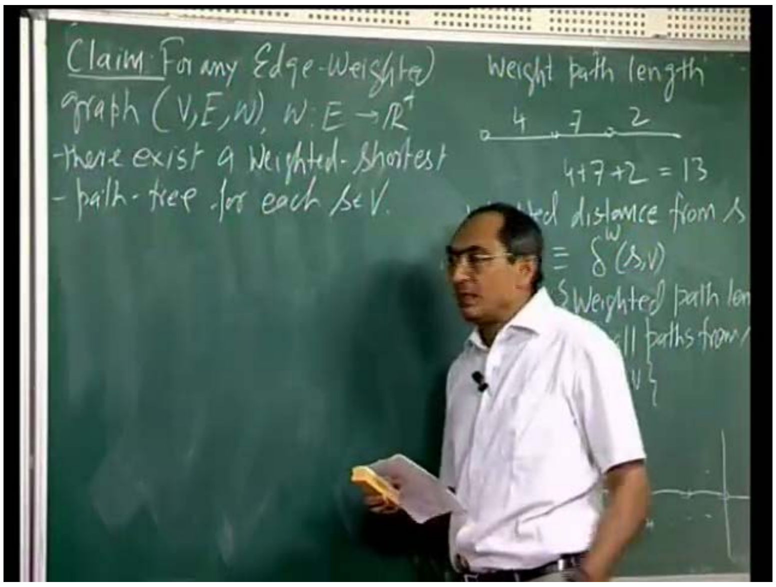
It will describe set of paths from S to every vertex which are shortest paths. So, we can call this a rooted directed tree, rooted to indicate that there is a special vertex in this. We could call this as root of the tree. All the edges are pointing away from the root and all these paths are appropriate directed paths. They are the shortest paths. So, the algorithm functions exactly the way it did in undirected case, but this time it will contain only the vertices which are reachable in undirected sense.

Now, the last thing I want to describe is edge weighted graphs. The edge weighted graphs can be expressed as this triple. This is again a set of vertices and set of edges. Now we are talking in terms of undirected graphs, but we have a third entity which is a function. In this context we are defining these functions as functions which assign a non negative integer, which we call weight to each edge. So, you know we can think of such graphs as two seven eight four nine twelve fourteen four and so on.

So, such a graph is an edge weighted graph and given this, we can again define the notion of the weight of a path. So, something called a weighted path length. So, suppose you happen to have a path four seven two, then weighted path length is nothing but 4 plus 7 plus 2 un weighted path length was 3. So, it is equal to 13. It is the weighted distance from, let us say of vertex x to V which is denoted now by the minimum of the weighted path lengths of all paths from S to V.

So, we have the similar parallel notions. Earlier, we had the shortest path length or the distance which is defined without worrying about the weights. Now, we just take into account the weights and once again note that if you have a path, let us say x 1 x 2 x 3 x 4 and so on, with weights alpha 1 alpha 2 alpha 3 and so on. If this is the minimum weight path from this Vertex, to let us say x n, then from any sub path you pick any two vertices. This has to be the shortest weighted path between these two vertices. If it was not, then I can replace it. If it adds some cycles, you remove those cycles. You can find a shorter path, may be lesser weight path from x 1 to x n. So, there is still there. So, final thing is because of this property, there is still a well defined shortest weighted path tree from every vertex. So, we can say that there exists a shortest weighted path.

(Refer Slide Time: 50:55)


Claim: For any Edge-Weighted graph (V,E,W), $W: E \to R^+$ - there exist a Weighted-Shortest-path-tree for each $s \in V$.

Weight path length

4 7 2

$4+7+2 = 13$

[illegible]ted distance from s

$\equiv \delta^W(s,v)$

{Weighted path len[illegible] [illegible]ll paths from [illegible] v}

So, the claim is without proof, but the proof is very similar. Without proof, we can say that, for any edge weighted graph with W going from E to R plus, this is to indicate that negative numbers are not allowed, there exists a weighted shortest path tree for each S V. So, in the next lecture, we will describe how to compute this. We will do that by modifying this very algorithm suitably.