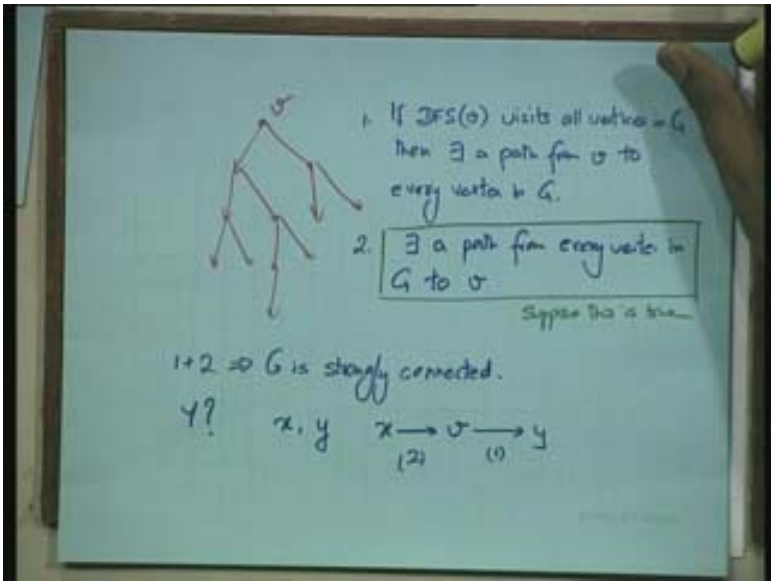
Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
IIT Delhi
Lecture – 30
Applications of DFS in Directed Graphs

Today we are going to look at more applications of depth first search in directed graphs. Recall that in the last class I had mentioned that we will talk about strong connectivity. We will see how to figure out, if a given directed graph is strongly connected or not and I have defined what is strong connectivity means. It essentially means that between every ordered pair of vertices there is a path in the graph so which means I take two vertices lets say u and v, there should be path from u to v and also a path from v to u. Then we would call the graph strongly connected. So in the last class there was a very simple algorithm that was suggested which was that take a vertex, do a DFS from there. Take another vertex, do a DFS from there and so and on which means do a DFS from every vertex in the graph.

If in each one of these DFS's, we include all the vertices of the graph only then is the graph strongly connected. That should be easy to see. Let's see if we can reduce the number of DFS calls that we make, instead of making n DFS's, can we reduce the number of DFS from n to lets say some small number. Let's say I take one vertex v and I do a DFS from here and when I do a DFS from here, I visit all the vertices in the graph. This is let's say the DFS tree I obtain and if these were the only vertices in the graph then I have visited all the vertices in the graph. So what do I know? I know that there is a path from v to every vertex in the graph. If DFS v visits all vertices in graph G then there exists a path from v to every vertex in G.

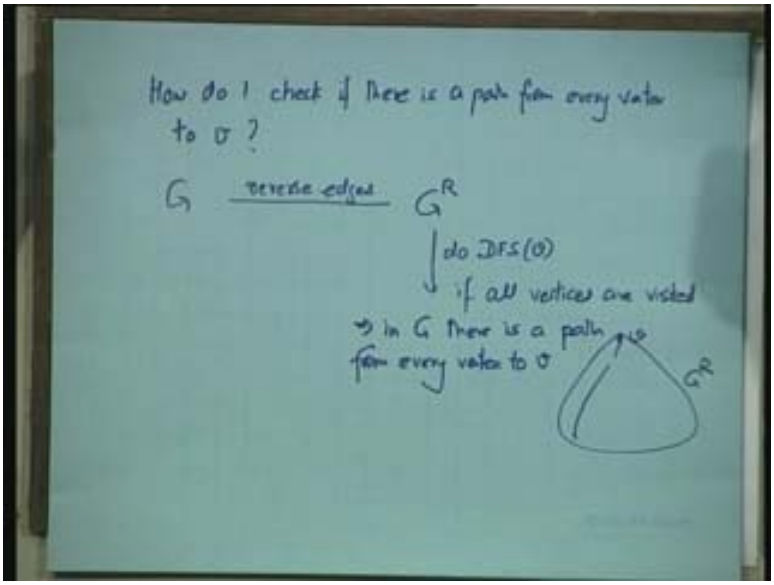
Suppose we could somehow figure out, we could also do the following. We could somehow figure out that there exists or I claim there exists a path from every vertex in G to v. Let's assume this is the case, suppose this is true that is [Hindi Conversation] then does that imply that the graph is strongly connected. Basically we are saying 1+2 implies G is strongly connected. So if we have to find a path between some two vertices like say x and y, so what will I do? go from x to v, by the statement two there is such a path from every vertex to v and then I go from v to y by statement one. So all I need to somehow ensure is that there exist a path from every vertex in the graph to v.

(Refer Slide Time: 05:25)



How will I ensure this? This says that there is a path from v to every vertex in the graph. If I can ensure that there is a path from every vertex in the graph to v, then I am done. How will I ensure that from every vertex in the graph there is a path to v? [Student: from the lower most vertex, we are ending first starting from the path deepest back edge have as a back edge]. So the question is how do I ensure that there is a path from every vertex to v? [Student: sir we can back edges] Think of something new [Student: also we need the cross edges to if there are some cross edges so we can go along from them to]. Suppose I were to do the following.

(Refer Slide Time: 08:14)

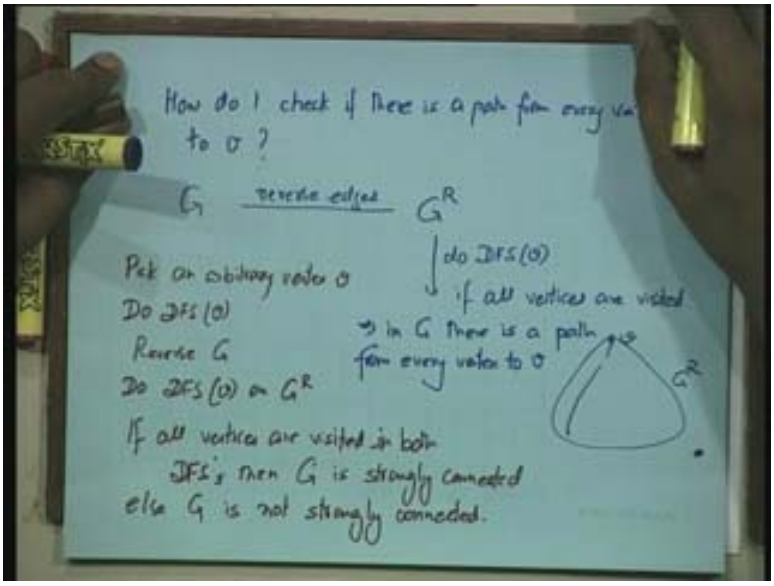


I took my graph G and I reversed [Hindi Conversation], reverse edges to get a graph G sup R [Hindi Conversation] [Student: do DFS from that vertex again now it look at a path] so do DFS [Student: that gives now] do DFS v on this graph G of sup R [Hindi Conversation]. [Student: if you get all the vertices] [Hindi Conversation] if all vertices are visited then this implies, in G there is a path from every vertex to v. Yes or no? [Hindi Conversation] [Student: yes sir] if every vertex gets visited in this DFS then I can say that the graph is strongly connected [Hindi Conversation].

In this graph there is no path from v to this vertex which means that in the original graph there is no path from this vertex to this vertex which means that the original graph is not strongly connected (Refer Slide Time: 09:01)[Hindi Conversation]. [Student: in adjacency list we can fetch change the function go along] we can also change the function DFS instead of looking at out adjacency edges, we can look at inadjacency edges and we don't even have to reverse the graph then [Hindi Conversation].

So essentially by using two DFS's, you can figure out if the graph is strongly connected or not. So everyone understands the procedure. Let me write down what the procedure is. Pick an arbitrary vertex v, do DFS v, reverse G, do DFS v on G reverse, if all vertices are visited in both DFS's then G is strongly connected else G is not strongly connected [Hindi Conversation].

(Refer Slide Time: 11:19)



We will now try to do it with one DFS. It's an academic exercise for this problem but at least you will learn the properties of DFS and because this is perfectly fine algorithm and it just requires two DFS's to do it. So let's look at another way of checking if a graph is strongly connected or not. Now we will use our whole gammon of definitions and terminologies. So what are we going to do now? We are going to do a DFS, we start from a vertex and we do a DFS. So let's say these are the red edges which form the DFS tree. [Hindi Conversation]

Suppose this is what I get my DFS tree as. Now I remarked in the last class that we are going to use ideas similar to what we developed for two edge connectivity. So two edge connectivity [Hindi Conversation], we were saying that when we are back tracking, lets say backtracking from this vertex I am going back to the parent because I have finished everything. I ensure that there is some edge which goes from this sub tree to it can only go to an ancestor or above because we said there are only back edges in the case of undirected DFS. So if an edge goes out of here, it cannot go to one of these nodes. It can go only to an ancestor. So we said we would like to have one such edge.

Now we need similar such thing in the case of strong connectivity. From here we would like that there is some edge going out [Hindi Conversation]. What kind of edges will be going out of this sub tree? Only back edges but if this were the sub tree I was considering then there could also be either there is back edge out of this sub tree or there is a cross edge out of this sub tree. So I could also have an edge going like this. It can happen. So this edge is also an edge going out of this sub tree, if there is an edge going out of the sub tree I am happy. This is the only thing we will require. Clearly this is necessary, if from this sub tree there is no edge going out then this graph is not strongly connected. Why? Because then you can only enter the sub tree, we can only come to these set of vertices, we can go out of this set of vertices. What I mean by that is if there were no edge going out of this sub tree then I cannot go from this vertex to the root for instance to vertex v because there is no edge going out of this 4 edges.

This is like an island in itself, you can only come into here but you cannot go out of here. [Student: but if an edge is going out what do you do?] So all I am saying is that we require that it is necessary that an edge go out whether that is sufficient, we have to figure out. It is clearly necessary that an edge goes out of every sub tree. You understand the difference between necessary and sufficiency. So it is necessary that an edge go out of every sub tree. You understand what I mean by out of every sub tree, so you are looking at the sub tree. What is a sub tree here? It is the descendance of any one vertex, take any one vertex, look at all its descendant that's a sub tree we are interested in and there has to be at least one edge going out of that. This is clearly necessary [Hindi Conversation] no edges is going out of that sub tree then we can stop the process at that point, stop our DFS and say that this graph is not strongly connected.

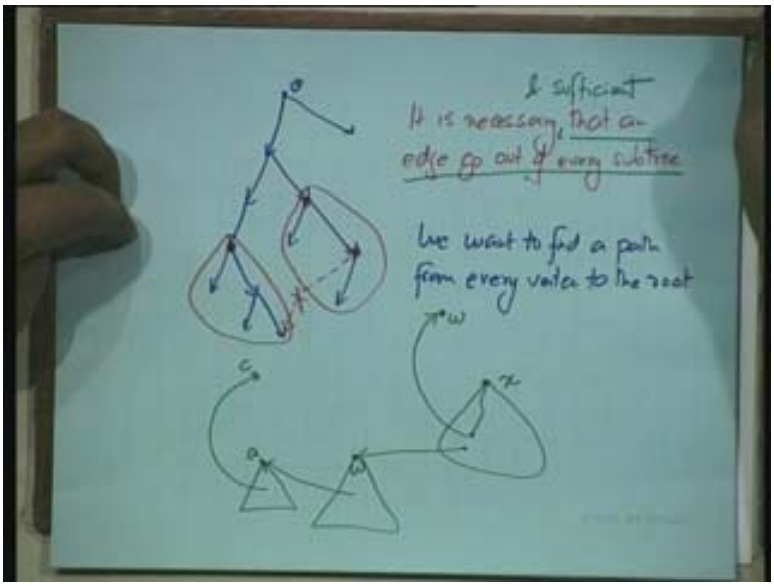
Now we will see how we can check this thing. We will look at the procedure, we will look at what modification to make to the DFS so that we can check that there is an edge going out of every sub tree. The check would be very similar to what we did for the case of two edge connectivity but now let's see that this condition is also sufficient. If from every sub tree there is an edge going out then the graph is strongly connected [Hindi Conversation]. Why? [Student:] why [student: arrival time is the difficulty] of what [student: all the sub trees if there is an edge from here to some other sub tree]. [Student: the vertex that is feature will have a lower arrival time] will have a lower arrival time than whom [Student: then the vertex in all the vertices of the sub tree] close, almost there. So what am I going to do? [Hindi Conversation] from the root I can reach every vertex. Now all I have to show you is that from every vertex I can reach the root.

If I can reach the root from every vertex I am done. Yes, because then how do I find a path between x and y? I go from x to the root and from the root I go to y. So all I have to show you is a path from every vertex to the root. So we want to find a path from every vertex to the root [Hindi Conversation]. We take some vertex, let's say this is the vertex x I want to go from x to the root v. how will I go? I am going to look at the sub tree rooted at x [Hindi Conversation], x is some vertex somewhere in my DFS [Hindi Conversation]. Will it be smaller than x or larger than x? It is smaller than x and [Hindi Conversation] smaller because this is a cross edge.

You would have first come here in fact [Hindi Conversation]. So let's give this name, suppose this node is w so I can reach a node w, so what have we said? We have said that from x, can I go from x to w? Yes, why? Because this is a sub tree, these are all descendant this node is also a descendant of x which means that I can come to this node and then I can take this edge and get to w. So what's the big deal about to getting to w? From x I can get to a node which has a strictly smaller arrival time than x. Now from w I can repeat the same process from w, I can get to a node which has a strictly smaller arrival time than w which means that may be [Hindi Conversation].

Now what do you know? You know that the arrival time of x is strictly larger than the arrival time of w which is strictly larger than the arrival time of a which is strictly larger than the arrival time of c [Hindi Conversation]. I can get to a node with the smaller arrival time [Hindi Conversation] [student: when you reach] when I reach the root. So that will give me a path from this node to the root [Hindi Conversation]. Just this requirement that an edge go out of every sub tree is both necessary and sufficient. Just this requirement and this is an easy requirement to check and we are going to do that next.

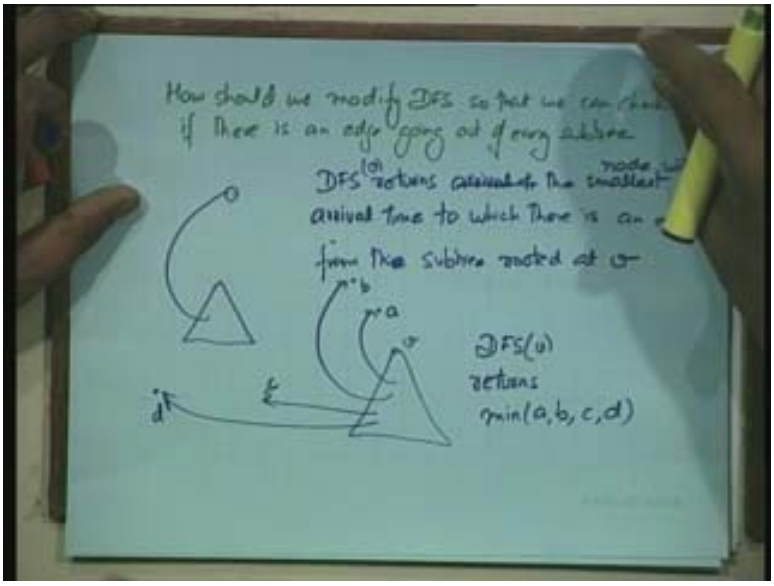
(Refer Slide Time: 21:44)



So how am I going to check this requirement? How should we modify DFS so that we can check if there is an edge going out of every sub tree? I am using the sub word sub tree a bit loosely, here by sub tree I mean the part of that tree which is composed of the descendants of any vertex. So take any vertex, look at all its descendants, that part of the tree is what I am calling a sub tree. The generic definition of a sub tree is slightly different. How will you modify DFS to be able to this? In the case of two edge connectivity we had modified DFS so that it returns to us the deepest back edge. The arrival time of the node to which there is a back edge from the sub tree and the smallest such arrival time [Hindi Conversation]. In the case of two edge connectivity what we have done was that we would return from every sub tree, would return the deepest back edge by that we mean the arrival time of this node.

Now we will do the same thing here. This DFS would return to us, not the deepest back edge anymore, the node with the smallest arrival time to which there is an edge from this sub tree. so let me write that down, DFS returns the smallest arrival time to which there is an edge from this sub tree rooted at let's say v. so DFS v returns this (Refer Slide Time: 25:29). What I am trying to say now is the following. When I do a DFS from v, there could be many edges going out of this sub tree. Let's say there are four edges going out of this sub tree and edge which is going out of the sub tree will either be a back edge or a cross edge. Forward edge cannot be going out of the sub tree, it can only be coming in to the sub tree or if it starts from here in the sub tree it will go within the sub tree only. So now essentially I am going to look at these four arrival times and take the smallest amongst them and that is the quantity the DFS we will return to v. If this arrival time was a, this was b, this was c, this was d, DFS v returns min of a b c d. Clearly b is smaller than a and d is also smaller than c.

(Refer Slide Time: 26.45)

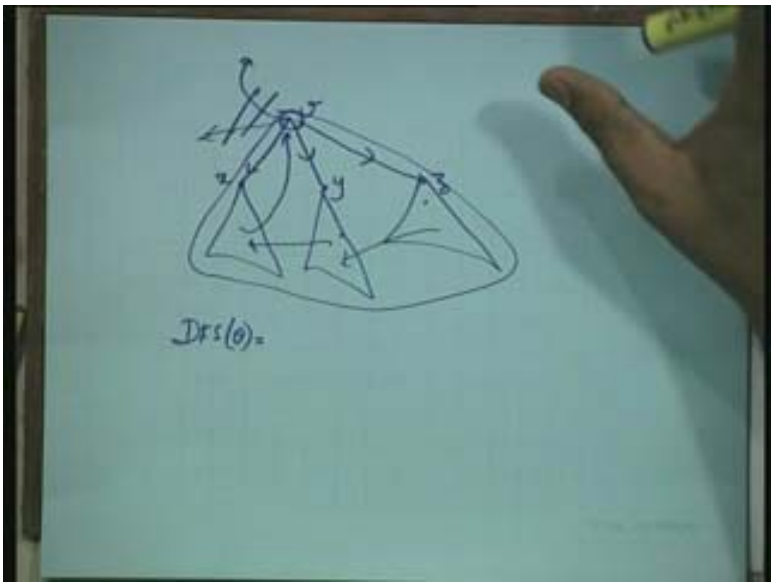


This is what we have to return. Now we have to figure out how we will return this thing. Note that DFS is a recursive procedure, when I do a DFS from vertex v, I end up doing a DFS from its neighboring vertices. So suppose its three neighboring vertices were x, y and z so I will do a DFS from here I will get this tree, let's say I do a DFS from here I get this tree, I do a DFS from there I get that tree (Refer Slide Time: 27:21). Now this DFS x will return something to me. So it will look at all the edges which are going out of this sub tree and look at the node with the smallest arrival time to which there is an edge from here.

Similarly DFS y would return something to me and DFS z would return something to me. So what is the value of DFS v going to be? DFS v would have a value, we have to find out the edge going out of this sub tree. so an edge which goes out of this sub tree would be an edge going out of here or out of here or out of here but an edge going out of here could end up here itself. So it is not really going out of this sub tree [student: we have to check it with arrival time of the.. then it will not] and we will also look at v of course. So from v we will look at the back edges out of v and cross edges out of v.

So we will basically look at all the edges going out of here, all the edges going out of here all the edges going out of here and all the edges going out of v and take the minimum of their arrival times. When will the edge be going out of this sub tree? When that arrival time is less than the arrival time of v [Hindi Conversation] there is an edge going out but [Hindi Conversation] there is no edge going out [Hindi Conversation]. So you have to check that [Hindi Conversation] that is less than the arrival time of v. If it is less than the arrival time of v then that means that the edges going out of the sub tree and we are okay if it is not less than arrival time of v then we can stop the procedure and say that the graph is not strongly connected.

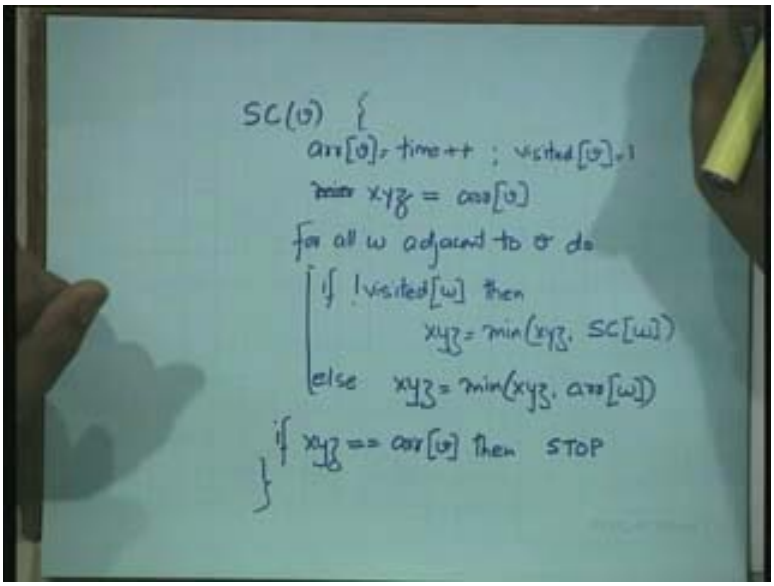
(Refer Slide Time: 29.36)



So now I just have to write the code for this. So what should I do? Let's call this a strongly connected. So what is strongly connected going to do? So first it is going to be setting up the arrival time of the vertex v. Now what should I do next? I need to keep track of this minimum so I need a variable for that so let me declare some variable. What should we call it? Min, no min is for the function so what shall I call it? You can't think of a name of a variable? xyz that's how you name a variable. Let's call it xyz, so what should be my initial value of xyz be? Arrival of v, no harm in setting it to this value and now what am I doing, what should I write next? For all w adjacent to v, out adjacent is the same as adjacent we are saying adjacent means out adjacent here; adjacent to v do, if not visited. Somewhere I have to set visited v equals one. As soon as I started DFS I set the visited variable to one.

If not visited v then what do I do? Then I do a DFS from there or I run this procedure from that vertex w and it will return something to me and I have to update what it returns. So xyz equals minimum of xyz, SC [w], else if visited w is true; if the vertex is w is already visited so then that means that is either a cross edge or a back edge starting from v else xyz equal min of xyz, arrival of w. What are we doing here? We are looking at the cross edges and the back edges starting from the vertex v and also including that into the min computation. Now what are mins? This is the end of this for loop. If xyz equals arrival v then stop and we will stop with saying that graph is not strongly connected else we just return.

(Refer Slide Time: 33.46)

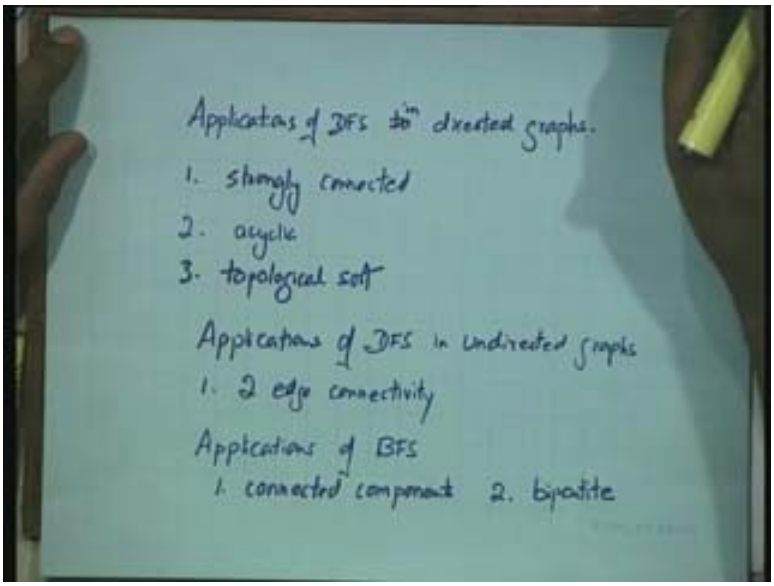


This will again have a small problem for the root vertex because for the root vertex this quantity will turn out to be zero, cannot be less than a zero. So that has to be checked we can't abort always, we have to just ensure that it is not the root vertex [Hindi Conversation]. So almost exactly like we did for two edge connectivity.

So how much time does this take? Almost the same time as depth first search. We have one additional variable which we are modifying and that's all the time, the additional time that we require. So as far as applications of DFS are concerned, you will see quite a few applications. Can someone tell me what all you have seen? Application of DFS in directed graphs, what are the applications we have seen so far? Two edge connected is not for directed graphs. Checking if a graph is strongly connected, checking if a graph is acyclic and of course also topological sort all though it is the same as that. There are a lot of other applications, I am not going to be taking them up in this class. Any questions so far?

So as a recap since we are now going to be switching topics, let me also look at what we have done for undirected graphs. We actually have looked at only one application that was for two edge connectivity. I did mention that you can use similar procedures, you can use DFS to also check if a graph is vertex connected and if it is plain but we did not do those two applications in detail in this class. What application did we see for breadth first search? Finding the connected components of the graph and checking if a graph is bipartite.

(Refer Slide Time: 37.42)



Also the shortest distance that's just breadth first search, the label that it returns are the shortest distance of the vertices from the root, the starting vertex and all of these are linear time procedures.