INDIAN INSTITUTE OF TECHNOLOGY MADRAS

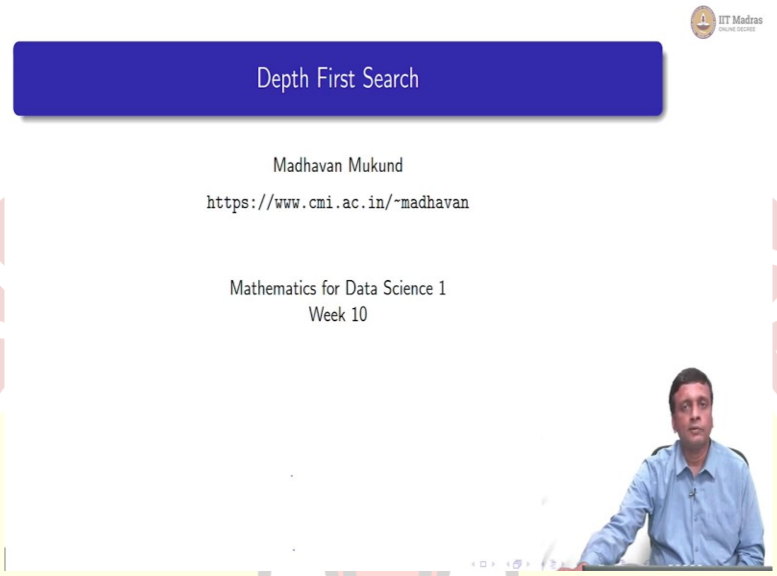
सिद्धिर्भवति कर्मजा

IIT Madras

ONLINE DEGREE

Mathematics for Data Science 1
Professor Madhavan Mukund
Depth first search

(Refer Slide Time: 0:14)



Depth First Search

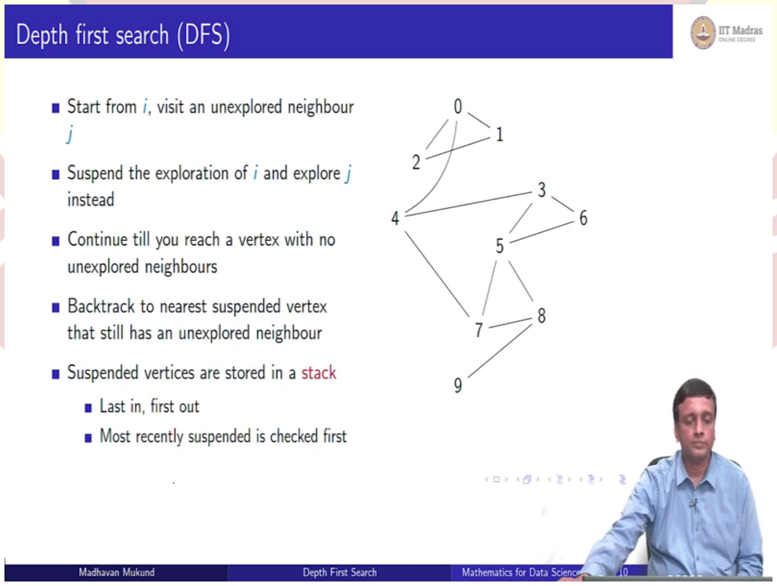
Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Mathematics for Data Science 1
Week 10

So, we had said that there are two systematic ways to explore a graph and we earlier looked at breadth first search which explore a graph level by level using a queue to maintain the unexplored vertices as we go along. The other strategy that I would use depth first search.

(Refer Slide Time: 0:31)



Depth first search (DFS)

- Start from i, visit an unexplored neighbour j
- Suspend the exploration of i and explore j instead
- Continue till you reach a vertex with no unexplored neighbours
- Backtrack to nearest suspended vertex that still has an unexplored neighbour
- Suspended vertices are stored in a stack
  - Last in, first out
  - Most recently suspended is checked first

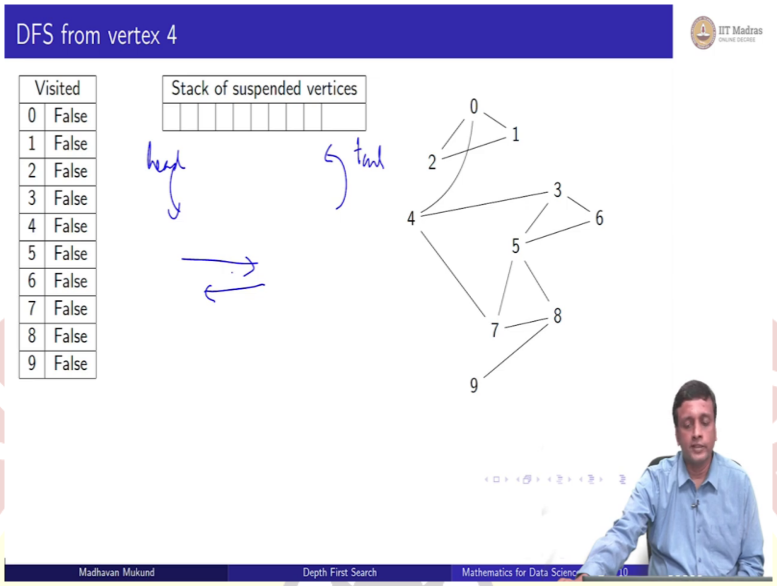
So, in depth first search we start from a vertex i and we pick anyone of its neighbor is not been explored in breadth first search. We pick all its neighbors and put them into a queue, here we pick any unexplored neighbor j and now what we do is we basically suspend the exploration of i and start exploring j instead. So then we look at j, we look at the neighbor of j, we pick again some neighbor of j which has not been explored.

We suspend j and go to the next vertex and we keep doing this until we run out of vertices that we can reach down this path. And now when I reach a vertex through this process which I cannot explore any further, I come back along the path I have taken. And find the first point where there was another choice for an unexplored. So you back trap to the nearest suspended vertex that still has an unexplored neighbor.

And then you explore that neighbor and so on. So here unlike in breadth first search where we had to keep track of these vertices which have been visited level by level and then put them into a queue and then we process this queue in this first in first out manner. So the queue, the vertices which are added level 1 get processed before the ones at level 2 and they get process before the ones at level 3 and so on.

In a depth first search if I have walked down some distance then I will come back to the point where I last stopped and see there was something else I could do from there and then I will walk back and so on. So I need not a first in first out but what is called a last in first out. So the last vertex has suspended is the first that I restore and check again. So this is called a stack, so these are 2 very fundamental ways of organizing sequences. A queue is a first in first out structure and that is used in breadth first search and a stack as we will see is used in this depth first search.

(Refer Slide Time: 2:24)



DFS from vertex 4

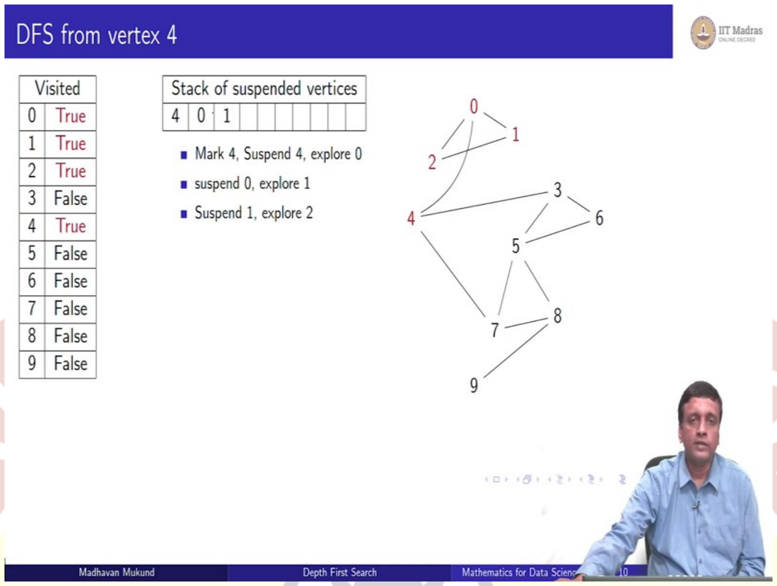
Visited	
0	False
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices

So, let us try to explore this depth first search, so for this our stack will again like our queue grow from left to right, our stack will again grow from left to right. So when I add this thing to the stack I will put it to the right. And now unlike a queue when I remove things from stack I will remove it from the right not from the left. Remember in a queue we had a head here and we had tail here.

So, we put things here and we brought them out here instead here I am going to grow the stack this way and also remove it from this end. So, that is going to be our strategies so for a change instead of 7 let suppose we start our depth first search from vertex 4.

(Refer Slide Time: 3:04)



DFS from vertex 4

Visited	
0	True
1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False
9	False

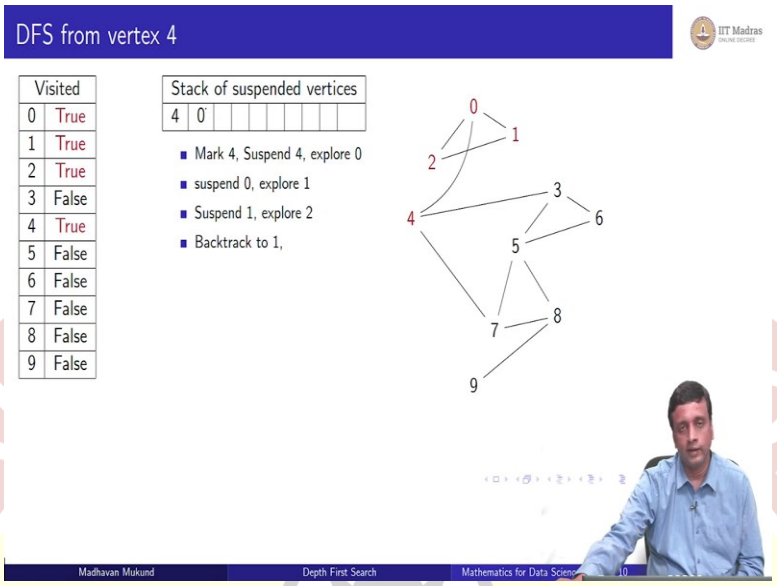
Stack of suspended vertices										
4	0	1								

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2

So, we first start as before by marking 4 as visited. And now we have to pick a neighbour of 4 and suspend 4 and start exploring that neighbour instead. So the neighbour of 4 has 0, 3 and 7 so we can pick anyone of them so let us pick the smallest one. So I suspend 4 and I start exploring 0 instead. So now I look at the neighbours of 0 and explore them if they have not been visited.

So notice that now visited this 2 for 4 and 4 and the stack has 4 in it but now I am going to suspend 0 and pick up one of its unexplored neighbour say one. So now I have marked 1 and 4. And so the stack is growing from left to right remember. So 0 has come on top of 4 in the stack. So now I have to explore 1, so explore 1 and I find it has only 1 neighbour which is unexplored namely 2. So I suspend 1 and I start exploring 2. Now when I look at 2, 2 has no neighbuors left to explore because it has only 2 neighbours 0 and 1 both have which I have already been visited.

(Refer Slide Time: 4:13)



DFS from vertex 4

Visited	
0	True
1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices

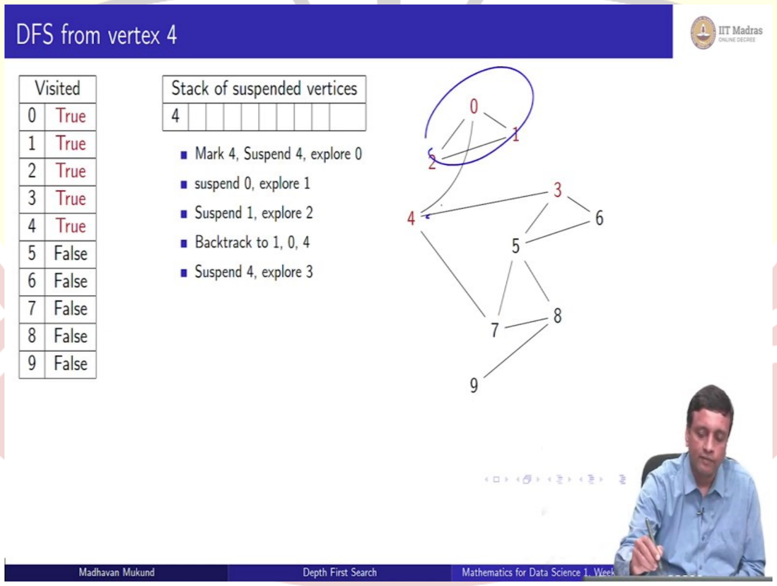
4	0								

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1,

Madhavan Mukund Depth First Search Mathematics for Data Science

So, I start moving back. So I back track from 2 to the most recently visited one which is 1 and see whether 1 has anything more to be done. But 1 has nothing more to be done because 1 had only 2 neighbours I have visited both of them.

(Refer Slide Time: 4:26)



DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	False
6	False
7	False
8	False
9	False

Stack of suspended vertices

4									

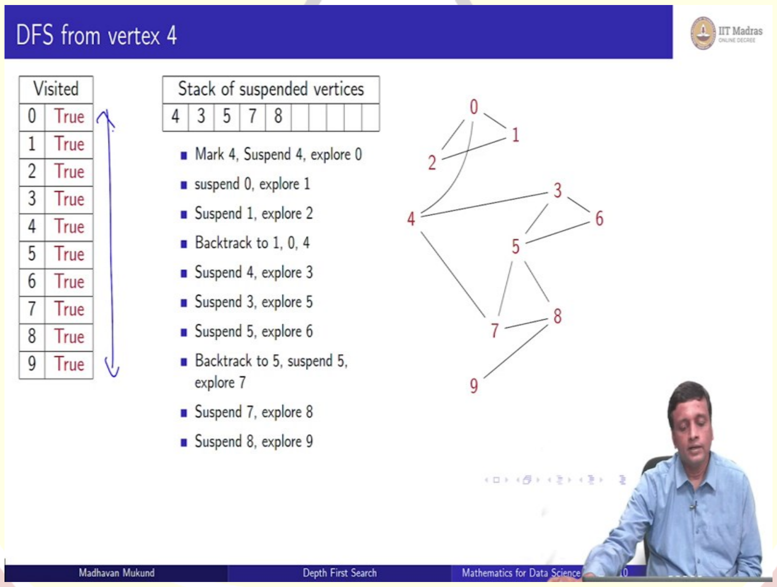
- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3

Madhavan Mukund Depth First Search Mathematics for Data Science 1, Week

So, I back track to the previous one which is 0. So I ask whether 0 has anything more to be done. So notice that from 0 I have visited 1 and from 1 I visited 2. So though when I started with 0 I had not seen 2 yet by that time I have come back to 0 through back tracking 2 has already been visited. So at the current state of 0 it has no unvisited neighbours. So 4 was visited before it came to 0.

1 was visited from 0 and 2 was indirectly visited from 1 and therefore is no longer available. So I have to back track from 0 as well. So I back track from 0 back to 4 and now I ask whether 4 has anything interesting to say. And 4 says yes I have an another vertex to visit which is 3. So I explore this 3 and I suspend the 4. So now I have finish this whole section and I have come here.

(Refer Slide Time: 5:21)



DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices

4	3	5	7	8					

- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9

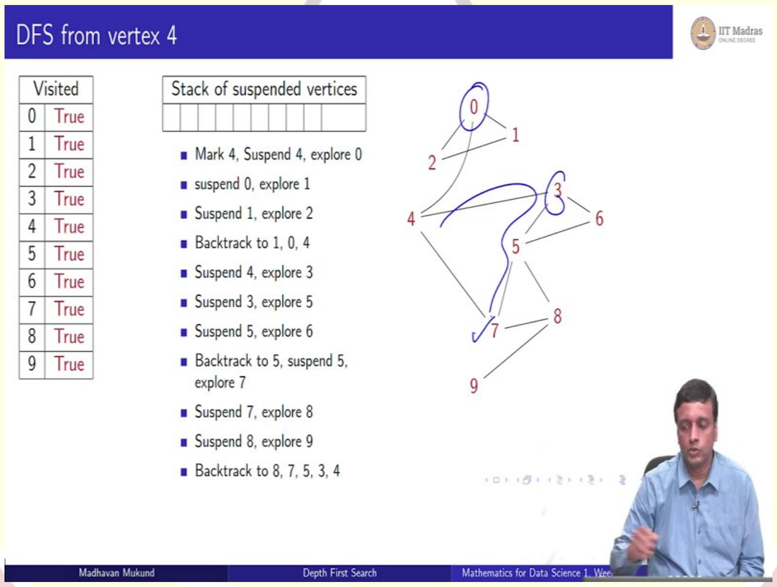
Madhavan Mukund Depth First Search Mathematics for Data Science

So now from 3 I have two neighbours 5 and 6. And I saw a suspend 3 and maybe I explore 5. And then from 5 I have I mean 2 unexplored neighbour 6 and 7. So maybe I suspend 5 and I explore 6. So now when I come to this point I have 4 which triggered 3 which triggered 5 which triggered 6. So I have 4, 3, and 5 on the stack and 6 has no new neighbours to explore because both 3 and 5 are already visited.

So, I will back track from 6 to the previous one which is 5 and ask whether 5 has anything more to do. Well, 5 does it has 7 and 8. So I will perhaps pick 7, so I will again suspend 5 this is the new suspension of 5 first time I suspended it because I have explored 6 when I came back to 5 and I have suspended it again to explore 7. 7 has a neighbour 8. So I will suspend 7 and explore 8. 8 has a neighbour 9.

So, I will suspend 8 and explore 9 and at this point if you look at the visited matrix everything this list, everything here has been marked true. But I still have this long queue a long stack of things which I have suspended. So I have to make sure nothings is missed out. So I will now look at 9 and 7, 9 has nothing left explore because it only had 1 neighbour 8 from which it came.

(Refer Slide Time: 6:47)



DFS from vertex 4

Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Stack of suspended vertices

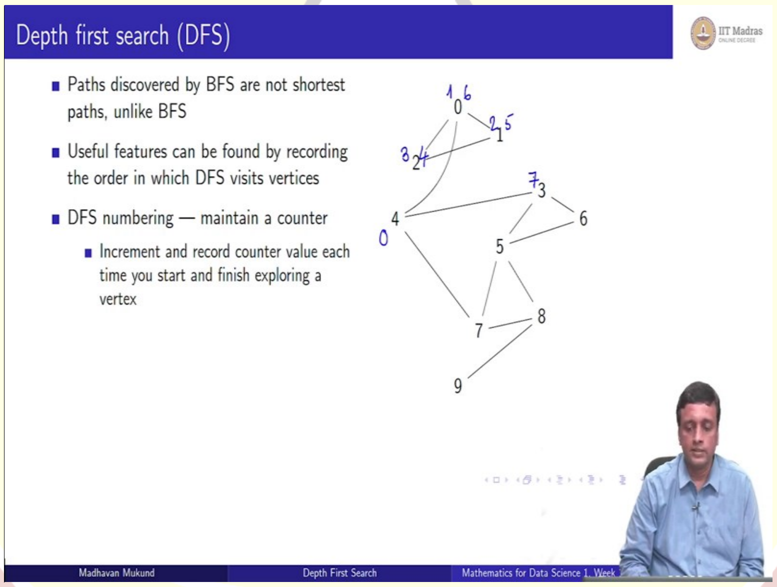
- Mark 4, Suspend 4, explore 0
- suspend 0, explore 1
- Suspend 1, explore 2
- Backtrack to 1, 0, 4
- Suspend 4, explore 3
- Suspend 3, explore 5
- Suspend 5, explore 6
- Backtrack to 5, suspend 5, explore 7
- Suspend 7, explore 8
- Suspend 8, explore 9
- Backtrack to 8, 7, 5, 3, 4

So, that is already marked. So I will go back to 8 but 8 has nothing more to do because 5 and 7 were also marked but 8 was triggered by 7. So I will go back to 7, 7 was triggered by 5, so I will go back to 5, 5 was triggered by 3 so I will go back to 3, 3 was triggered by 4 so I have come back to the empty stack. So I have nothing on the suspended list I have also now like before from 4 I explored 0 and 1 but indirectly through this 6 I have already explored 7.

So, 4 also has nothing to be done. So I say 4 is terminated and I quite. So this was how depth first search work. So in a way you can imagine depth first search the way you kind of browse on the internet. You start reading something interesting then you click on a link because you sees interesting you follow that and before you know where you are you had started reading something and you are somewhere far away.

So, then you have to go back follow back this links and go back to where you started and continue. So that is more or less what depth first search is doing. So you find the first interesting vertex and you go there. You keep distance suspension go there. Then find the first interesting vertex go there and so on.

(Refer Slide Time: 7:43)



Depth first search (DFS)

- Paths discovered by BFS are not shortest paths, unlike BFS
- Useful features can be found by recording the order in which DFS visits vertices
- DFS numbering — maintain a counter
  - Increment and record counter value each time you start and finish exploring a vertex

So, depth first search finds these long paths like we said the path it found from 4 to 7 was it said that 4 triggered 3 in case we could have kept this parent information which we did not but for everything which is marked as visited like in the breadth first search. You could also mark in depth first search why it was marked visited. So you could keep this parent information. So we said that 4 marked 3, 3 marked 5, 5 marked 6 then came back and then 5 marked 7.

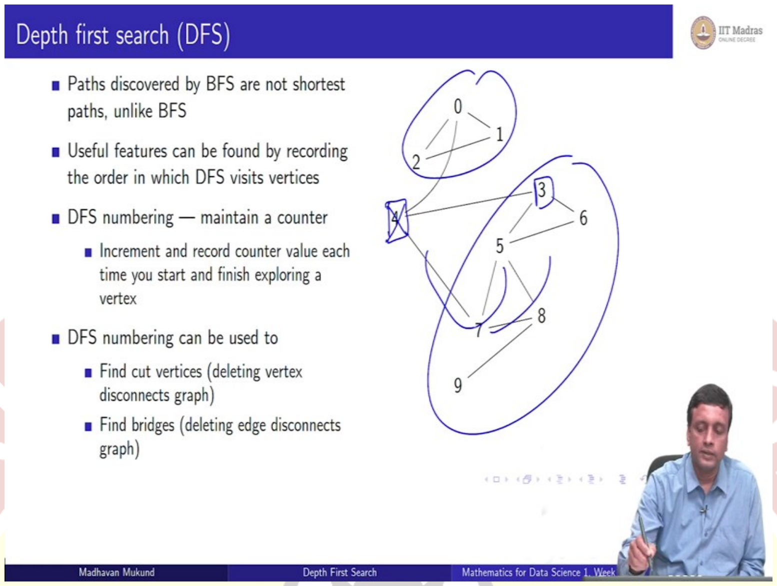
So, we found this long path 4 to 3 to 5 to 7 if we are kept the parent information you would have said parent of 7 is 5, parent of 5 is 3 and parent of 3 is 4. But this is obviously not the shortest path. So it does not do what breadth first search does not terms of shortest paths. So it seems a bit strange that we use this kind of indirect way of exploring when we have something which seems to be a better one namely breadth first search.

It turns out that actually depth first search is very useful for other things. So one thing that we can do in depth first search is keep track of how we visit these vertices. So what I can do, we will do it formally next time but informally I can say that we keep a counter. So we say that when the counter is 0, we entered 4. When the counter was 0 we entered 4 and from 4 we entered the vertex 0. So at this point the counter became 1.

From 0 we entered 2, entered 1 so at this point a counter became 2 from 1 we entered this. So this point the counter become 3. Now, we finished because the nothing to do at 2. So we finished 2 at the counter was 4, when we finished 1 the counter is 5, when we finished 0 the counter is 6. And now we have come back to 4 so now from 4 when we explore 7 the 3 the counter is 7 and so on. So in this way we can keep a counter incremented every time we enter a vertex for the first time and record it against that vertex as the incoming number of that vertex.

And then every time we finish processing a vertex that is all its edges have been visited then all its neighbours have been visited then we mark it saying that this is when we finished it. So we started 1 vertex 1 at counter 2, we finished vertex 1 at counter 5, we started vertex 0 at counter 1 we finished vertex 0 at counter 6. So notice that the counter value tells you something. It tells you that 1 was visited after 0 because 1 started at 2, 0 started at 1 and 0 finished after 1 because 1 finished at 5 and 0 finished at 6.

(Refer Slide Time: 10:19)



Depth first search (DFS)

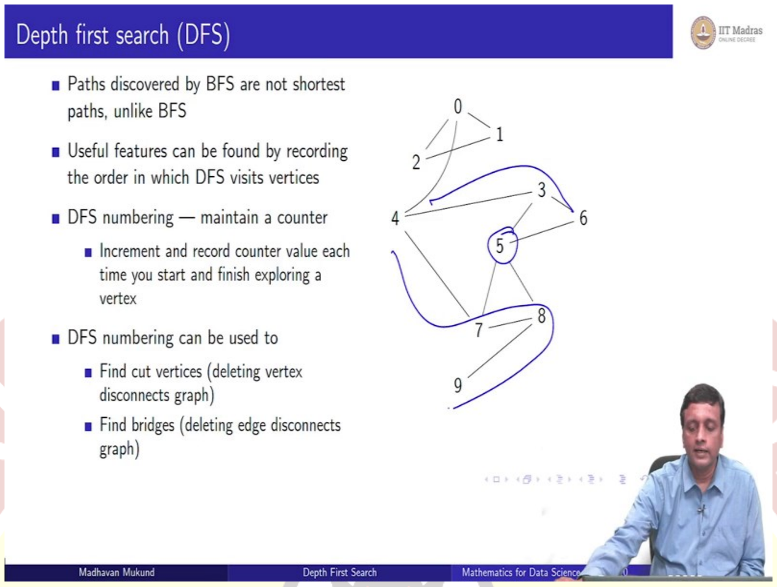
- Paths discovered by BFS are not shortest paths, unlike BFS
- Useful features can be found by recording the order in which DFS visits vertices
- DFS numbering — maintain a counter
  - Increment and record counter value each time you start and finish exploring a vertex
- DFS numbering can be used to
  - Find cut vertices (deleting vertex disconnects graph)
  - Find bridges (deleting edge disconnects graph)

So, this DFS numbering is very useful and we can use it to find many interesting things. So there are some things that we talk about last time as problems on graph. We talked about coloring, we talked about matching and so on. But sometimes you also want to finds special vertices. So for example, supposing there is a critical vertex in your graph. So imagine an electrical network.

Supposing there is one power station which is very critical that anything happen to this power station then the entire electrical network will get divided into 2 disconnected portions. So this is called a cut vertex or sometimes called an articulation points. So in this network for example if this is a airline network. And we are imaging that some airport is unavailable because of bad weather.

So, supposing there is a cyclone, so supposing vertex 4 the airport at city number 4 is knocked out of service because of a cyclone then you can see that there is no way to get from these vertices to these vertices. So we can still travel between 3 to 9. We can still travel between 0 1 2 but we cannot travel from 0, 1, 2 section to this. So now the graph has become disconnected. So would we discover that 4 is such a vertex. For instance, 3 is not a such a vertex if I remove 3 the graph still remains connected by this outer path.

(Refer Slide time: 11:37)

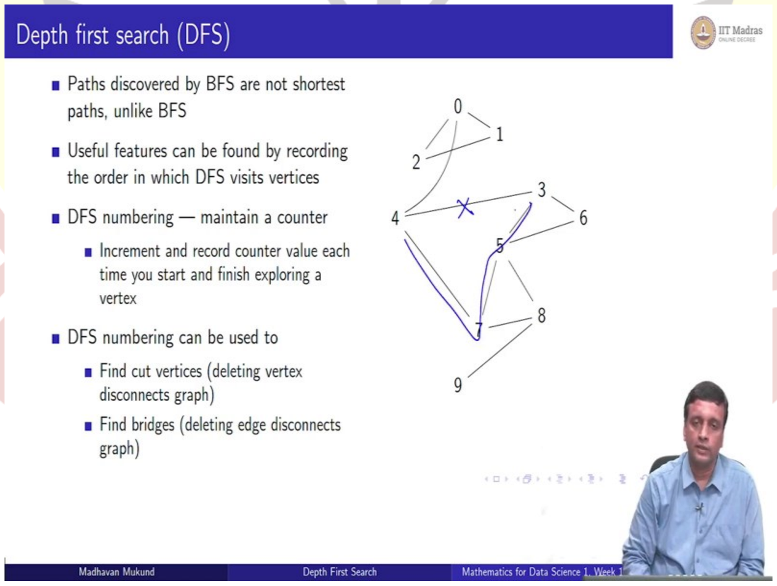


Depth first search (DFS)

- Paths discovered by BFS are not shortest paths, unlike BFS
- Useful features can be found by recording the order in which DFS visits vertices
- DFS numbering — maintain a counter
  - Increment and record counter value each time you start and finish exploring a vertex
- DFS numbering can be used to
  - Find cut vertices (deleting vertex disconnects graph)
  - Find bridges (deleting edge disconnects graph)

So, this 3 is not a, so if I remove 3 there is no (())(11:39). If I remove 5 also there is no problem, I can still go from 4 to 6 this way and I can come to these, this way. So which are the articulation points of cut vertices, you can discover that using DFS numbering. A related thing is this is for vertices, related question is for edges. So are there edges which are critical for me.

(Refer Slide Time: 11:59)

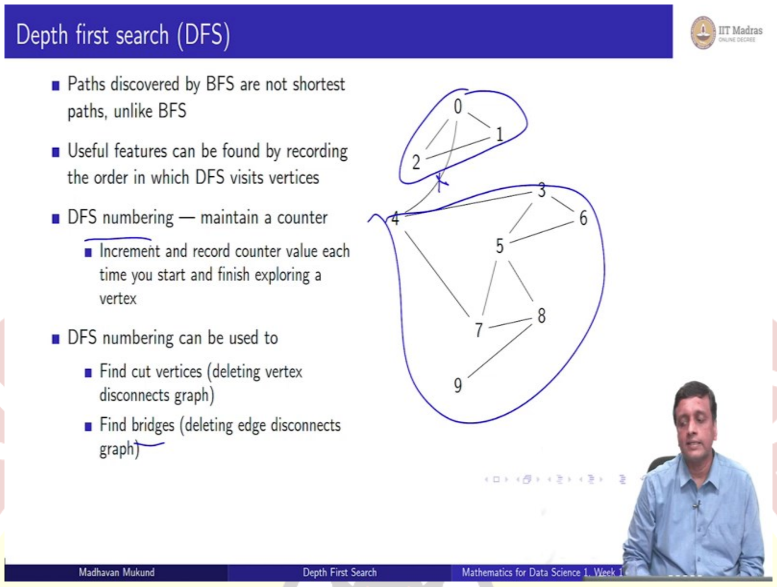


Depth first search (DFS)

- Paths discovered by BFS are not shortest paths, unlike BFS
- Useful features can be found by recording the order in which DFS visits vertices
- DFS numbering — maintain a counter
  - Increment and record counter value each time you start and finish exploring a vertex
- DFS numbering can be used to
  - Find cut vertices (deleting vertex disconnects graph)
  - Find bridges (deleting edge disconnects graph)

If there is a root which I cannot follow will that disconnect things for me? So now you can say that this route is okay because it is not a critical route because of if I cannot go from 4 to 3 directly I can still follow an indirect path and go from 4 to 3.

(Refer Slide Time: 12:16)



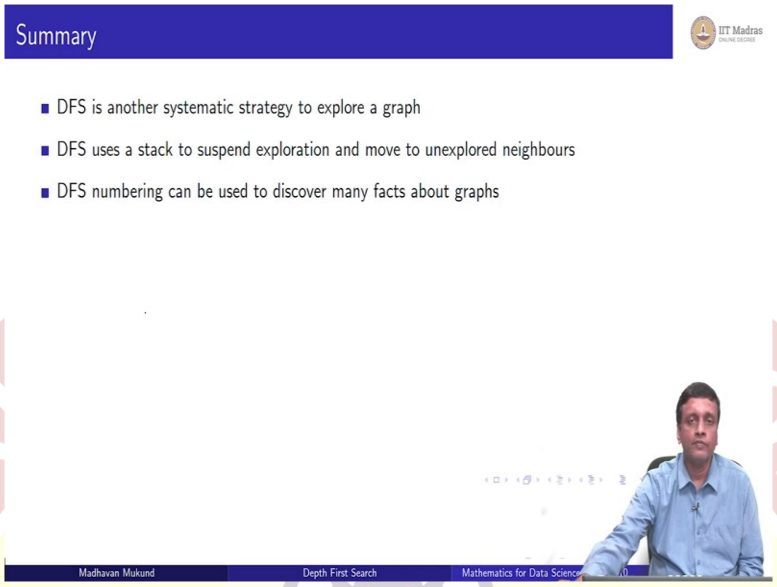
Depth first search (DFS)

- Paths discovered by BFS are not shortest paths, unlike BFS
- Useful features can be found by recording the order in which DFS visits vertices
- DFS numbering — maintain a counter
  - Increment and record counter value each time you start and finish exploring a vertex
- DFS numbering can be used to
  - Find cut vertices (deleting vertex disconnects graph)
  - Find bridges (deleting edge disconnects graph)

But again if I knock off this one, if I knock off the route from 4 to 0 then it turns out that 0 1 2 is disconnect from the rest of the network so this is called a bridge. So these kinds of properties of graphs, these cut vertices, bridges and many other interesting things can be discovered as a byproduct of depth first search.

So, you do a depth first search then you do this DFS numbering which we will describe in a later class. And using this DFS numbering you can actually find out interesting properties of the graph. So that is why depth first search though it does not find shortest paths it finds out interesting structure within a graph.

(Refer Slide Time: 12:51)



Summary

- DFS is another systematic strategy to explore a graph
- DFS uses a stack to suspend exploration and move to unexplored neighbours
- DFS numbering can be used to discover many facts about graphs

So, to summarize DFS is a different strategy from breadth first search, is another systematic strategy to explore a graph. So what we do is we start for the vertex, suspend it go to an unvisited neighbour, suspend it go to an unvisited neighbour and so on. And in order to keep track of these suspended vertices and how to resume them in a systematic way we use a stack. So we use this last in first out data structure in order to keep track of how to resume vertices when we come back from a terminated computation.

And we saw that although not in detail just informally, we saw that if we keep track of the sequence in which we visit vertices, when we finish, when we entered, when we finish we get this DFS numbering and with this DFS numbering we can actually uncover some structural properties of graphs which are quite interesting.