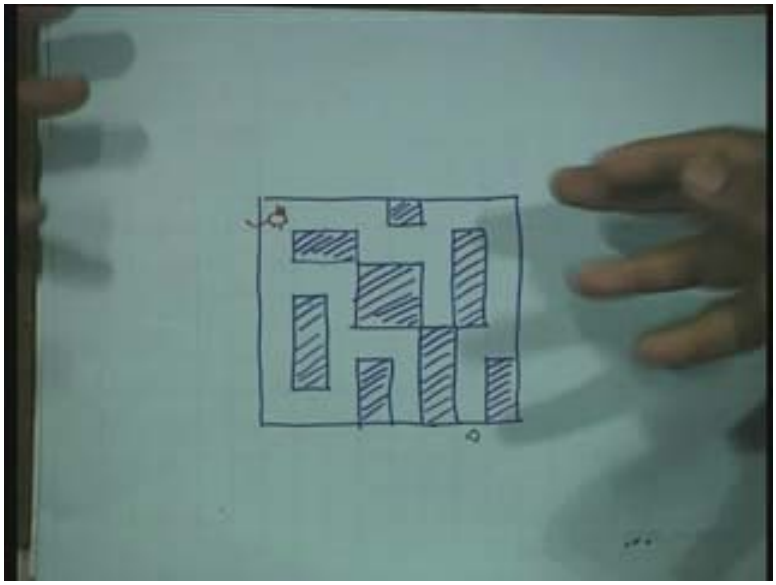
Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture – 27
Depth First Search (DFS)

Today we are going to be talking about depth first search. This is another way of searching a graph. In the previous class we looked at breadth first search. There are certain applications where depth first search is more meaningful than breadth first search and we are going to look at these application also. One application is the following. You are a mouse, you all know who you are. So you are a mouse and there has to be whatever something, carrot. No, I thought you said carrot, it was cat. There is a piece of cheese at the end of the maze and you have to find your way to this cheese.

(Refer Slide Time: 02.06)



This can be thought of as a graph in the following sense. At each of these squares I put down a vertex. We come to that in a minute. What will a mouse do? The mouse is not going to do a breadth first search unlikely that is going to do. It is not even going to do a depth first search but what it ends up doing is something like a depth first search. What this mouse is going to do as all other mice would is that it is going to go off in one direction. Try to explore that path, that direction fully and if it's not able to get to the cheese, it is going to try and backtrack. We will understand what all of that means.

Let’s assume that our mouse has photogenic memory. When it comes back to the certain place, it knows that it has been at this place and it knows what path it took when it was in that place last time. We of course have data structures to keep track of this information but let’s say mouse can also keep track of this. Let’s say it started from here and it went down one step and it did not find its cheese. So it came down another step not yet, so let’s say it decided to go right. Why? I don’t know, it just decided to go right. It is said to go right and then it moved another step and at this point, it can only go down. It went down and again it came down here and lets say whenever it has option it can always goes right. It goes right again.

Let’s say that the mouse tries to take this direction first, east and then it tries to take this direction south and west and north. Now there is no doubt about left and right. It will try to go again east and did not find the cheese, so now it cannot go east, so it goes south and south and now it is stuck. It cannot go anywhere else because all the three sides are blocked but it can move, when it gets stuck like this it backtracks. Backtracks means goes back to the place where it came from. It goes back and at this point, it knows that it has been here before. It went south, so it now tries to go west but it cannot go west. So it has no other possibility but has to go back to where it came from and it came from here, so it goes back.

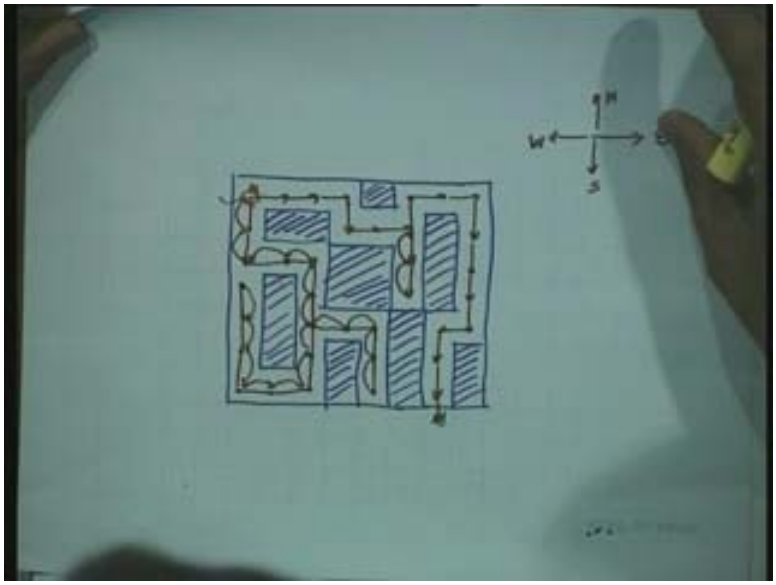
Similarly here it goes back and it goes back. At this point when it reaches, it sees that it has another option which it has not explored which is going south. So it goes south now and then it comes here, it comes here. Now it will have no other option but to go west, west, north, north, north and now it comes to this point. It comes to this point but it sees that this is the point which it has been to before. It is already been to this place. It is not going to go to this point any more. If it has been at a particular cell before at a particular location before, it will not go there again because otherwise it will just keep going in a loop. It will not make any progress.

Since it has been to this point before it will not go here. At this point it has no other option left, it cannot go east, it cannot go west. If it goes north, it goes to a place where it has been before. So it backtracks. At all of these points it will have to backtrack because there is nothing to be done and it comes back at this point. At this point it had explored the two options it had. The third option is not an option, it cannot go west so it will backtrack, backtrack, backtrack come here.

Now here if it tries to go south, it sees well this is a place I have been to before. Why should I go there? It is after all just trying to find where the cheese is. If it has been to a place before there is no point going there again because it will then repeat the same sequence of steps. So it doesn’t go here. That’s one important thing that the mouse does and so it backtracks and backtrack and comes back to the starting position. What have we learnt? It is not so easy to find cheese. Now it will take the other option comes here, comes here, comes here, comes here, comes here, comes here. He said it will first try to go south, south, struck, backtrack, backtrack. At this point it has another option goes there, goes there, goes there, goes there and now it is a homerun. Time to clap, no.

It has found the cheese. This is what we will call depth first search and we will formalize this shortly. But this is fairly natural way of exploring. It is not artificial, you don't have to be a mouse to be doing this. You take a particular direction and you try to follow it till wherever it will reach till whatever you can, using that direction. So direction here now corresponds to in our graph sittings, it will correspond to taking one edge, going out along that edge to the next vertex and continuing and seeing where all you can reach. Then if you cannot reach any other vertex then you start backtracking. While there was a notion of goal here which was the cheese and you would stop here. In our depth first search there is no goal really. So our depth first search would just be a mechanism of exploring all the vertices of the graph. We will keep continuing our depth first search till we do not come back to the starting vertex and there are no other options left from the starting vertex so to say.

(Refer Slide Time: 7:58)



Our search is basically a method of visiting all the vertices with the graph. Now let me do a depth first search on a graph and show you what you get. I will start including some terminology now. This is the graph, we are working with an undirected graph for now. The notion of a depth first search is applicable also for directed graphs. So small simple graph and six vertices and I am starting from this vertex. So as in the case of breadth first search there is always a notion of a starting vertex. I am starting from this vertex. What am I going to do? I am going to take one edge out of this vertex. In the case of the mouse we choose a particular ordering. We said first we will take the option going east then south, then west, then north. Here we can choose whichever we want and typically we are going to take… so recall we are working with the adjacency list implementation.

For every node, so recall an adjacency list implementation, you have an array and for each vertex you have a linked list and this linked list is a list of adjacent vertices. The first edge I will consider is the edge going to the first vertex and when I have worked with this edge and I am explored everything, all the places I can reach with this end.

I come back to this vertex then the next vertex I will consider is this one and so on and on. Suppose here this was the first edge I considered. I went along this edge to this vertex. I am now going to just so that you follow the procedure. I am going to do a time stamping. What does it mean? Nothing much. I start at time zero. When I reach a new vertex, I increment my time. I will say I reach this vertex at time 1, so 1, 2, 3 will tell you what is the sequence in which I visited the vertices. I reach this vertex at time one. Now I start from this vertex, I am going to look at the label options I have. I have three different options.

Let me say I first took this option and come to this vertex. I came to this vertex at time two. At this vertex do I have any other option? There is this edge going out but when I go along this edge, I see I come to a vertex which I have already been to, vertex zero here. There is nothing to be done at this vertex. What do I do now? I backtrack, I leave this vertex. I am done with this vertex. When I am done with the vertex I am again going to increment my time. I came to this vertex at time two and I am going to say I am done with this vertex at time three. I will just increment our time counter. This is not particularly useful this time counter but there will be one application where we will use it. This is just right now to show you how things are progressing. I am done this with this vertex and then where do I go back? Backtrack which means go back to the vertex where we came from. So I come back to this vertex. Now can I backtrack out of this vertex? No, because there are no other adjacent vertices which I have not explored.

Let's say this is the next one I go to or let's say this is the next one I go to. Why not? I take this one and I reach this vertex at time 4. No, our numbering is when we reach a vertex, we increment the counter and we backtrack from a vertex then we increment the counter. That's the only way we number. I am not saying that I leave from here at time three, I get to here at time four then I could do that also but why keep incrementing unnecessarily. I am going to increment only when I reach a new vertex, so I reached a new vertex I give it a time stamp of a 4. Now at this vertex what are the other options I have? I can go along this edge or I can go along this edge.

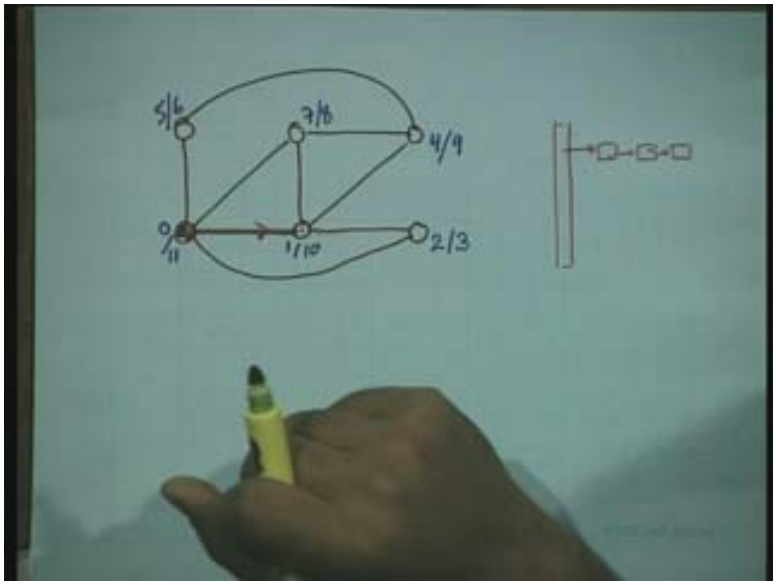
Let's say I decided to go along this edge. I decided to go along this edge, I came to this vertex at time 5 say. At this point I could go along this edge to this vertex but again this vertex is already visited. I am not going to do that. Since there is no other option left, I will backtrack out of here. I have finished my visit at this vertex at time six. Always backtrack to where we came from. We came from 4 so we go back to that. Now I come back to 4 but there is one other option at four which I have not explored which is going like this. I am going to follow that now and reach this vertex at time 7. Yes? When I reach this vertex at time 7, now I look at the other options I have. This is one option but this leads to me to a vertex which is already visited.

This is another option I have but this also leads me to a vertex which is already visited. So no other options, I backtrack out of seven. At time 8 I backtrack out of seven and I come back to this vertex. At this vertex I have explored all options. I went along this edge, I went along this edge and had come along this edge, so this is the only edge left. In some sense I go back along this. I have explored this, I have explored this so I am ready

to backtrack out of this vertex. I backtrack out of this vertex at time nine. Backtrack out of this vertex at time nine and where do I end up? At this vertex. (Refer Slide Time: 15:48) So now reached this vertex. At this vertex I have explored this possibility, I have explored this possibility. I have not yet explored this possibility but this is meaningless because this vertex is already visited. I have explored all possibilities out of this vertex, so I now backtrack out of this. I backtrack out of this at time 10 and I come back to this vertex. At this vertex I have only explored this edge yet.

Now I go explore this edge but this is going to a vertex which is already visited, so I cannot do anything. This is going to a vertex which is already visited so I cannot do anything. This is going to a vertex which is already visited, I cannot do anything. I am done with this vertex also. At time 11 I finish at this vertex. Each vertex is been given two numbers. The time at which we came there, the time at which we left, 6 vertices so there should be 12 numbers in all.

(Refer Slide Time: 16.49)

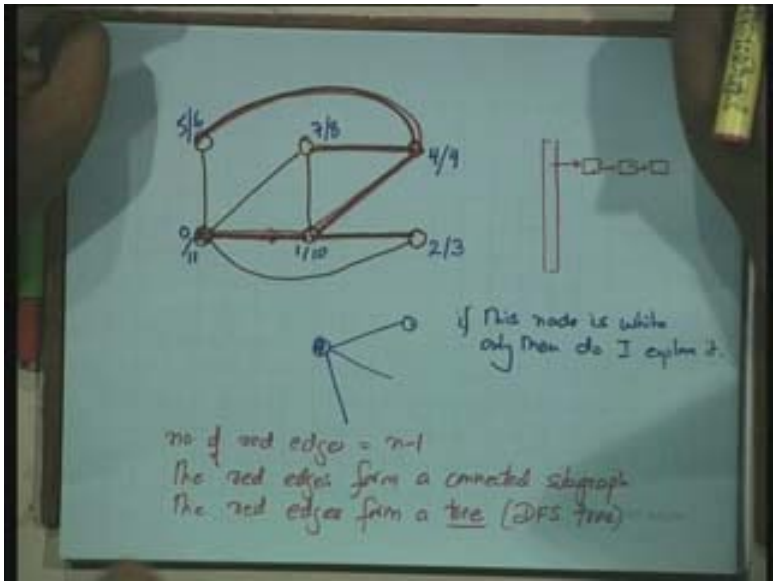


That's what we have 0 through 11. This procedure is called a depth first search. Any questions so far? Once again we will have a notion of black, grey and white vertices just as we had in the case of breadth first search. What should our initial color on the vertices be? White, unvisited same as breadth first search. When should I color a vertex grey? When I reach that node then I color it grey. When should I color it black? When I leave that node eventually then I should color it black. Is it clear to everyone? Same as in the breadth first search. When I had removed the node from the queue, when I inserted the node into the queue in breadth first search I color it grey. When I removed it from the queue I put its neighbors back into the queue. If any of the neighbors is white I put into the queue and I color this node black. So corresponding thing here would be when I backtrack from a node. I have explored all possibilities going out of that node then I color that node black. So if I am at a node, I will look at its neighbors and which neighbor will I go to?

Suppose this is the first neighbor I consider, in what circumstances, what should be the color of this neighbor so that I would go to this neighbor, white. Yet to be explored. Only if this node is white, will I go to this node. If it is black or grey then I will not go to this node at all. I am at this node, this is what I am saying. If this node is white only then do I explore it so to say. Let's understand this example a bit more. Let's mark out the edges along which we traveled in the course of our breadth first search with red. I travel along this edge because I came from here to here along this edge, how did I reach two from this one? I traveled along this edge, so once I went along this edge then I backtracked along this edge. Then I went along this edge then I went along this edge, then I backtracked along this edge. Then I went along this edge, backtracked along this, backtracked along this, backtracked along this. Each one of these red edges, I have went along them twice.

Once I went along the edge, the other time I backtrack along the edge. How many such edges are there? How many red edges are there? [Student: n minus 1] The number of these red edges would be n minus 1. Why? [Student: forward tree] why it is a tree? Who said it is a tree? We are going to use a different argument for it's a tree. First why should this have n minus 1 edges? For every vertex there is one edge along which I came to that vertex and the same edge along which I backtrack from that vertex. So for every vertex there is a unique edge except for the first vertex. Since for every vertex there is a unique edge except for one, there should be n minus 1 edges. So number of red edges equals n minus 1. If I just look at the red edges, they form a connected sub graph. Why do they form a connected sub graph? Because by walking along the red edges, I could visit all the vertices. Yes, all I did was walk along the red edges. That's all I did. So by just by using the red edges, I could visit all the vertices starting from the root, from this start vertex. These red edges form a connected sub graph.

(Refer Slide Time: 23:00)



So connected sub graphs with n minus 1 edges is a tree. Yes, we have done this before. If I have connected sub graph with only n minus 1 edges in it, it cannot have a cycle we have proved this. So it's a tree. The red edges form a tree and this tree is also called the DFS tree, the depth first search tree. Is this clear? Just as we had a breadth first search tree, a breadth first search tree was defined in terms of predecessors. Here also we can have a notion of a predecessor. What is the predecessor of this vertex? The vertex which was visited at time one because why should that be the predecessor? I came to here from there so that is an actual notion of predecessor and these edges then… the same thing, same idea is getting repeated.

Just as we had a breadth first search tree there, we have a depth first search tree here. But the breadth first search and the depth first search tree are completely different. What I am going to do now is to redraw this tree. I am going to keep it like this. Can you all see the picture? I am going to redraw it so that it looks like a rooted tree now. This is this vertex and let me draw it in brown. This is that and the next vertex would be there and from here I have this, so this is this vertex that I have drawn, from here I have an edge to there and I have another edge. These are all the 6 vertices.

If you want, I will put down the numbers so that you can also see the correspondences. This is 0 slash 11, this is the 1 slash 10, this is 2 slash 3, this is 4 slash 9, 5 slash 6, 7 slash 8. Let me also draw the other edges of the graph. Right now I have only drawn the tree edges, the DFS tree edges. These are the edges along which we traveled. Let me also draw the other edges. How many other edges do we have? We have 4 other edges. There is one edge from here to here, there is one edge from 1/ 10 to 7/8. This is another edge, this is this edge. There is one edge from 7 8 to 0 11 this edge, there is this edge 0 11 to 5 6. Let me draw it like that. This is an undirected graph so these directions that I have shown are meaningless. It is just to signify that this is how we moved and now I can get rid of this picture. This is our graph. I have just redrawn it so that now it looks like a rooted tree and the predecessor of the node now is just the parent of that node.

If I define this as the root then there is a natural parent child relationship between the nodes. The parent of this node is this and it is also the predecessor. So quite often we will talk in terms of parent child siblings and all for a DFS tree. When we say that basically means we are thinking of the starting vertex is the root of the tree and we are basically hanging the tree from there and then whatever is the parent child relationship that gets defined that's what we are working with.

Let's look at more properties of depth first search. These green edges, so the brown edges are called tree edges, that's the brown edges. The green edges are called back edges and we will see why they are called back edges and why not front edges. What's back about them? Now let me think of depth first search has been done on this graph. This is the entire graph. All the edges and the vertices are here. I started from this vertex, I came down to this, I came down to this and then I looked at, this was one option available to me. But this edge was going back to a vertex which I had already visited. Yes, and so this is called a back edge, going back to a vertex which is already visited. [Hindi]

Then it is going back to a vertex which was already visited so I don't go there. There is no other option left here so I backtrack. This is the option here I come here, I come here again this is an edge going back to a vertex which is already visited. I don't go along this edge. I backtrack from here, I come here. This is an edge going back to a vertex already visited. I don't go along this, I backtrack, I backtrack. I could now look at this edge again and say well this is going ahead to a vertex already visited. I could potentially have also called it a front edge but since we first considered it as a back edge, we are going to stick to the term back edge.

This is an edge we going to a vertex already visited, so we don't consider this and since all option I exhausted here, back track, I come here. These are all edges which we have so to say have been classified as back edges. We don't or they are going to vertices which are already visited. We don't do anything and we are done. Who can formally define for me what a back edge is? What is a property of a back edge? Now I think I am getting ahead of myself. Now I have the following question, could there be an edge from this vertex to this vertex. Could this dotted red edge be? So I have drawn a dotted but most likely it should not be there but why? When I came to two, when I came to this vertex at time two then I backtracked out of this vertex only because there was no other option available to me. But if this edge was there then this was an option available to me why because this vertex was not yet visited. This vertex was visited only at time 4. At time 2 this vertex was not visited and so it is still a white vertex and so I would have gone along this edge and if I had gone along this edge then this would not be the picture at all.

Clearly this edge is not there in the graph. [Student: sir we have the level difference] No, we will understand what these edges are in a second. There is nothing to do with level here unfortunately. This is again two we are saying edges which jump a level in breadth first search, they cannot be because if they were then that would not have been our breadth first search, that would have not been this collection of levels. If this edge was there then this should have not been the picture at all, it would have been something completely different. So this edge is not there. This edge is not there, similarly this edge from here to here is not there or from here to here is not there.

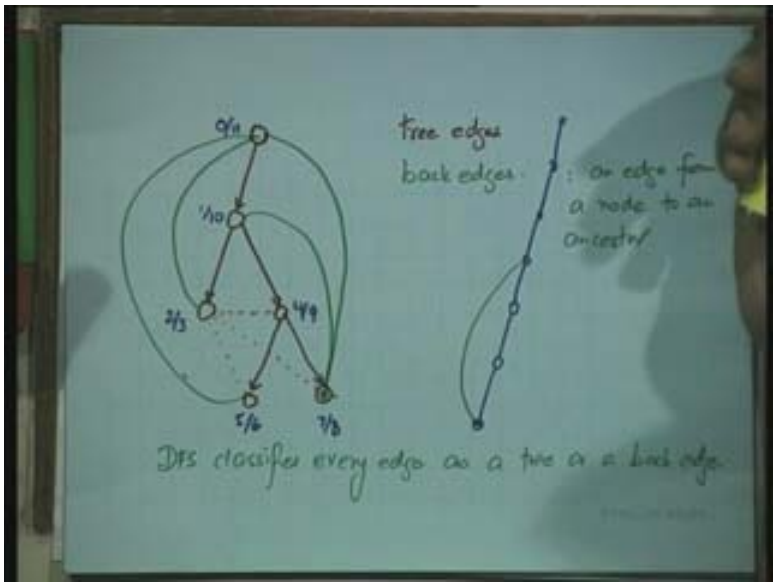
What are these edges which are not there? What can I say about edges which are there and edges which are not there? Sibling but this and this are not sibling. [Student: there is no ancestor which]. What are the edges which can go from here which can emanate from here? They are only edges which can go up to root or to ancestors? [Student: ancestors] So let's understand this, this is an imp very important point. I have reached the certain vertex and this is let's say the sequence of vertices along which I reached here. I am sitting at this vertex at this point and I am ready to backtrack, ready to backtrack means there is no other option available to me. What are the other edges which could have started from here? These are only two vertices which have already been visited. These are clearly vertices which have already been visited. There could be edges from here to here but why can't there be an edge from here to some other vertex which has already been visited. [Student: there will be no need instead of vertex] No, so why did you say only ancestors? Suppose this was my vertex here, why are we saying that only two ancestor.

Why can't we have an edge from here to here? [Student: because then it would be a ancestor of this and have been visited earlier it would be] If there was an edge from here to here… [Student: it has to be child of ancestor when do we backtrack before backtracking we would cover that vertex that path would have already been covered]. What is a formal way of saying this? Why should we have only edges from a vertex? Let's just put down what we have concluded so far. From a vertex we can have edges only to ancestors and such edges are called back edges. What is a back edge? So now we are ready to define what a back edge is. Anyone, what is a back edge? An edge [student: from a node to another] from a node to an ancestor. What is an ancestor? An ancestor with respect to the depth first search with respect to the DFS tree. The notion of an ancestor is coming in only because we have defined DFS tree.

An edge from a node to an ancestor is called a back edge but not to… [Student: but not to parent] An edge from a node to a parent is a tree edge. So we will distinguish between tree and back edges. An ancestor is not a parent let's say. Those are the back edges and these edges, the once in the red dotted here are not back edges because they are not going from a node to its ancestor. Neither is this node an ancestor of this nor is this an ancestor of this. So such edges cannot be there at all in our graph.

Depth first search basically means, so after you do a depth first search you end up dividing now the set of edges into two classes tree edges and back edges and there is no other edge. [Student: we can just say that any edge] Every edge gets classified either as a tree edge or as a back edge. Let me write this down. So DFS classifies every edge as a tree or a back edge. This is similar to breadth first search means breadth first search classifies every edge as an edge going between adjacent levels or going within the same level, let's see.

(Refer Slide Time: 36.49)



So here depth first search also does this thing for us. Clear to everyone? We are still talking about depth first search in undirected graphs. When we come to directed graphs things will change a bit, keep that in mind. How do we implement depth first search? Looks like a fairly complicated thing. Stacks or recursion let's see. What are the things we needed? We said we have to keep track of whether a vertex is visited or it's not visited. Whether what the color of a vertex is. So actually we don't even need to distinguish between grey and black.

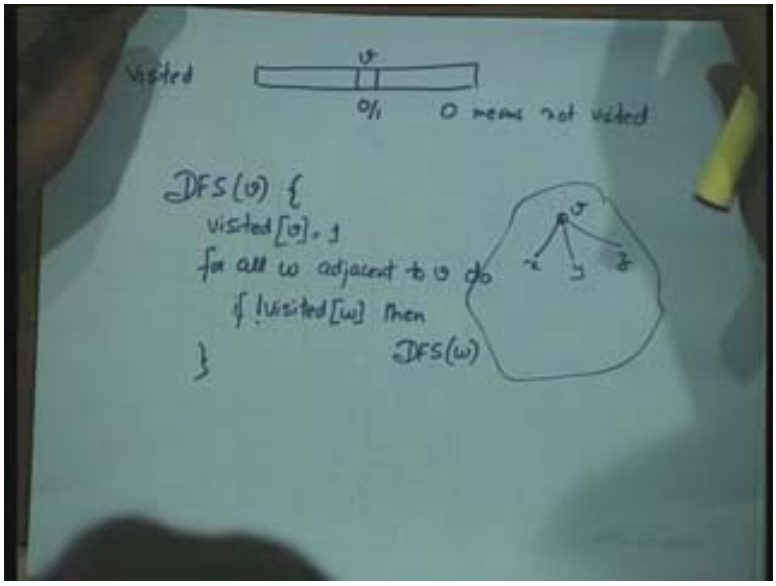
We just need to distinguish between white and non-white, whether a vertex is visited or it was not visited. That was the only thing we really needed. We are going to keep an array called visited. So what will this have? Basically it will have an entry for every vertex. For a vertex v it will have an entry let's say 0 or 1, 0 if it is not visited and 1 if it is visited. This is a zero one entry, zero means not visited. Initially all the entries of this array would be a zero. [Hindi]

Suppose I wanted to do a depth first search starting from a vertex v in my graph. So now my graph is going to become more abstract. This is my graph, this is a vertex v, I want to do a depth first search from this vertex. So what am I going to do? What does depth first search involve, what is the first thing I should do? First I should mark this vertex as visited. Clearly let's say this is the very first thing I do, visited v equals one. Now I have to look at all the vertices one after the other, adjacent vertices. So let's say this had 3 adjacent vertices x, y, z.

Let's put down a loop for all w adjacent to v, do something. For all vertices w, so w is just a running variable so to say which will take the value x, y or z depending upon which this is. What should I do? [Student: doffs dfs dfs w] I just do visited dfs w right away. No. [Student:] If it is not visited. If visited w [student: equal to one] equal to zero, if not of visited w then [student: dfs] just say DFS (w). [Hindi] else [student: no else how can you define] no else then what? [Student: the else w will backtrack] Backtrack. [Student: else backtrack if for all w] for all w [Hindi] [Student: not of visited of w then we backtrack visited w equal to one] Visited w equal to one [Hindi] (Refer Slide Time: 41:15).

If it has now then [student: sir predecessor] backtrack [student: [Hindi]. if all w is not visited then we backtrack to the predecessor sir after we have done]. Basically [Hindi] but this takes care of everything for us, all our backtracking everything. It is not trivial to understand this part. Why it is taking care of all the things for us? [Hindi] For all [student: adjacent vertices] [Hindi] all adjacent vertices [Hindi] but to convenience yourself this is doing all of that. You know recursion is not magic right it's after all just a piece of code. [Hindi] It is particularly well suited for depth first search. You can write three line program, four line program for something we spent 30 minutes telling what the procedure is.

(Refer Slide Time: 43.19)



What is that is happening? Why is this working out for us? So let's try and understand that and I will just… I think I will just explain it on this picture. I started a depth first search from v. This is what my depth first search v was. Let's say the vertices were considered in this order, exactly this order x, y, z. visited v is one, visited x y z are all zero, as are visited of all the other vertices I just started my depth first search here. Then I came to this vertex. Since visited of x is zero, I launched a depth first search here. That's what we would have done here.

I launched depth first search on this. What is this depth first search going to do? Let's say our depth first search visits a certain bunch of vertices which are not already visited and marks them visited one. As a consequence this guy is going to visit a bunch of vertices and set visited one for each one of them, visited at one. And then it is going to terminate, every program has to terminate. It also terminates but what is it that terminates? DFS on x. When DFS on x terminates, this recursive call terminates. Where do we end up? When this terminates we end up in the DFS of v because this is the DFS which called DFS (x).
The picture is something like the following. You had DFS (v), making a call to DFS (x). This did a lot of recursive calls but at some point it terminated. After it terminated we made a call to DFS (y). When you made a call to DFS (y), why did we make a call to DFS (y)? Because we are looking at all the adjacent vertices and I am assuming for now that y was not visited in this DFS.

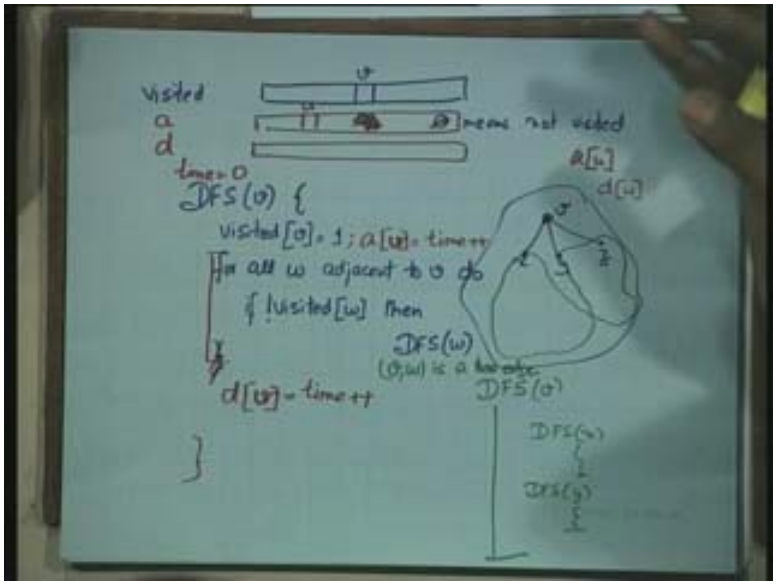
Suppose it was not visited. So it was still at zero. I made a call to DFS y now. As a consequence, it visited another bunch of vertices. it would not have visited any vertex which was already visited by x. We are ensuring that not going to a vertex whose visited is already set to a one. So it visited another bunch of vertices and then it terminated.
Now I am going to go to vertex z and try to launch a DFS there but I see that z is already visited. Because z was set to visited, when I did my DFS (y). When that happens, z is already visited so I don't launch a DFS here. I have taken care of all adjacent vertices and

so this terminates now. This terminates means the DFS on v terminates, this whole process terminates. The claim is I would have visited all the vertices that were to be visited. From this vertex which are the vertices I can reach. If I reach a certain vertex, either I can reach it through x or through y or through z. Because after all these are the only three edges incident at this vertex. If I can reach it from x then that means it should have been visited in DFS (x). If I can reach it from y, it should have been reachable from DFS (y). If I can reach it from z, it should also have been reachable from DFS (y). Why? Because z is reachable from y. If z is reachable from y then after I had reached z, I would have continued and visited all vertices which can be reached from z. So anything that is reachable from v therefore is visited and so our DFS from v should terminate and that is exactly what is being done here.

We are not done with this yet as you can imagine. I want to add my timestamps. How should I modify this procedure? [Student: starts] Suppose with each vertex I want not juts visited but I have two other arrays arrival. Let's call it a and departure let's call it d. So a of v so this is also an array and this is also an array. It is not a zero one array sorry, so this zero one was for the previous thing. If I have a vertex u, so a of u will be an integer which will tell me what time I reached vertex u. And d of u would be another integer which will tell me at what time I left vertex u. What modification should I make to this piece of code? [Student: when d of u] [Student: a of u comes when you start DFS v after visited]. So a of u equals [student: plus plus b plus whatever] time plus plus [student: and after DFS not after DFS w after] [Hindi] [student: after the for loop] [Hindi].

It's just saying stamp and increment so that the next guy doesn't get the same stamp. It could be plus plus time or time plus plus, doesn't make too much of difference. It will just change the starting. Yes, no? So and time could initially be zero [Hindi]. You can also modify this procedure to identify which edge is a tree edge and which edge is a back edge. Can you do that? [Student: if visited w] [Student: for DFS w we can mark its edge]. Suppose I also wanted this information. Every edge which is a tree edge, I want to mark the tree edge. [Student: with is equal to one then] when I am ready to launch DFS w then that means…. What does that mean, what can I conclude at this point? [Student: noise] which edge? v w [student: v], the edge v w is a tree edge that I can conclude. [Hindi] I can write that statement [Hindi].

(Refer Slide Time: 52.48)



v w is a tree edge. If you have identified what the tree edges are then its equivalent to identifying what the back edges are, anything which is not a tree edge is a back edge. I am going to stop here today. In the next class we are going to see to analyze the running time of this procedure and we are going to spend a couple of classes in applications of depth first search.