Data Structures and Algorithms

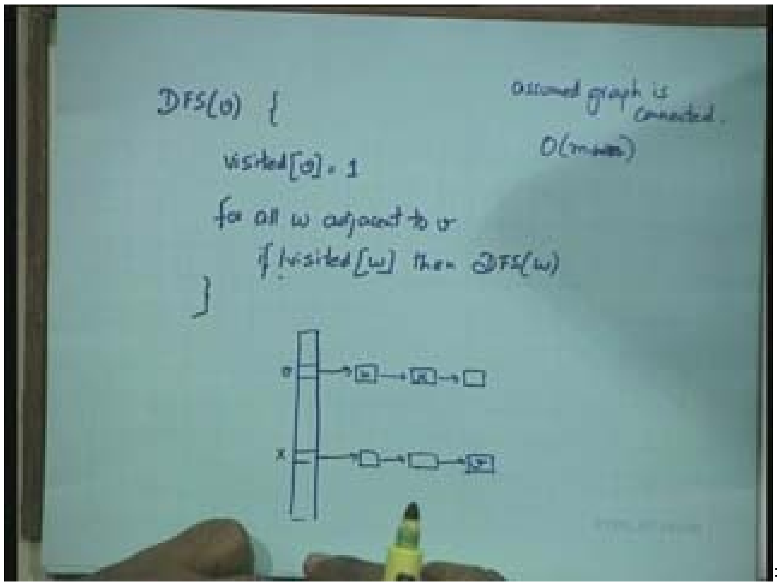
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture – 28
Applications of DFS

Today we are going to be talking about applications of depth first search. In the last class we looked at the depth first search procedure. We will also be discussing the running time of depth first search today and then looking at an application of depth first search to check if a given graph is two edge connected. Recall what we did in the last class. We wrote a small piece of code depth first search v. Can someone tell me what this was? The first thing we do is we said visited v equals one lets say and then I will not worry about the counters for now because that1's for all w adjacent to v. He was saying children of v, there is no notion of children of a node in a graph. For all w adjacent to v, if not of visited w then DFS w, that's it. That's what we said our DFS procedure is. It first marks the node as visited then it will start the DFS on each of the adjacent nodes provided they have not already been visited. That's what DFS v corresponds to. How much time does this procedure take? It is a recursive procedure, so you have to do something careful. Some careful analysis of the running time. How much time does it take? Why? Pardon [student: every] compared, what do you mean by compare? [Student: like we have to check the end is visited or not].

For every edge we will have to look at the edge twice. The answer is right, you are basically doing a total time of order m. Actually I can just say order m because I am assuming that the graph is connected. In a connected graph m is at least as large as n minus 1 so you can always say order m. So let's just say order m and we have assumed graph is connected. Now let's try and see what's happening here. For this I am going to use my adjacency list representation because that will also give you a better picture of what this is. Recall that in the adjacency list representation of the graph, there will be one entry corresponding to node v. This would be the adjacency list of v, the nodes which are adjacent to v. If a node x is adjacent to v then v is also adjacent to x because it is an undirected graph.

So x [Hindi] (Refer Slide Time: 04:57). When I do this step for all w adjacent to v, what does this say? How will I actually translate it into code? This is pseudo code, you don't really have a statement like for all w adjacent to v. What will you do? You will basically have to traverse this list, for that you will have one pointer which points to the first node and then when you have looked at this node then you will update the pointer to point to the next and so and on till the pointer becomes null or reference becomes null. Then you would have reached the end of the list. That's how you will be actually implementing it.
Basically every time I go through this loop I advance that reference, I advance that pointer by one. Now if this was the very first node u and it was not visited then what would I do? I would start a DFS on u. As a consequence I will do some other computation and when that computation finishes, I come back to this DFS procedure. The DFS that I was doing on v and I retrieve that pointer.

(Refer Slide Time: 06:01)



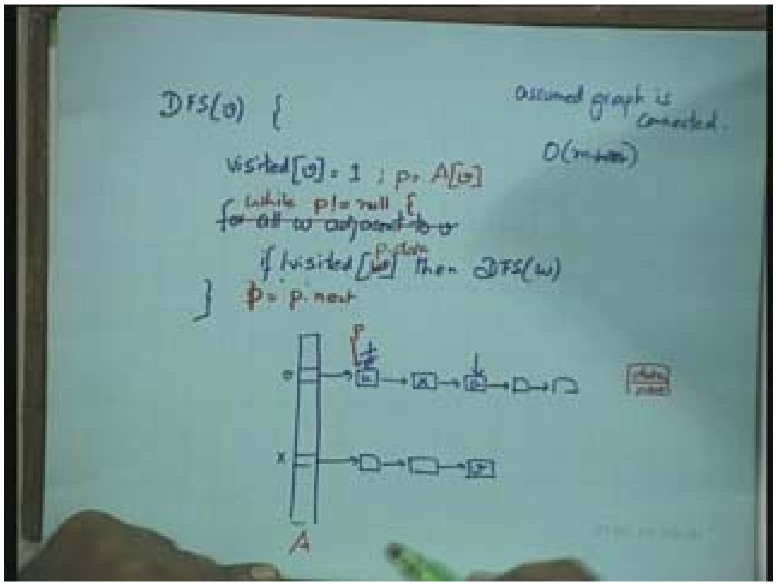
```
DFS(v) {
    visited[v] = 1
    for all w adjacent to v
        if !visited[w] then DFS(w)
}
```

#

Essentially I would have that the pointer is still pointing here. That because that was a local variable so to say. You follow what I am saying? I would not start so, suppose let me say I advance through this pointer. This was the longer list and I reached this pointer reached at this point. Let's say this node is A and A was not visited, so I went and did DFS of A. When I came back from DFS of A then what is the next node I will see in this for loop. I will go beyond this A, I would not continue. I will not start all the way from beginning and how I am retrieving the fact that I was here last, this pointer was pointing to this information. This is coming from the recursion. The fact that in a recursion when I make a recursive call I store the local variables. [Hindi]

If I were not to write for all w adjacent to v, I would have written something like the following. I would have let's call this array A. I would have p equals A of V. What is P? P is this pointer of v, so p initially is pointing to here and I would be replacing this by, while p not equal to null, good. Something like this, this might not be completely correct but it will be something like this. While P not equal to null what would I do? I will do something like this and I will do P equals p dot next. You understand what this is. Basically each one of them has a next and this w will p dot data or p dot node or some such thing. What is w? Because we don't have notion of w, so it will be p dot data let's say. Basically I am saying each one of these nodes has two members, one is data and one is next and P is referring to this.

(Refer Slide Time: 8.45)



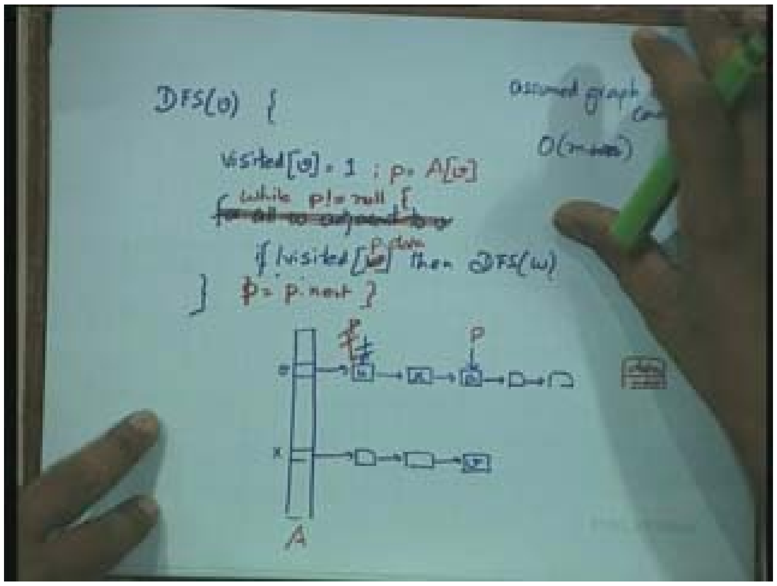
```
DFS(v) {
    visited[v] = 1 ; p = A[v]
    while p != null {
    ~~for all w adjacent to v~~
        if !visited[p.data] then DFS(w)
    } p = p.next
```

Why am I going into all of this? If I have to replace this line, I would have to do something like this. This is how you will actually code it up and this while loop, this braces ends here. The while loop will now have these two statements in it. Why am I doing this? Now note that visited is not local variable, visited is some global array. The only local variable that I have here now is my P. When I make this recursive call, this P is stored on the stack. So that when I return from this recursive call, when I finished DFS of A I was so let's now say P was pointing to here, I am doing DFS of A. When I return from here, I will retrieve this P back and I will increment p or advance p like this, P would now point to the next one. I would come back to this P, I would not come back to this P right at the beginning or some such thing. Why am I saying this? This is crucial for the running time of the procedure. You can understand why, because if every time I was going to start from the beginning then I can't say order m. I am now using a same fact, I will be traversing this list only once for each node. [Hindi]

Once I start traversing the adjacency list of a node, I don't repeat any entry in it. I kind of just keep moving forward in that adjacency list. What does that mean? For each node I am effectively spending time proportional to the degree of that node. Some of that degrees of that node is the number of edges, two times the number of edges in the graph and so the total time required is order m. Yes, she said each edge is visited twice. That's also true, v x, x v. The edge v x is being looked at here when I look at this node because x is now treated as a adjacent node of v and the same edge v x is looked at here, when I look at this node x. Because v is being treated as an adjacent node of v. Actually every edge is looked twice, exactly twice. This gives you the running time. Is this clear? We are heavily using the fact that this is a recursive program and when this recursive call finishes, we retrieve this particular local variable. This local variable gets back the same value it had before this recursive call was made. Now if I told you that implement DFS without using recursion, you will have to use a stack.

You will essentially simulate the recursion by using the stack but now it should be clear what variables will you store on the stack. What information would you store on your stack? [Student: current pointer] basically P, the value of P. In the case of recursion two things are stored actually, not just P someone else has to tell me what else is stored. [Student: v also] v local variable [Hindi] they are stored and the parameters to that procedure are also stored. The parameters to this procedure is v, that will also have to be stored in the stack. Of course the stack also stores return address and stuff like that. But those things you don't need here because you know exactly where it is returning to. This will be your sixth assignment. You will have to implement it. I will give out the details later but you will have to implement DFS without using recursion so that you understand this way.

(Refer Slide Time: 13:16)



DFS(v) {
visited[v] = 1 ; p = A[v]
while p != null {
~~for all w adjacent to v~~
if !visited[p.info] then DFS(w)
} p = p.next }
assumed graph
O(n+m)
p
v
x
A

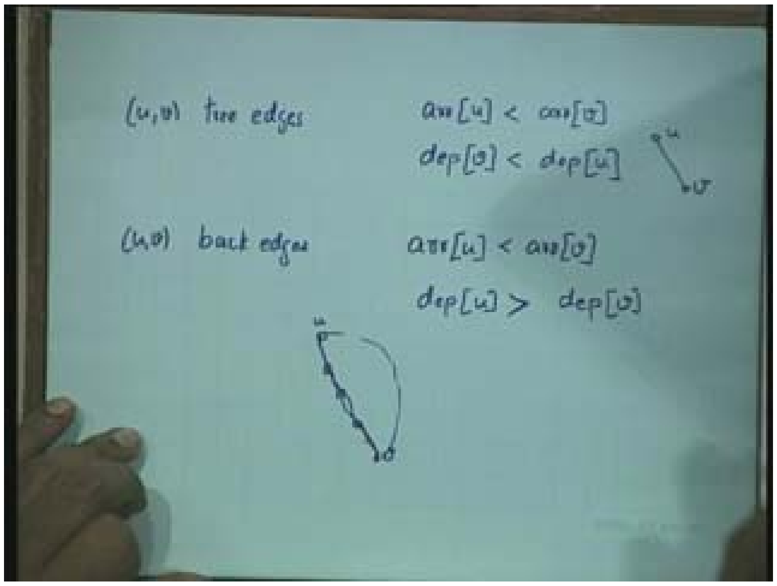
We looked at DFS, we have classified, the edges has tree edges and back edges and we have looked at what the running time of DFS is. One other thing that perhaps we should do before we proceed further is look at this distinction between tree edges and back edges once more. So tree edges and back edges. Suppose I have an edge u v which is a tree edge and if u v were a back edge let's see what can we say about the relation between the arrivals time and the departure times for the nodes u and v. Let's see, let me ask you these questions. Suppose I tell you that the arrival time of u was less than the arrival time of v. That is I reached node u before I reached node v [student: back edge] pardon [student: u v form a back edge] u v form a back edge.

Suppose I told you, I gave you this information. I reached u before I reached v. Now what is the relation between the departure times of u and v? [Student: departure of v is less than is less than the departure of] departure of v is greater than or less than [student: less than less than] departure of u is less than or departure of v is less than [student: departure of um v is less than] [Hindi] you reached u first, u v is a back edge.

You reached u first and then you reached v. What does that mean? v is a descendant of u in the tree. u v is a back edge, so there has to be an ancestor descendant relationship between the nodes of u v. One has to be an ancestor of the other. Which is an ancestor of which? Clearly you will reach an ancestor before you reach the descendant because you are coming down from the top of the tree. So u is a ancestor of v. In fact I have shown that in the picture essentially. So u is an ancestor of v. If u is an ancestor of v in this tree then first I would have finished v and only then I would have backtracked all the way and come back to u and finished u. [student: same] No, this is for the back edge. This is not a tree path, sorry.

There is some tree path between u and v also of course and lets say this is the edge u v. There are other nodes on this tree path, so you would have finished a descendant before you move up because that's how you backtrack. You finished a node and only then you move back up and then you finish that and you move back and back and so on. The departure time of u is more than the departure time of v. You would have left v, we would have finished v before we finished u. What if u v is a tree edge? Would that make the difference? Suppose once again that arrival of u is before arrival of v. If arrival of u is before arrival of v then u is a parent of v. So u v have a parent child relationship not just an ancestor dissonant but an parent child. u is a parent of v [student: why] because it is a tree edge. u v is a tree edge. So when I depart from v after that only will I depart from u. So departure time of v is less than the departure time of u. Note I cannot say that the departure time of u is one more than the departure time of v [student: because] because it might have more children. Similarly I cannot say that the arrival time of v is one more than the arrival time of u because there could be other children also.

(Refer Slide Time: 18:05)



(u,v) tree edges $\quad arr[u] < arr[v]$

$dep[v] < dep[u]$

(u,v) back edges $\quad arr[u] < arr[v]$

$dep[u] > dep[v]$

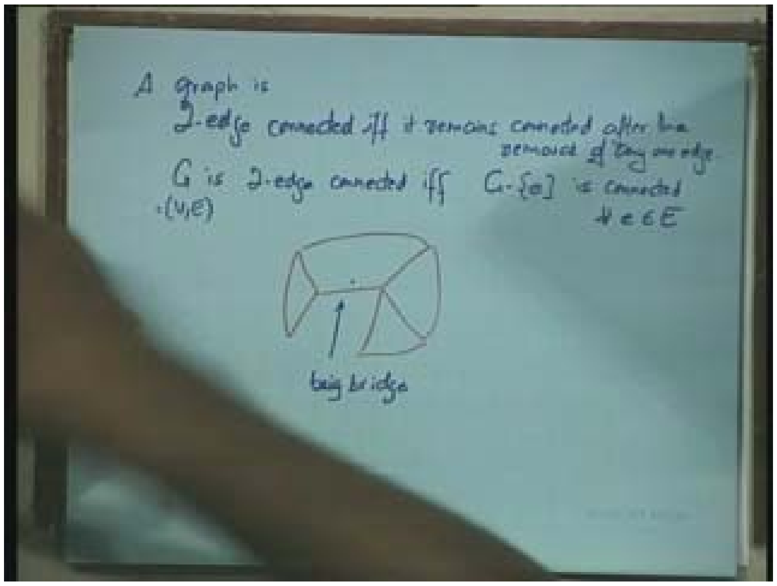
The relationship is the same but these are the same relationship but they coming from different reasons, roughly the same reason agreed, granted.

Now let’s look at an application of depth first search to determine if a graph is two edge connected. Let me write down connected. So that’s the term that I am going to be using. Graph G is two edge connected if and only if G minus e is connected for all e. [Hindi] I am saying take any edge, capital E is the set of edges. When I write graph as v, E; v is the set of vertices and capital E is the set of edges, little e is an edge, G minus e means remove the edge from the graph. If it is still connected and this is true for every edge for the graph then we say that the graph is two edge connected. In words a graph is two edge connected if it remains connected even after the removal of any edge. Yes, only one edge.

A graph is two edge connected if and only if it remains connected after the removal of any one-edge. Let’s see. Is this graph two edge connected? if I remove this edge then it becomes disconnected. Such an edge whose removal disconnects the graph is also called a bridge. This edge we would call it a bridge. This graph is not two edge connected. Is this graph two edge connected? [Student: yes] yes, so graph which is two edge connected will not have any bridge edge. A graph which is not two edge connected will have a bridge edge.

Why do you think this notion of two edge connectivity would be useful? If this were a computer network and some link fails then you are interested in whether the network is still connected or not. If your network was two edge connected to begin with then no matter which link fails. Your network can still keep functioning because it will still remain connected. But if the network were not two edge connected to begin with then the failure of a link can call the network to break down into disconnected components. So that messages cannot go from one connected component to the other anymore.

(Refer Slide Time: 21:55)



A graph is
2-edge connected iff it remains connected after the removal of any one edge

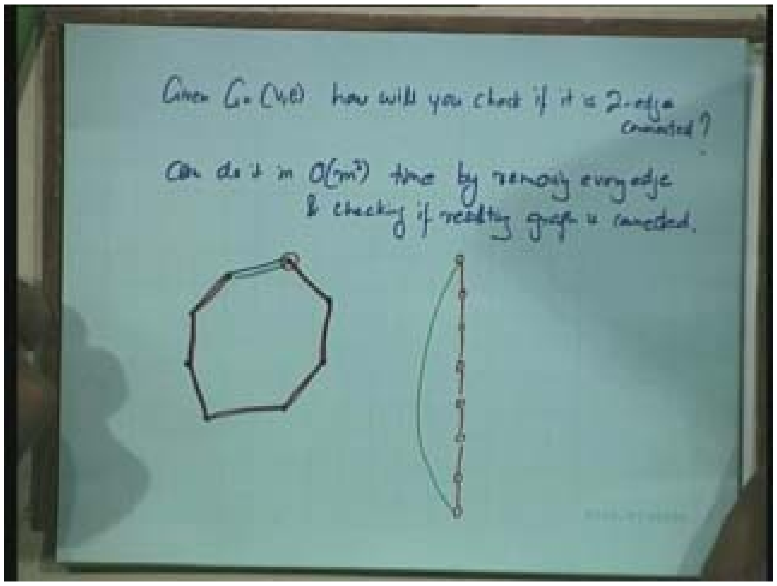
G is 2-edge connected iff $G - \{e\}$ is connected $\forall e \in E$
$=(V,E)$

This is basically measuring liability of the network. Now the question is a following. I give you a graph and I ask you is it two edge connected.

How will you check if it is two edge connected? Anyone? [Student: if we can suppose we have from one end point to the other point of an edge sir each vertex] For each vertex if we can find a cycle. How you will check whether a given graph is two edge connected? We have to do it fast. Pardon [Student: we will look at the departure time] If you look at departure times. [Student: sir between any two node there should be a tree edge also and back edge also] Between any two nodes, there should be a tree edge and a back edge? [Hindi] [Student: so on removal of either of one its still remain connected] [Hindi] Each node should be visited twice? [Student: between any two node] Let's don't worry too much about DFS because that is not straight forward but you will see how to do it.

Let's see can you check this property of two edge connectivity by some other mechanism? [Student: sir by BFS] BFS yes, what will you do with BFS? You have to check if the graph remains connected even after removal of an edge. So take an edge, remove it check if it is connected then take another edge remove it, check if it is connected. Take another edge remove it, check if it is connected. [Hindi] So that's more expensive. We can do it in order m square by removing every edge and checking if resulting graph is connected. Yes, but that's expensive for us. So we want to do something in order m time, linear time. [Student: say that v vertex has a back edge] If every vertex has a back edge only then is the graph two edge connected. [Hindi] Is a cycle two edge connected graph? It is two edge connected. [Hindi] What will be the tree edges? [Hindi] This will be my DFS tree, the one in red and the only back edge will be this last edge.

(Refer Slide Time: 27:05)



Given $G=(V,E)$ how will you check if it is 2-edge connected?

Can do it in $O(m^2)$ time by removing every edge & checking if resulting graph is connected.

My DFS tree, if I were to draw it differently would look like this. It would be a path, my DFS tree with one back edge only. [Hindi] [Student: when you are back tracking from that vertex if you get a back edge that means there is a cycle connected that particular path of traversal so you can then compute something] Good, we are getting somewhere.

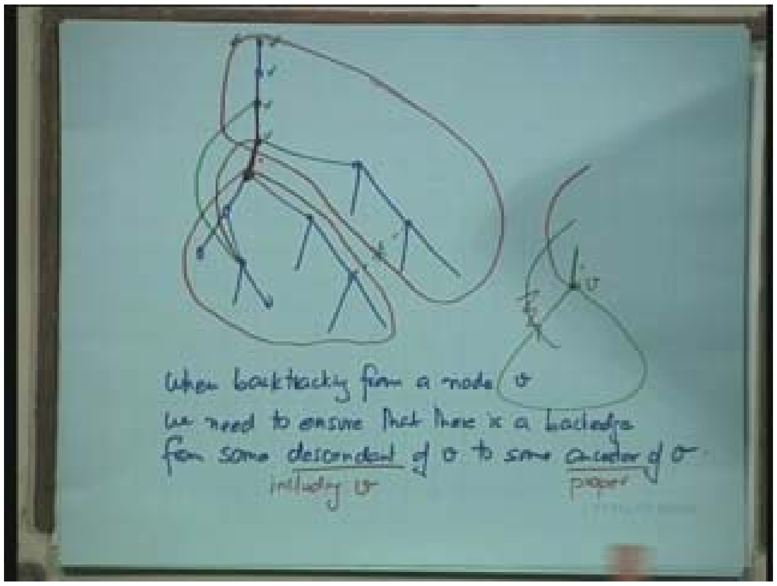
So let’s develop this, one step at a time. I did a depth first search, so clearly we have to do a depth first search. Suppose this is the depth first search tree I obtain. I have just drawn the tree edges, there are other back edges. I have not drawn them yet. Now when I am backtracking out of this node, backtracking means I am going back up because I have explored this entire thing. What do I require? I have explored this entire thing, I have not yet gone here. So I have come from here, I came like this, I came like this, I explored this entire thing.

Now I am back tacking. I want to ensure that this edge along which I am backtracking is not a bridge. I want to ensure that it is not a bridge. [Hindi] What will ensure that this edge, I am only interested in this one edge not being a bridge? What we ensure that this one edge is not bridge? [Student: it is connected, it is also back edge] There is a back edge from where? [Student: from this vertex] From this vertex? [Student: yes yes] From this vertex [Hindi] any of these vertices say [Hindi] (Refer Slide Time: 29:25). If you remove this edge, let’s look at the blue tree, the DFS tree. In a tree when I remove one edge, I will get two pieces not more. This will be one piece and the other remaining piece. I cannot create more than two pieces by removing an edge. I create exactly two in a tree. Now these two pieces are connected among themselves by the tree edges, by the blue edges.

I can go from any node to any other node. How will I go from a node in this piece to the other piece? By going from that node to the end point of this green edge, taking this green edge, going there and then going from here to whichever node I wanted to go to. So which means every pair of node is connected which means that the entire thing is connected, even after I have removed this red edge. This red edge is not a bridge. If such an edge is present then this edge is not a bridge. [Hindi] But it has to go to this node or beyond. [Hindi] I will get two pieces which are disconnected from each other because there will be no edge going from here to anyone here because we said there is no edge of this kind. [Hindi] This is the condition we have to check. Everyone understands? What is the condition we have to check? When I backtrack from a node, I have to check that there is some edge from here which is going to an ancestor of this node. [Hindi] How will I check this? Linear time [Hindi]. There should be some edge from its descendent to one of its ancestors, that‘s all.

Let me write it down in words. When backtracking from a node v, we need to ensure that there is a back edge from some descendent of v to some ancestor of v. [Hindi] I have said descendent without saying proper descendent. So descendent includes the node itself but this ancestor is a proper ancestor which means parent or above. [Hindi] Now we have to somehow ensure this. How will we check this property and we have to do it all fast.

(Refer Slide Time: 35:47)


When backtracking from a node v
we need to ensure that there is a backedge
from some descendant of v to some ancestor of v
including v
proper

[Hindi] We are only permitted order m time. [Student: do we keep track of the back edges then we back track on particular node we believe from our record that particular back edge that node] keep track of all back edges. Do we need to keep track of all back edges? [Student: which] [Hindi] I am interested in a back edge but do I need to keep track of all the back edges starting from… which is the back edge which is of interest. [Hindi] If we keep track of every back edge then we are going to be spending a lot of time. [Student: sir we will delete the] but there is one back edge which is of interest to us. Which is the one back edge which is of interest to us? [Hindi] Clearly if I know [Hindi], you understand what I mean by [Hindi] is going to the node which is closest to the root. How can I figure out which is the deepest back edge? By looking at the arrival time of the other end point [Hindi]. So just by looking at arrival time of this end point, I can figure out what the [Hindi].

Now question is how am I going to build this information recursively. I have to find out the deepest back edge from this sub tree. So I am just keeping track of the deepest back edge. Now I have to find out the deepest back edge from this sub tree. How will I find out the deepest back edge from this sub tree? Suppose recursively I have done this information. I have figured this information. From the sub tree I know the deepest back edge because after all we are doing recursively. When I run the DFS from here, I actually end up running DFS's from let's say these three. Suppose I figured out the deepest back edge from this sub tree, I figured out the deepest back edge from this sub tree and I figured out the deepest back edge from this sub tree.

How can I compute the deepest back edge from this sub tree? [Student: compute that compare] Compare all three and take the minimum. [Student: and from u also] the one with the minimum arrival time. [Student: corresponding with the minimum arrival time]. So this will give me an edge whose other end point has a smallest possible arrival time. This will give me an edge whose other point has a smallest possible arrival time, this will give me an edge whose other end point has the smallest arrival time. Now if I take the minimum of these three, it will give me the edge which has the smallest possible arrival time which emanates from this sub tree. Is this true? [Hindi]. Now let's write our DFS procedure. So this is our, lets give it some other name to distinguish it from DFS. Let's call it two EC, two edge connectivity. So we are writing a two edge connectivity procedure. Once again it will take as input a particular node. We are going to write it as the DFS thing and I will tell you how we have to call the DFS, this two edge connectivity procedure eventually.

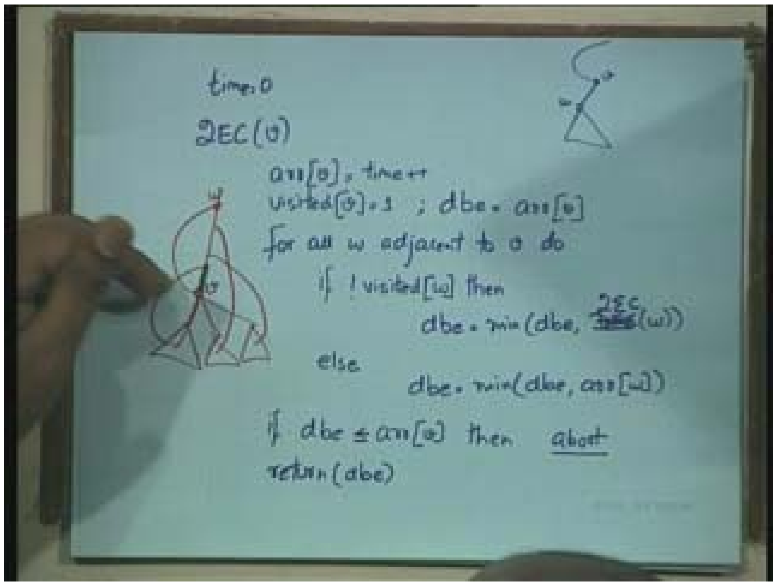
Recall I need the arrival times of the nodes. So I should maintain my arrival counter suitably. So what should I do from my arrival counter? [Student: time is equal to zero] [Hindi] For all w adjacent to v, same thing. For all w adjacent to v do. What should I do? [Hindi] Arrival value of this node itself would be a natural thing to do. Let's set that at that. Now deepest back edge [Hindi] as I do my DFS calls, I have to kind of keep updating this variable. What should I do for all w adjacent to v do. If not visited w then then what should I do? [Student: then if then arrival v or smaller than dbe] arrival v then dbe equal minimum of dbe, 2 EC of w [Hindi]. [Student: sir even it is even it is visited] if the node is visited then what does that mean, what edge is it? [Student: back edge] it is a back edge [Hindi]. So else what should I write on here? [Student: if you are maintaining that d ec w and some particular array then you can write that minimum of I will say I have to write the same thing again minimum]. We will just write the same thing, dbe equals minimum of dbe comma [student: 2 ec w arrival of] not 2 ec, we are not running a arrival of w. [Hindi]

Now what do I have to check? [Student: if it is less than or] if dbe is less than arrival v. [student: never be possible] [Hindi] If dbe is less than arrival v then continue. [Student: There we can then we can continue] then we can continue. If dbe equals arrival v then [Hindi] then abort. Basically then you stop your procedure saying you found a bridge. You can do whatever you want, I will just write abort here. You should not write abort, you should end gracefully. But you understand what I am saying. Basically why have we said equal to and not greater than or equal to. Greater [Hindi]. [Student: sir can you explain else part if w is] he wants me to explain the else part. Why do we need the else part, you are wondering. [Student: we need the else part] you need the else part, great. [Student: but why arr w what is the significance of w] what arrival w? The else corresponds to a back edge starting from v. It is going to a node w. w is a node which is adjacent to v, so it is going to a node w.

How am I keeping track of deepest? I am keeping track of deepest by arrival numbers of the nodes. So that's why I am comparing it the arrival of that node with this. [Hindi] so I need that and if the deepest is less than this v then its okay, I can continue. If it is here only, deepest is this then this means that this edge is a bridge. [Hindi] No, we cannot say anything about the arrival time of this verses the arrival time of this. I have said this before. We cannot say that this is one less than that. Basically we have to modify this so that I am not considering this edge, this tree edge. This has to be modified so that the tree edge is not considered, this parent edge is not considered [Hindi] (Refer Slide Time: 49:55).

When we do the DFS from that vertex, we visit all the vertices. All the back edges, if any coming from below cannot go to some smaller number. Clearly they are only going up to this vertex because this vertex has the arrival number zero. There is now vertex with arrival number less than zero. [Student: dbe should not be zero] [Hindi] basically there are many ways of doing it. You could perhaps have marked this edge as a tree edge. You will have to think of ways of doing this. I will leave that as an exercise. These are two minor things but they are important. Your procedure would not run at all, if you were to ignore them. What is a running time? [Hindi] So essentially as same as before. For every edge we are spending a constant amount of time. The total running time is still order m. this is clear? So actually very sophisticated procedures can be built on top of DFS.

(Refer Slide Time: 52:46)



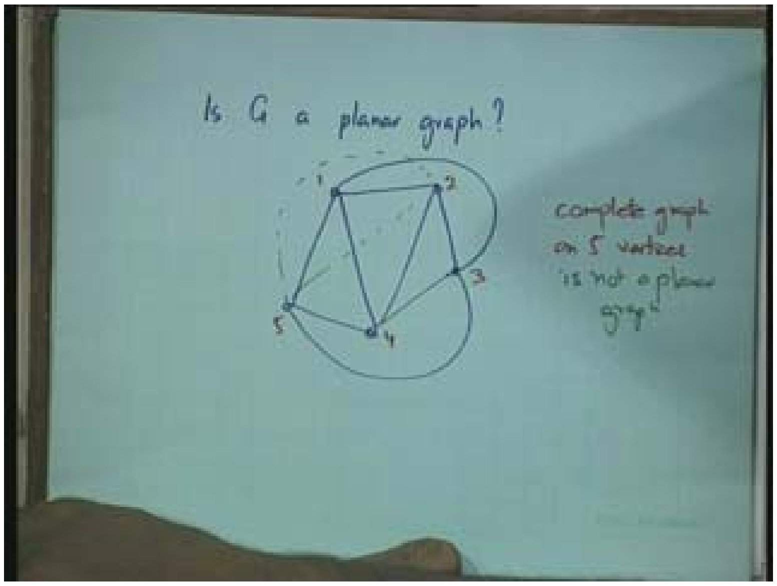
```
time=0
2EC(v)
    arr[v] = time++
    visited[v] = 1  ; dbe = arr[v]
    for all w adjacent to v do
        if !visited[w] then
                dbe = min(dbe, 2EC(w))
        else
                dbe = min(dbe, arr[w])
    if dbe = arr[v] then  abort
    return(dbe)
```

There are many other graph problems which can be solved in liner time. They might seem very complicated problem but you can essentially solve them in linear time using depth first search. I will mention one other problem because I have a couple of minutes. We will not of course discuss it. The other problem is, is G a planar graph? Do you know what a planar graph is? You are not done a discrete math's courses. No, you are doing it next semester. [Hindi] that is not sufficient. What is a planar graph? A planar graph is a graph which can be drawn in the plane such that its edges do not intersect. [Hindi] This is a planar graph. I can draw it whichever way I want but the edges should not intersect.

Now suppose what is this graph? This graph is the complete graph or almost the complete graph on five vertices. I told you what the definition of a complete graph is. Complete graph or a cleak (Refer Slide Time: 54.55) Is this the complete graph, compete graph on five vertices. No, why not? The edge 2 5 is missing, [Hindi] 2 5 is the edge which is missing. [Hindi] There is no way I can draw 2 5 here without crossing and that has nothing to do with the way I drew the initial thing [Hindi]. There is no way I can draw 2 5. Actually this complete graph on 5 vertices is not a planar graph [Hindi].

If I were to draw it this way, it would cross with this. If I were to draw it like this, it would still crosses with this edge. If I were to draw it like this, it would cross with this edge and so on and on. There is no over drawing this. So that's the question, is a given graph a planar graph. This problem can be solved using depth first search. So that you can get an algorithm which runs in linear time, order m time, very sophisticated algorithm to check if the graph is planar or not.

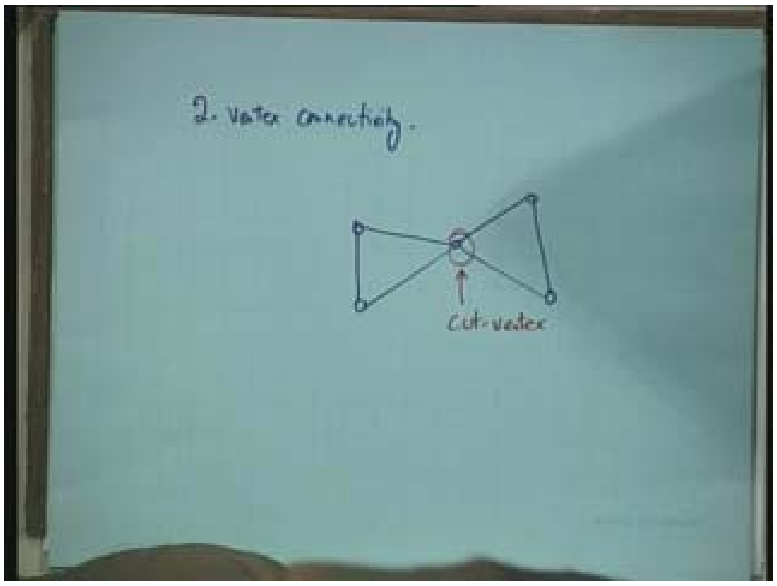
(Refer Slide Time: 56:07)



This is an example of a non planar graph that I had shown. You learn more about this in your discrete math's course. What are non-planar graphs, what can you say about non planar graphs? [Hindi] This is another example of depth first search but we are not going to be taking up this in this course. There is one third example which I will do in two minutes. So just as I defined two edge connectivity, I can define two vertex connectivity.
Just replace the edge by a vertex. A graph is two vertex connected, if removing any vertex still keeps the graph connected. This corresponds to computer failures.

Now instead of link failures earlier [Hindi] then you would call it a two vertex connected. So no matter which computer breaks down. If the network is still functioning, it is still connected. Then you would call it a two vertex connected graph. For instance this would be an example of a graph. Is this two vertex connected? No, why because if I remove this vertex, it becomes disconnected. When I remove a vertex, I also remove the edges incident to that vertex. Clearly it becomes disconnected but this is a two edge connected graph. This graph is two edge connected. Once again the same question given a graph, is it two vertex connected that can be checked by depth first search in linear time. Such a vertex is called a cut vertex, bridge [Hindi] corresponding notion is cut vertex here.

(Refer Slide Time: 58:31)



In today's class we have done example of depth first search which is checking if a given graph is two edge connected. There are many other application that depth first search can be put to. I have shown you, I mentioned briefly two examples, checking if a given graph is a planar graph and checking if a given graph is two vertex connected. So next class we are going to look at depth first search in directed graphs and see how it is going to be different from undirected graphs.