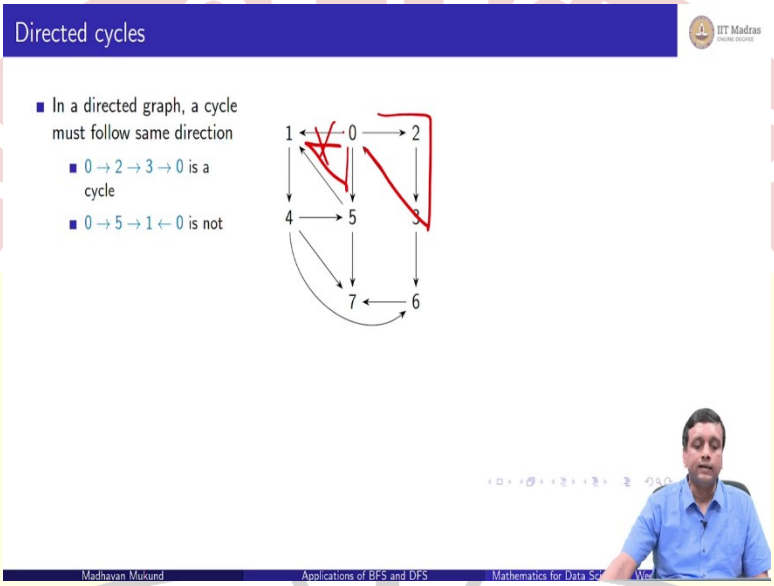
IIT Madras

ONLINE DEGREE

Mathematics for Data Science 1
Professor. Madhavan Mukund
Department of Computer Science
Chennai Mathematical Institute
Lecture No. 65
Applications of BFS and DFS-2

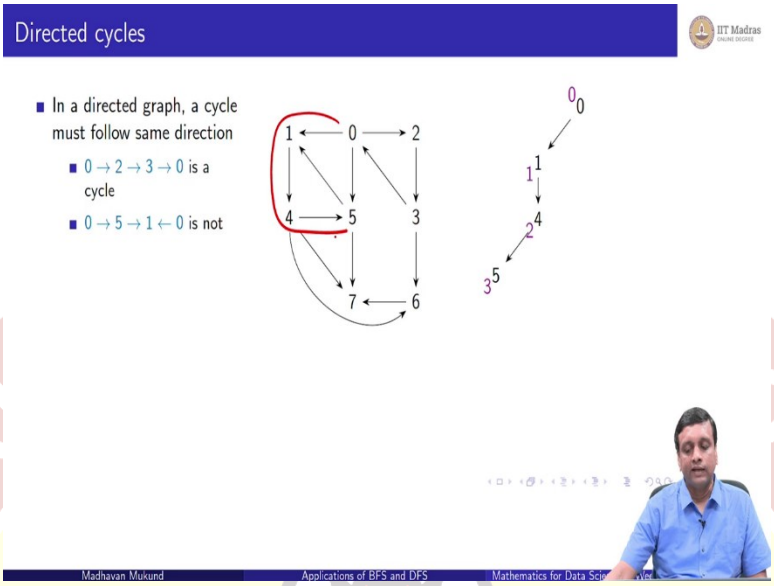
(Refer Slide Time: 0:4)

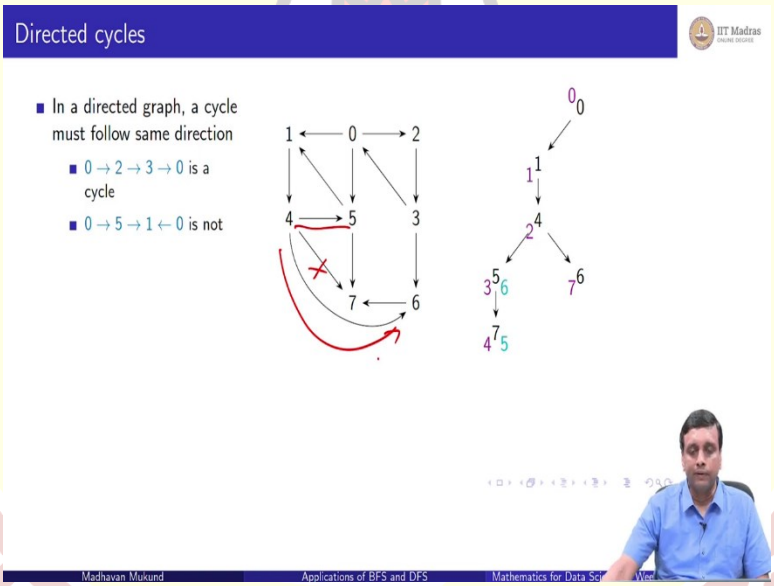


So, now what happens in a directed graph? So, in directed graph a cycle also is directed, that is I must go around from a vertex through a set of neighbors and come back, but following the same direction. So, it is like going around a circle in a one-way street, you have to follow the one-way street, you cannot suddenly go down in one-way street the wrong way. So for instance, I can go from 0 to 2 then I can go from 2 to 3, and then I can come back from 3 to 0. So, this is the cycle in the directed sense because I am going forwards at every step.

On the other hand, if I try to go from 0 to 5, and then from 5 to 1, and then I try to come back from 1 to 0, this is not allowed because 1 to 0 is not in the same sense. So, if I ignore the direction there is a cycle 0 5 1 0, but with directions there is no such cycle, so we are interested in directed cycles.

(Refer Slide Time: 1:08)





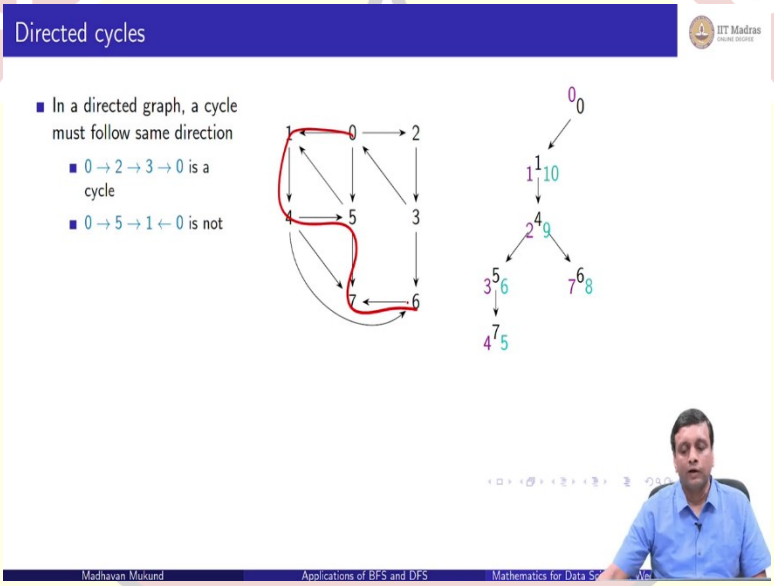
So, again we again do a DFS and do the DFS numbering, it is exactly the same, there is no difference in DFS whether it is directed or undirected, so we follow the same protocol for these pre and post numbers, so we start at 0 in this case. We are starting at 0 and then we are going to systematically explore its neighbors. So, we enter 0 with pre number 0, we enter its first neighbor with pre number 1 and now we explore 1.

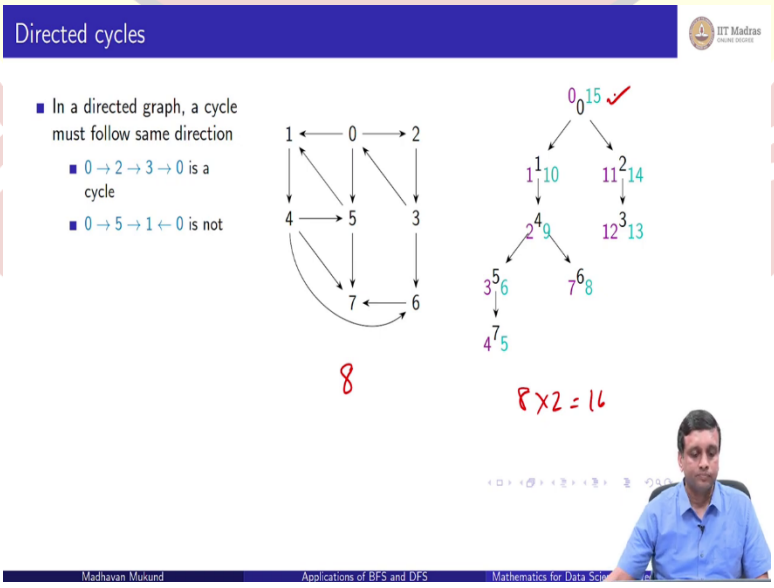
So, we enter 4 with pre number 2, from 4 we have many ways to go, so the first way is 5, so we enter 5, so we have now come down this way. So, we have gone from here and we have reached here. So, we enter 5 with pre number 3, from 5 we can go to 7 with pre number 4. And now notice

that 7 has no outgoing vertices, edges at all. So, from 7 I cannot go anywhere because it has no outgoing edges, so from 7 I exit with number 5. So, I have a post number 5 and I come back to 5.

Now I ask whether we can do anything from 5, well 5 had only two outgoing edges, back to 1 and forward to 7. But 1 was already covered and 7 has just been covered, so 5 also exits with number 6. Now I come back to 4, so 4 I explored 5, 7 is no longer available because 7 is already done through 5, but this long edge from 4 to 6 is there, so I enter 6 with pre number 7. Again from 6 the only thing I could have done is go to 7 which I have already seen, so I exit 6 with number 8.

(Refer Slide Time: 2:36)



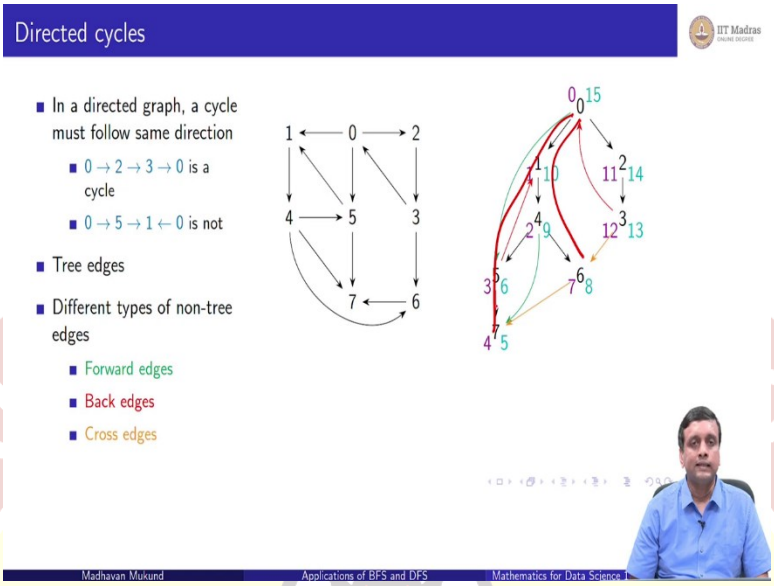


Now I am done with 4, so I exit 4 with number 9. Now I am also done with 1, so I exit 1 with number 10, because 1 had only one outgoing edge to 4. So, I come back to 0, so I in some sense I started here I went here, then in the process I explored everything on this side. So, now I go to the right side, so I explore the other neighbor of 0 which is 2 by entering it with number 11 because number 11 because I exited last one with number 10.

From 2 I can go to 3 with entry number 12, from 3 I cannot go anywhere else, I can only go to 6 or 0 which are both already visited, so I exit 3 with number 13. I go back and exit 2 with number 14, and I go back and I exit 0 with number 15. So, you should just do the sanity check, there are 8 vertices here, so there should be 8 into 2 is equal to 16 numbers.

Every time I enter a vertex and I exit a vertex, so if I start at 0, I must end with 15 and I indeed do end with 15. If you do not end with 15 then there is a problem. So, if you start with n, vertex with n graph, graph with n vertices and you do a DFS, then at the end the last exit number should be 2n minus 1.

(Refer Slide Time: 3:46)



So, here we have tree edges which is what we have drawn when we were drawing these, so all these edges which we followed when we did the DFS are the tree edges. So, this is now a directed tree but otherwise is the same structure as before, so it is tree which connects all the vertices that we visited. But the non-tree edges now come in different flavors.

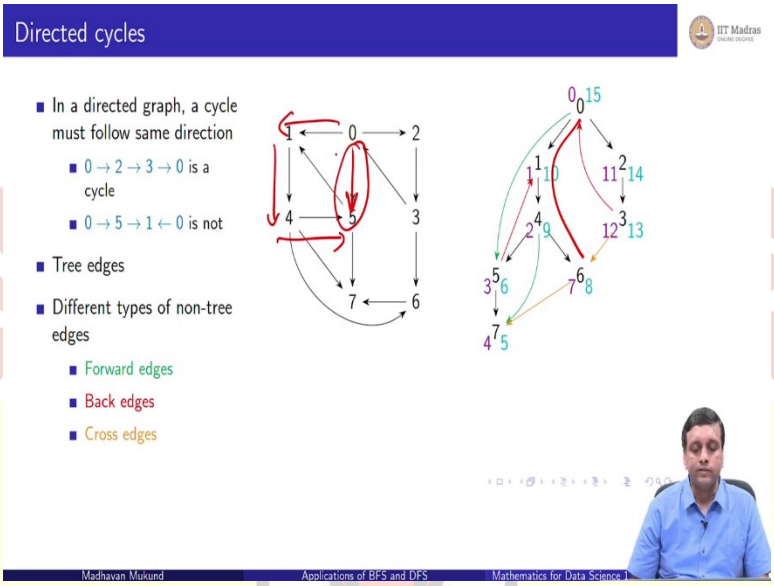
So, one type of non-tree edge is one which follows the direction of the tree, so it goes from a higher node in the tree to a lower node in the tree. So, the non-tree edge is in the same direction as the path that it is by passing in some sense you are kind of short circuiting, you are going like a flyover on a road, you are going over some intersections and reaching a later point. So, 0 to 3 is a forward edge, so is 4 to 7.

So these are not part of the tree, but they traverse the tree in the same direction as the edges that they are skipping. The converse would be a backward edge, it goes up a path in the tree. So it goes from a lower node in the tree to a higher node but again along a path which already exists. So, I am going from 5 back to 1, and there is a path from 1 to 4 to 5 or I am going from 3 back to 0, and there is a path from 0 to 2 to 3.

And finally there could be edges which cut across different branches of the tree, for instance, I can go from 6 to 7, so 6 is on this branch and 7 is on this branch, so it is not that I am going up or down from 6 to 7, I am going across. So, these are called cross edges. So, these are the 3 types of edges

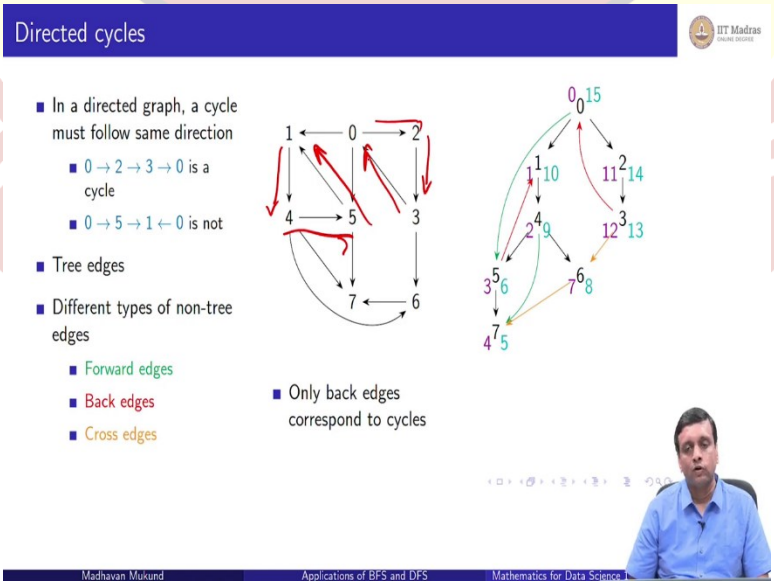
that you could have in a directed graph which are not in the DFS tree; forward edges, back edges and cross edges.

(Refer Slide Time: 5:28)



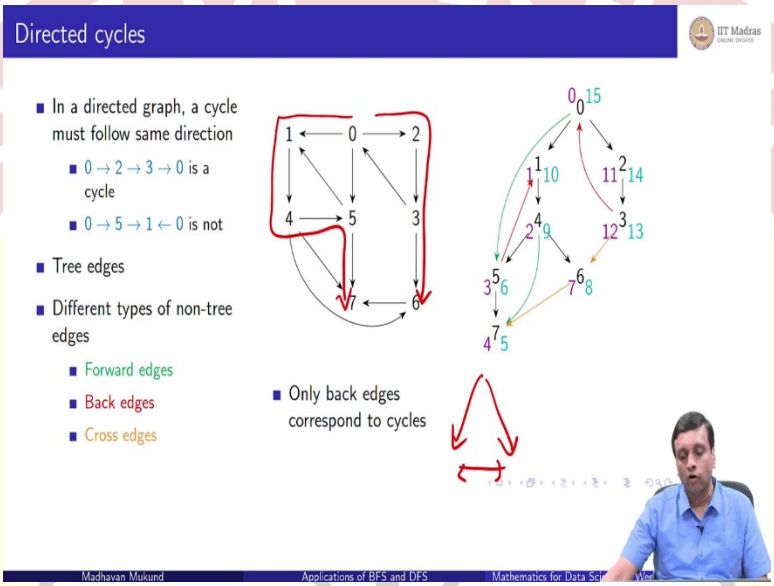
So, now if you look at this carefully, so let us look at this forward edge 0 to 5. So 0 to 5 is a forward edge and the other edges that it was corresponding to are 0 to 1, 1 to 4 and 4 to 5. So, this was the new edge that I added. So, clearly adding this new edge to the existing path did not create a cycle, because it is going in the wrong direction. So, a forward edge does not create a cycle.

(Refer Slide Time: 5:59)



On the other hand, a backward edge does create a cycle. So, if I go from 1 to 4, and then I find that there is a backward edge from 5 to 1, then there is a cycle. Similarly, if I go from 0 to 3 via 2 and then I find that there is a backward edge from 3 to 0, then there is a cycle. So, forward edges do not create cycle, backward edges do create cycles and you can also check that cross edges will not create a cycle.

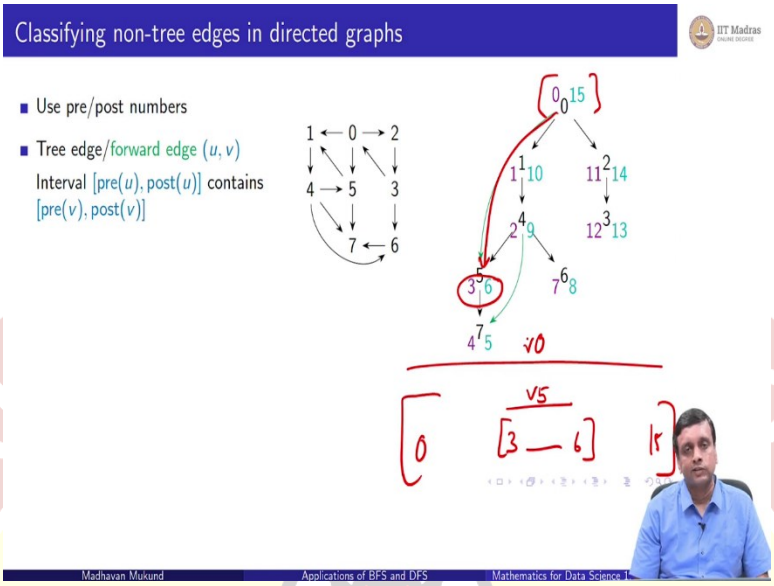
(Refer Slide Time: 6:26)



So, cross edges will actually go down different, different paths and so therefore they do not create a cycle. So, for instance, if I look at this here the cross edge from 6 to 7, so I have a 0 1 4 5 7 path, so I have this path and I have 0 2 6 path. And now these two paths are going in opposite direction, so no matter how I connected this way or that way, so I have a path like this and I have a path like this, so there is no way that I can connect these two paths either left to right or right to left, form a cycle, because the paths themselves are going in the wrong direction.

So, what we want to do now is to identify not just the non-tree edges, so in the undirected case it was very simple, every non-tree edge indicated a cycle. Now in the directed case we are saying it little more subtle than that, it is not enough to just to find a non-tree edge, you must find a non-tree back edge. So, how do we know which of the edges which are not in my DFS tree are forward edges, which are back edges and which are cross edges.

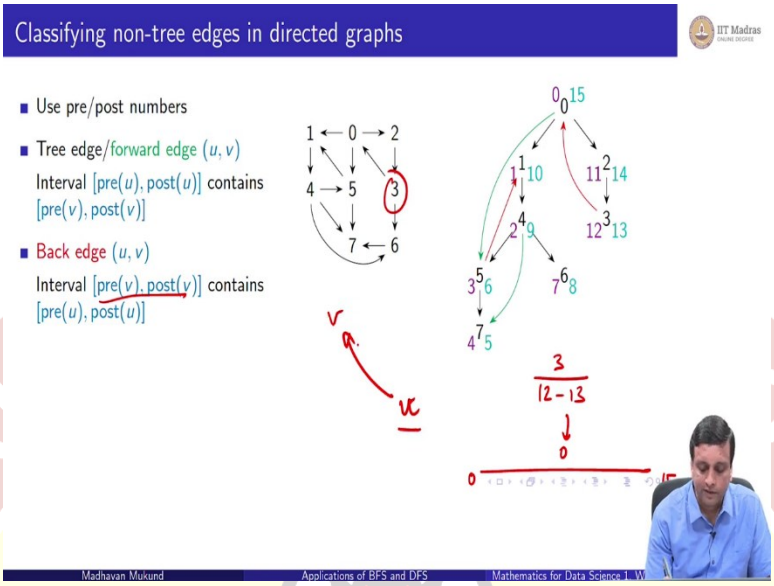
(Refer Slide Time: 7:31)



So, the problem is that of classifying these non-tree edges. And this is the first instance where we will actually use these pre and post numbers. So, if I have a forward edge from u to v, so in this case I have this forward edge here, from 0 to 5, then we will look at the interval, I say that I entered 5 at 3 and I left at 6. So, I have an interval 3 to 6, during this period I was processing 5, I was going to neighbors of 5 and so on. And when I finished processing 5 I was at 6.

And look at 0, 0 has an interval which is from 0 to 15. So, I started processing 0 when the counter was 0 and I finished it when it was 15, so what this says is that the entire processing of vertex 5 happened before I finished vertex 0. So, vertex 5 was processed as a part of vertex 0 processing, so if now the back edge, if the edge goes from the bigger interval to the smaller interval, then it means it is a forward edge, because it is a vertex which was processed earlier to a vertex which was processed later, because the interval is smaller.

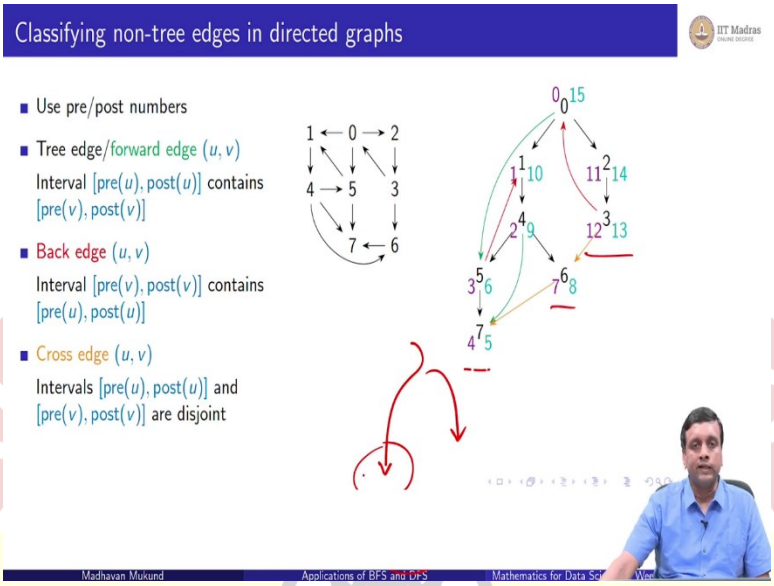
(Refer Slide Time: 8:42)



On the other hand, if I have a back edge, then it is exactly the reverse. So, I am going from say 3 which has an interval 12 to 13, and I am going back to 0 which has its interval 0 to 15. So since I am going backwards, again this indicates that I did the processing of 3 while I was doing the process of 0. So, therefore, this is an edge back from 3 to 0, 3 happened later, so now because the edge is reversed, it is asking that the starting interval is included in the ending interval or it rather the ending interval is included in the starting.

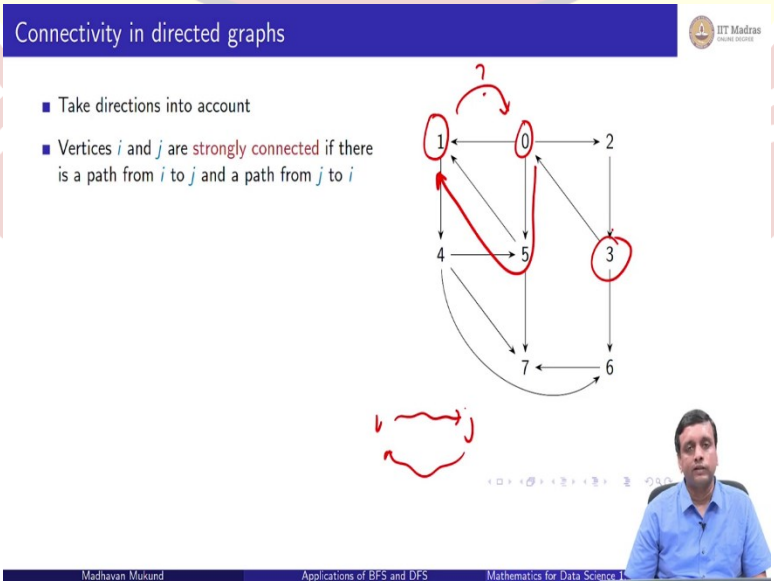
So the, if I am going back from v to u, from u to v, so then I want that this interval is smaller than this interval. So, the ending interval is bigger than the starting interval.

(Refer Slide Time: 9:37)



And finally if I have a cross edge, you will see that these are actually disjoint, because they are happening on different things, so I finished processing one path, so I came down this thing I finished it and then I went back and started here. So, when I started on the right hand side path, I had finished the left hand side path, so all these numbers are exhausted. So, basically the out has happened before the in there, so the two intervals will be disjoint. So, this is how we can use this pre and post numbers with the vertices to discover which of the non-tree edges are back edges and therefore, we can decide whether or not are directed graph actually has a cycle.

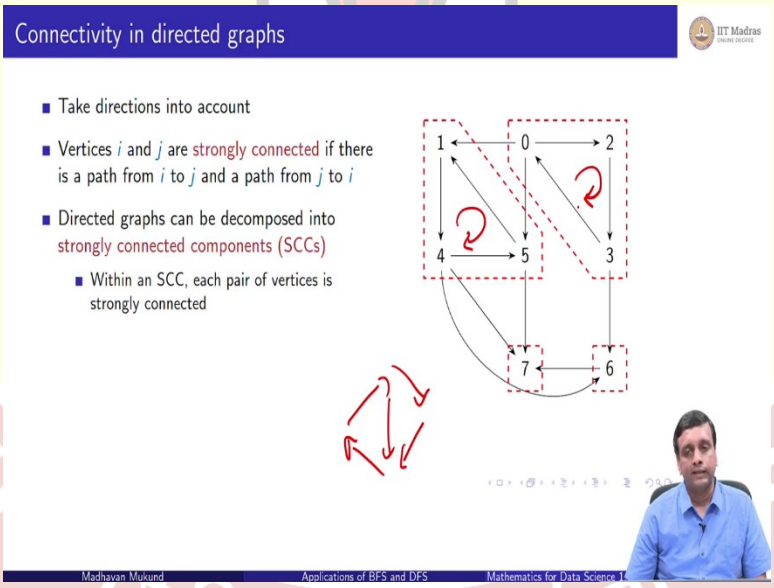
(Refer Slide Time: 10:09)



So, just like cycles have to take directions into account, so does the notion of connectivity. So, we said that in an undirected graph, we said a graph is connected if every vertex can be reached from every other vertex. And now in a directed graph we have to ask whether I can go following the directions. So, I say that a pair of vertices i and j are strongly connected if I go from i I can go to j and then I can come back from j to i by a different path. So, in this case I say i and j are strongly connected, if they were not strongly connected, it is possible I can go from i to j but I cannot come back.

So for instance, if I look at, say for instance 0 and 1 in this case, I can go from 0 to 1 by following this path, but there is no way to go from 1 to 0. Because I cannot go from 1 except 4 and I cannot, basically the only way to come back to 0 in this graph is to come by a 3, because that is the only incoming edge to 0 and there I cannot reach from 1 to 3 by any path. So, 0 and 1 are connected in one direction but not connected backwards and therefore they are not strongly connected.

(Refer Slide Time: 11:25)

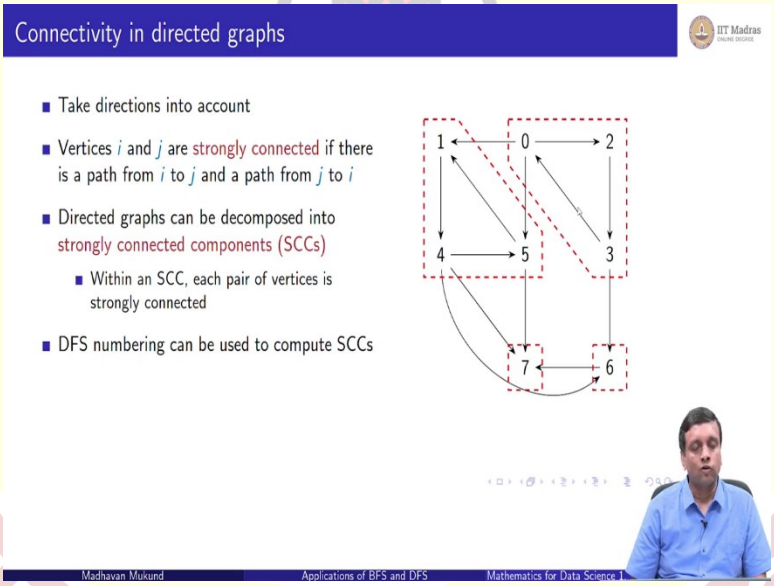


So, therefore, a correct notion of component that we need for a directed graph is one where not just that every pair of vertices is connected but every pair of vertices in that component is strongly connected. I can go from everywhere to everywhere and come back, so I can go anywhere in that component without worrying about where I am starting. So, this is what is called an SCC or a Strongly Connected Component.

So, you can see that for 3 vertices, like we have in this a 3 vertex strongly connected component is just a cycle. So, basically if I have a, if I have a directed cycle, if I have something like this, then this will be a strongly connected component. But I could have more edges, I could have something like this and so on. It does not matter if there are more edges, but there should be a minimum number of edges so that I can go from anywhere to anywhere in both directions.

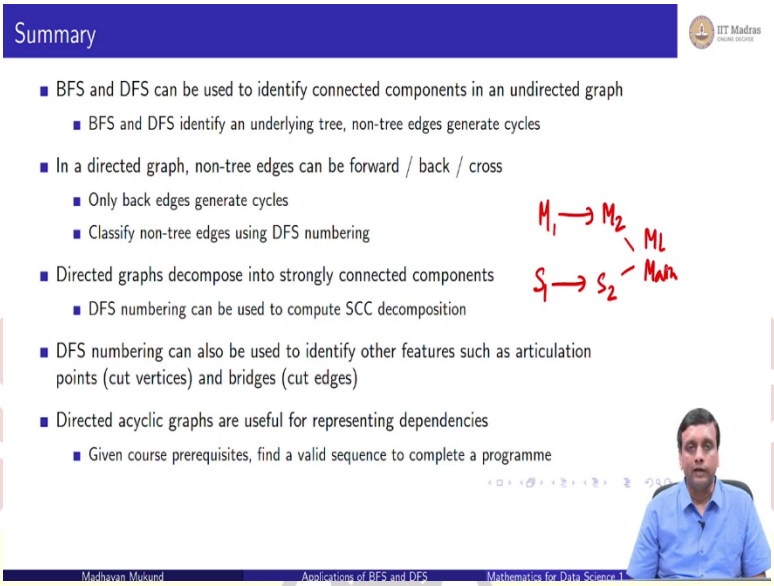
So, in this particular graph 1 4 5 forms a cycle because I can go around this in this direction and reach anywhere from anywhere. Similarly, 0 2 3 forms a cycle, but 7 and 6 now are stuck on their own because if I leave 6 I cannot come back to 6 from any of these paths, if I leave 7 I cannot come back to 7, I cannot leave 7 at all in fact because 7 has no outgoing edges. So therefore the strongly connected components in this graph are the ones which I marked in red.

(Refer Slide Time: 12:40)



So, what we are not going to cover in this particular course but maybe at a later stage is that this DFS numbering that we just did can also be used to compute these strongly connected components. So, we saw that it can be used to compute back edges and detect cycles, but it can also be used to detect strongly connected components.

(Refer Slide Time: 12:58)



So to summarize, we saw that BFS and DFS are primary strategies for reachability in a graph, but what we have seen now is that we can do much more with BFS and DFS. So, the first thing we can do is identify the connected components in an undirected graph. So, by doing BFS or DFS we first uncover a tree, so we identify some edges in the graph which we process during the BFS and DFS, this form a tree. And any edge which is outside this tree must form a cycle with the edges in the tree, so any time there are non-tree edges in the graph after I finish my BFS or DFS, we can generate, we can say that there is definitely a cycle in the graph.

However, for directed graph we saw that this is little bit more complicated, so we have 3 types of non-tree edges; forward edges, back edges and cross edges. And of these only the back edges generate cycles, so this is one instance where we used this DFS numbering this pre and post numbering for vertices in order to detect which of these non-tree edges are back edges.

And finally, we described the notion of a strongly connected component, and although we did not actually calculate them, we claim that the DFS numbering that we have done can also be used to identify the strongly connected components. So, a strongly connected component, remember has one where every pair of vertices is reachable from each other. So, we mentioned last time and we will not elaborate on this, but DFS numbering can be used for many other things, so one thing it can do is identify the so called critical vertices.

So, cut vertex or an articulation point is one which if I destroy it, it will disconnect the graphs. So, if I remove it from the graph, the remaining graph falls apart, so this is critical for instance if you are looking at a network of say a power network, if there is a power station which is relaying power and if it goes out of action and the power network now disconnects, so two parts do not get power from each other, then that is something that we have to be extra careful to protect.

So, these are also important things to discover in your graph. Similarly, there could be cut edges, if I cut a connection between two nodes, then the graph falls apart, so these are called bridges. So, articulation points and bridges can also be identified in a graph using this notion of DFS numbering. And finally, this idea that we are looking for cycles in a graph is particularly important in the directed sense, so there is a very important class of graphs which we will look at next week, which is called directed acyclic graphs.

So, a directed acyclic graph as it suggests is a directed graph which has no cycles in it. Now a directed acyclic graph is very often used to represent these kind of prerequisites or preconditions or dependencies. So, supposing I want to describe for instance the course contents of this program and I say that you have to do maths 1 before you do maths 2, and if I have to do stats 1 before you do stats 2 and maybe there is no correlation between maths 1 and stats 1, so you can do them in any order and you can postpone one to the other, but you cannot do maths 2 before maths 1.

So those are, so I have M_1 and I have an edge saying M_1 must be before M_2 and I have one saying that S_1 must be before S_2. And now maybe I have something which says that in the third semester there is an math for ML in which I must have completed both M_2 and S_2. And now I ask in what order you can do these things? So, clearly you can do math for ML only after you finish all these 4 courses, but these 4 courses you can be a little flexible, you can do S_1 for instance, after M2 or you can do M_1 after S_1.

So, these kind of properties about which sequences can be compatible with a set of dependencies are used very often and this can be done by analyzing this directed acyclic graphs which we will do later on.