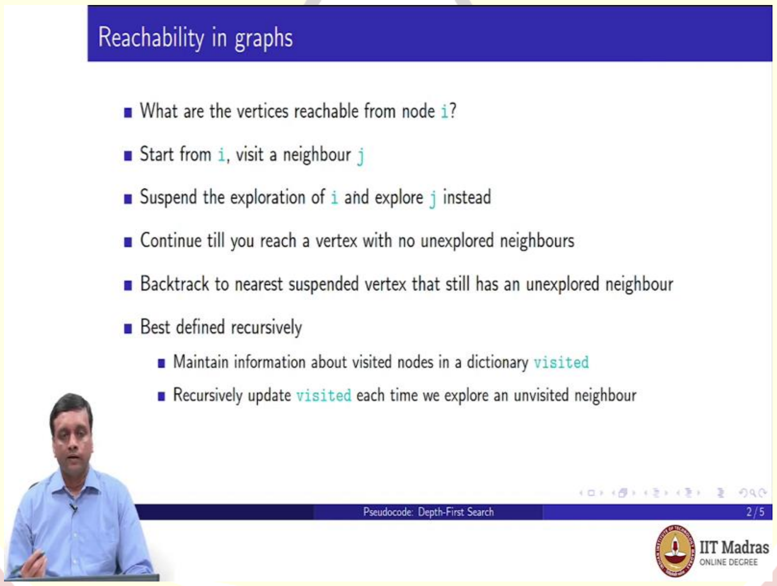
IIT Madras

ONLINE DEGREE

Computational Thinking
Professor Madhavan Mukund
Department of Computer Science
Chennai mathematical Institute
Indian Institute of Technology, Madras
Pseudocode for Depth First Search (DFS) and recursive procedure call

So, we looked at Recursive Procedures and we talked about inductive definitions where we have a base case and inductive case, but one of the important recursive procedures, which we have described informally and which we should now look at is a way to systematically explore a graph and this is called depth first search.

(Refer Slide Time: 00:36)



Reachability in graphs

- What are the vertices reachable from node `i`?
- Start from `i`, visit a 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
- Best defined recursively
    - Maintain information about visited nodes in a dictionary `visited`
    - Recursively update `visited` each time we explore an unvisited neighbour

Pseudocode: Depth-First Search 2 / 5

IIT Madras ONLINE DEGREE

So, suppose we start at a node i in a graph and we want to know what are the nodes or vertices that are reachable from node i. So the obvious thing to do is to start exploring the neighbours of i. So, there are two ways to do this, we can explore all the neighbours then the neighbours of the neighbours and so on and this is called breadth first search. I find all the vertices which are reachable in one step from i then all there are reachable in two steps from i all there reachable in three steps from i and so on.

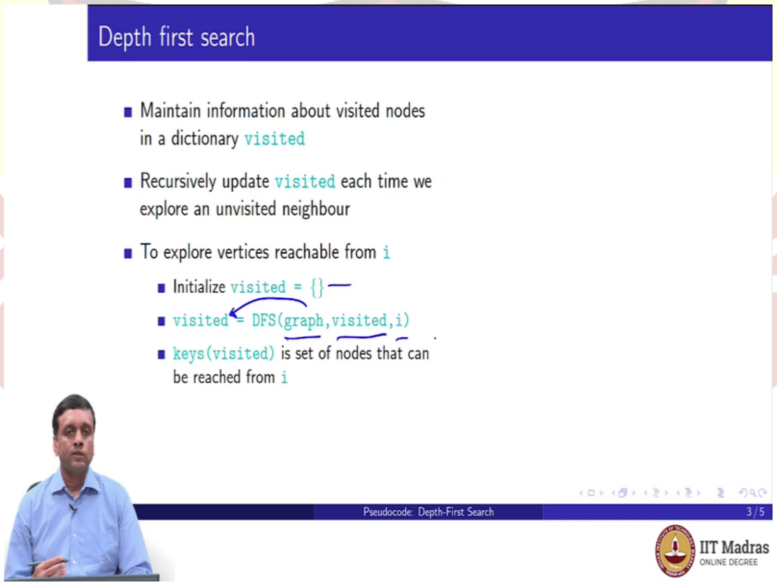
So, this is breadth first search and this will eventually find all the vertices. Another way which is very interesting is depth first search, which is I say okay pick any neighbour of i now go to that neighbour and continue exploring from that neighbour do not come back and look at the next neighbour of i until we have explored the first neighbour j.

So, we start from i visit a neighbour j then we recursively explore j. Now, why will this recursion stop that is because we have already visited i so when we explore j, we have to tell j not to visit i again so we have one fewer node to visit. We keep doing this until we cannot visit anything else.

Now, we come back to i and see whether it still has any unexplored neighbours and continue. So, this procedure called depth-first search is best described recursively. So, we will see a recursive implementation and we will see how it works. So, in order to make the recursion work, we have to make sure that when we explored the neighbour of I that we do not come back and start exploring i again so we have to say that i is already being explored it is already been visited.

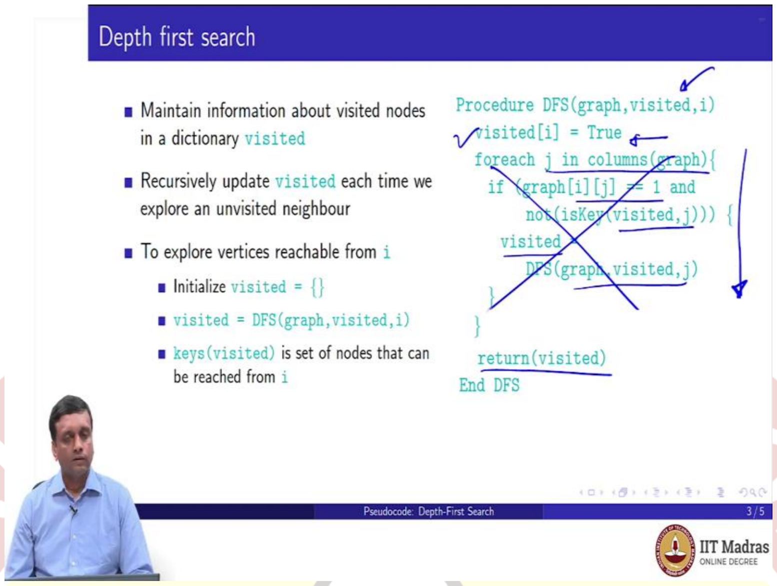
So, we have to maintain some extra information about which nodes have been visited and keep incrementing this and once a node is visited it will not be re explored. So, this is why the computation keeps reducing in size in terms of the argument. So, depth first search will keep calling itself with a smaller set of nodes to explore and eventually when there are node nodes left to explore depth first search will return without doing anything.

(Refer Slide Time: 02:42)



Depth first search

- Maintain information about visited nodes in a dictionary `visited`
- Recursively update `visited` each time we explore an unvisited neighbour
- To explore vertices reachable from `i`
  - Initialize `visited = {}`
  - `visited = DFS(graph,visited,i)`
  - `keys(visited)` is set of nodes that can be reached from `i`

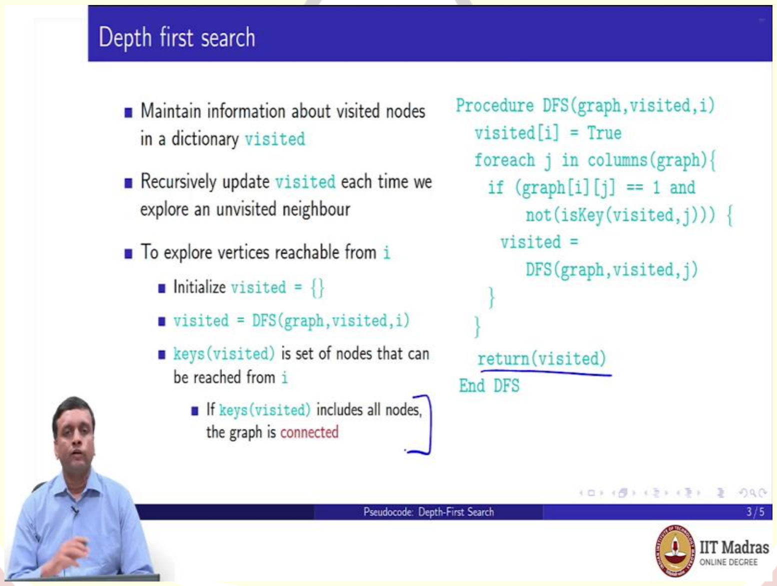

Depth first search

- Maintain information about visited nodes in a dictionary visited
- Recursively update visited each time we explore an unvisited neighbour
- To explore vertices reachable from i
  - Initialize visited = {}
  - visited = DFS(graph,visited,i)
  - keys(visited) is set of nodes that can be reached from i

```
Procedure DFS(graph,visited,i)
  visited[i] = True
  foreach j in columns(graph){
    if (graph[i][j] == 1 and
        not(isKey(visited,j))) {
      visited =
        DFS(graph,visited,j)
    }
  }
  return(visited)
End DFS
```

Pseudocode: Depth-First Search 3 / 5

IIT Madras ONLINE DEGREE


Depth first search

- Maintain information about visited nodes in a dictionary visited
- Recursively update visited each time we explore an unvisited neighbour
- To explore vertices reachable from i
  - Initialize visited = {}
  - visited = DFS(graph,visited,i)
  - keys(visited) is set of nodes that can be reached from i
    - If keys(visited) includes all nodes, the graph is connected

```
Procedure DFS(graph,visited,i)
  visited[i] = True
  foreach j in columns(graph){
    if (graph[i][j] == 1 and
        not(isKey(visited,j))) {
      visited =
        DFS(graph,visited,j)
    }
  }
  return(visited)
End DFS
```

Pseudocode: Depth-First Search 3 / 5

IIT Madras ONLINE DEGREE

So, we keep this information in a dictionary, so visited will be a dictionary and each key will represent a visited node a node for which we have already been there before. So we initialize it to be the empty dictionary and we will now have this procedure which will take as input of course this the graph this is the matrix describing the edges in the graph it will take the current state of visited vertices. So, current dictionary telling us which vertices have been visited so far.

And then which is the node from where we are exploring. So i is the new node, which we are going to explore and out to after exploring i the set of visited vertices is going to be updated by i of course and everything the reach from i so that will come back. So, I will update my visited vertices by taking the whatever dictionary I get back from this depth first search and updating it.

So, in this process after I have executed depth first search and updated visited by reassigning it like this then the keys of this visited matrix if I started with an empty dictionary will be precisely those vertices which I can reach starting from i. So assuming that I start with the empty thing and then I call this right at the beginning. So, i is the first vertex I explore then when I started I will explore nothing and then after I gone through this what I would like is the DFS returns all the vertices that were visited in the process of exploring the neighbours of i.

And in particular, of course if this tells us which all things are reachable. So, here is the procedure is very simple. So we assume that we get an incoming matrix called visited and the reason that we are exploring i is because i has not been visited so far. So the first thing we do is that we add an entry for i write so we say that visited i is true.

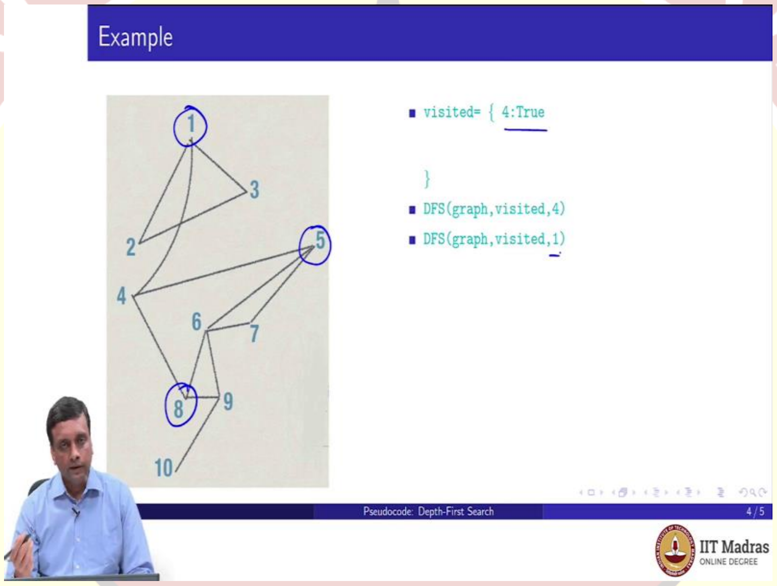
So, if visited i were already true, we should not be here that is an assumption the assumption is we will not call DFS for a node which is already visited. So, when I come in to DFS with a node that I am trying to explored the first thing to do is to mark it as visited and now I need to check all the neighbours of i so I go through every column in my graph matrix check if there is an edge from i to j and (())(05:06) i to j and I have not visited that neighbour before.

So, j is not a key in the visited matrix then I will suspend my exploration of i and I will visit j instead. So, I will do a DFS from j and at the end of the DFS from j I will get back a bunch of nodes which are visited through j. So, I will update the visited matrix to be that matrix so notice that I passed it visited i to be true. So it will come back with visited i to be true because I have already said it be true but a whole bunch of other things will be j and all that will be thing.

And finally when I have finished processing all the neighbours of i I will return the visited matrix that i have computed I mean the vertex i has computed, notice in particular that if all these neighbours of i if every i, j which is set to 1 actually is already a key then this will do nothing at i it will do nothing except mark i it will set visited of i to be true then this whole thing will just do nothing because for every vertex that possibly go through from i you have already seen it before. So, you will just update visited with one more (())(06:17) visited i is true and then come back. So that is the extreme case which happens when everything has been visited in the neighbourhood of i.

So, let us understand how this works by looking at an example. So, the first thing to notice that if I start with the visited matrix to be visited dictionary to be empty and then I explore from i and it comes back with every vertex visited that means that the graph is connected that is starting from i I can reach every vertex in the graph. And if I do not get back all the vertices, that means the graph is disconnected because there are some vertices that I cannot reach from i and therefore those fall into a different group of vertices, which are not connected to the group, it is connected to i.

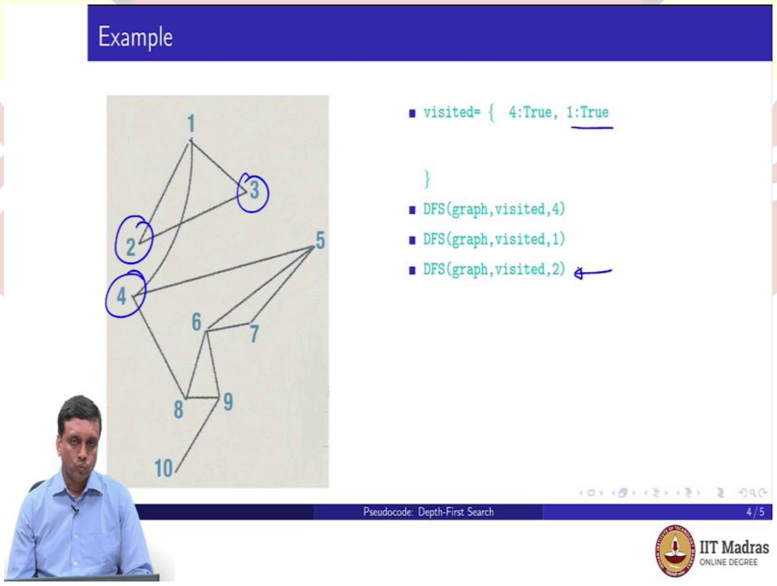
(Refer Slide Time: 06:59)



Example

- visited= { 4:True

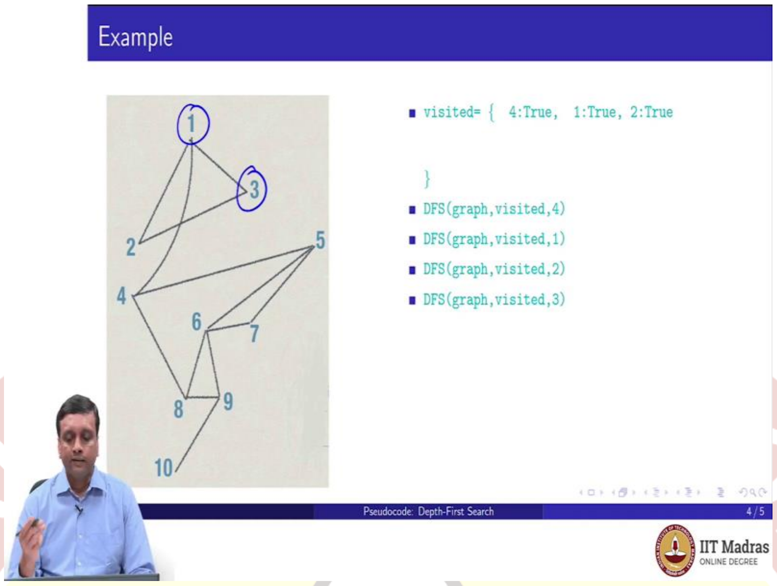
  }
- DFS(graph,visited,4)
- DFS(graph,visited,1)



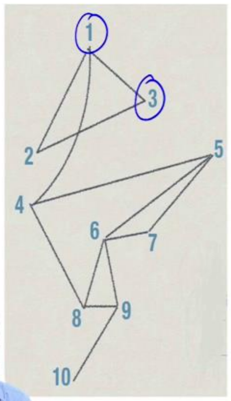
Example

- visited= { 4:True, 1:True

  }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)



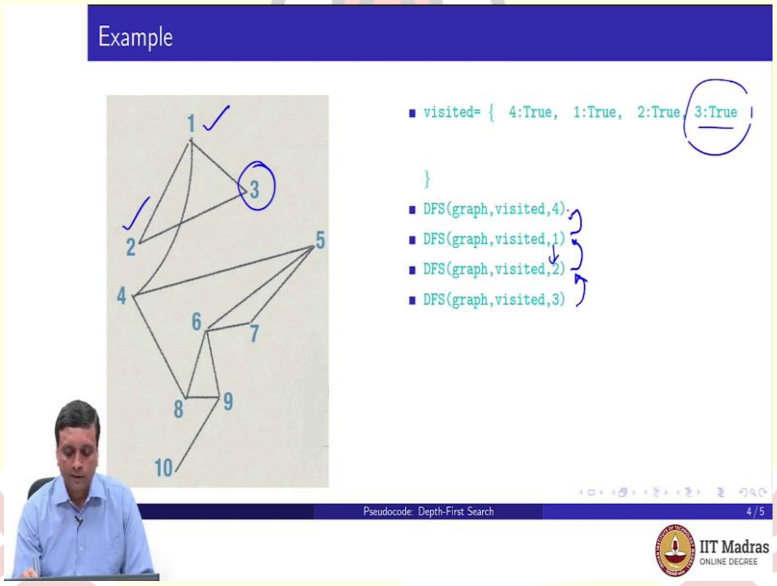
Example



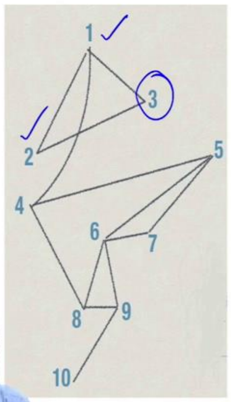
- `visited= { 4:True, 1:True, 2:True`

  `}`
- `DFS(graph,visited,4)`
- `DFS(graph,visited,1)`
- `DFS(graph,visited,2)`
- `DFS(graph,visited,3)`

IIT Madras ONLINE DEGREE



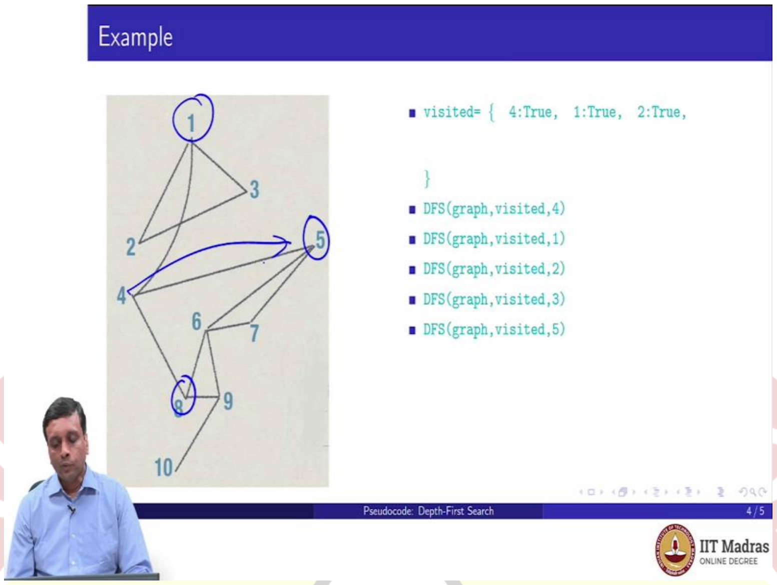
Example



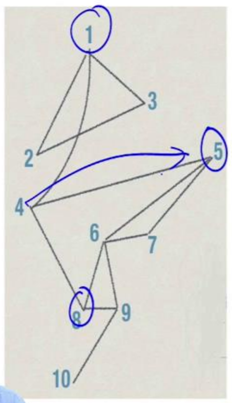
- `visited= { 4:True, 1:True, 2:True, 3:True`

  `}`
- `DFS(graph,visited,4)`
- `DFS(graph,visited,1)`
- `DFS(graph,visited,2)`
- `DFS(graph,visited,3)`

IIT Madras ONLINE DEGREE

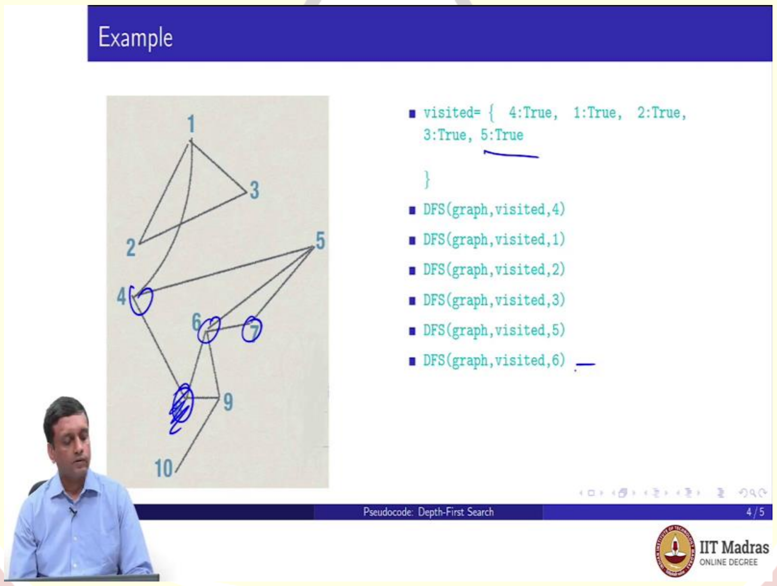


Example

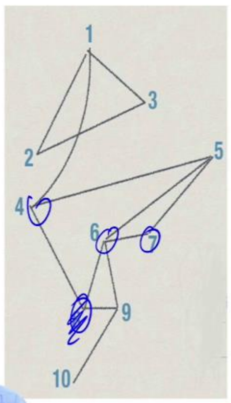


- visited= { 4:True, 1:True, 2:True,

  }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)

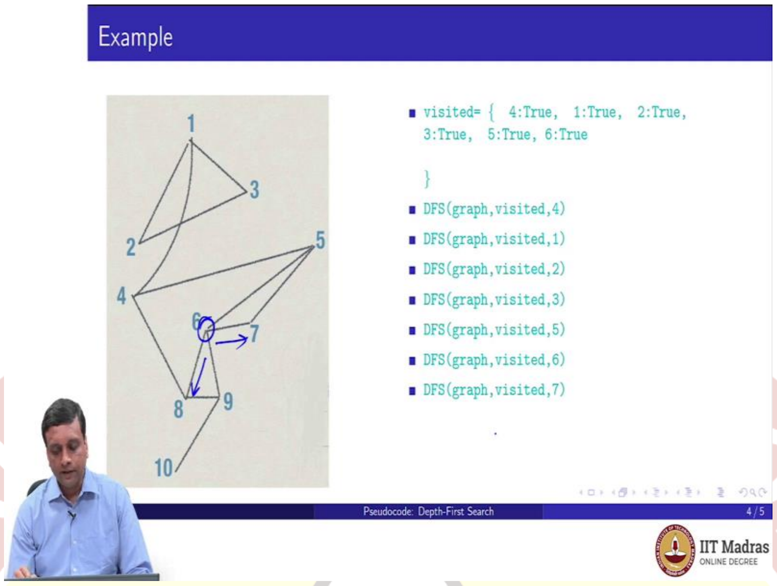


Example

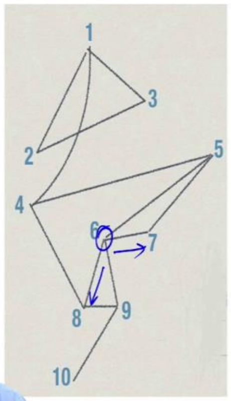


- visited= { 4:True, 1:True, 2:True, 3:True, 5:True

  }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)
- DFS(graph,visited,6)

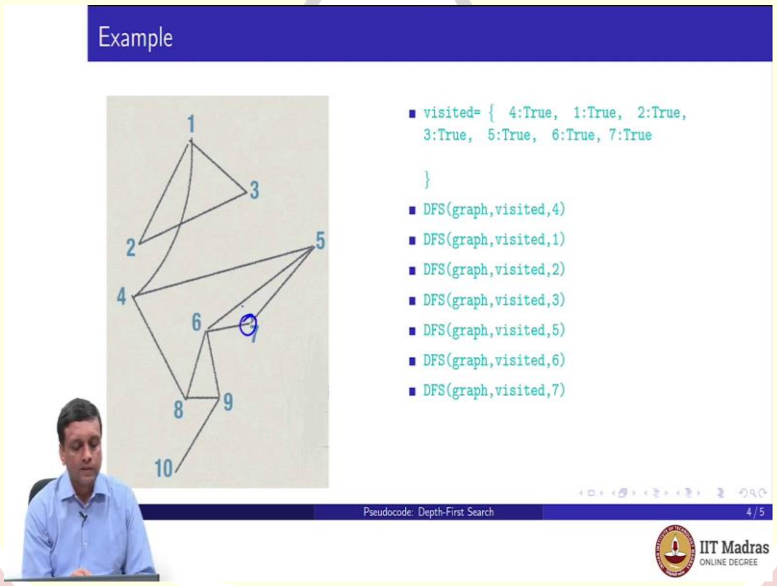


Example

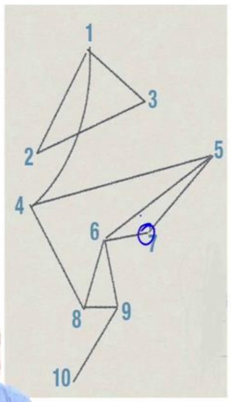


- visited= { 4:True, 1:True, 2:True, 3:True, 5:True, 6:True }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)
- DFS(graph,visited,6)
- DFS(graph,visited,7)

IIT Madras ONLINE DEGREE

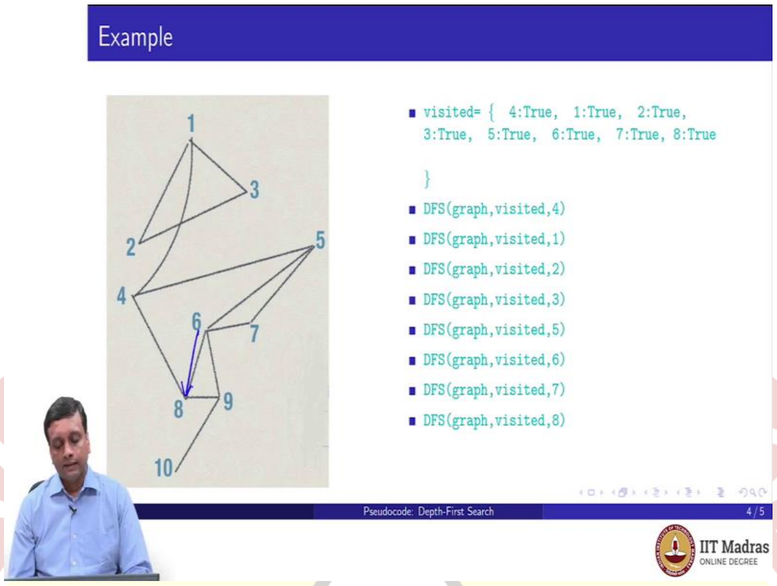


Example

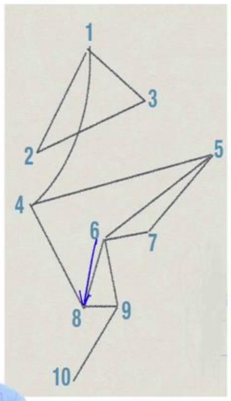


- visited= { 4:True, 1:True, 2:True, 3:True, 5:True, 6:True, 7:True }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)
- DFS(graph,visited,6)
- DFS(graph,visited,7)

IIT Madras ONLINE DEGREE



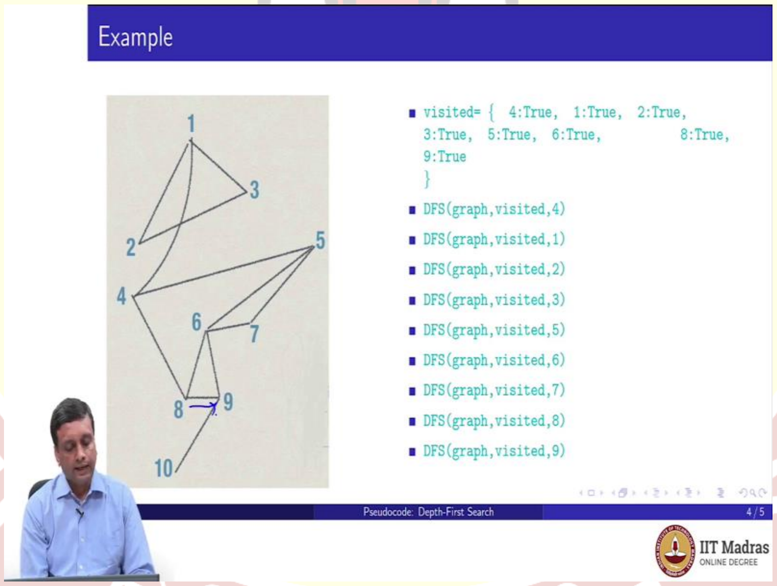
Example



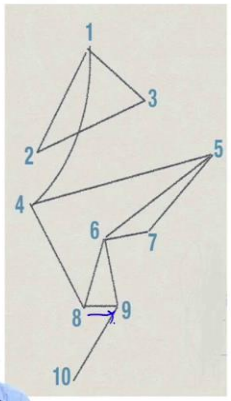
- visited= { 4:True, 1:True, 2:True, 3:True, 5:True, 6:True, 7:True, 8:True }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)
- DFS(graph,visited,6)
- DFS(graph,visited,7)
- DFS(graph,visited,8)

Pseudocode: Depth-First Search 4 / 5

IIT Madras ONLINE DEGREE



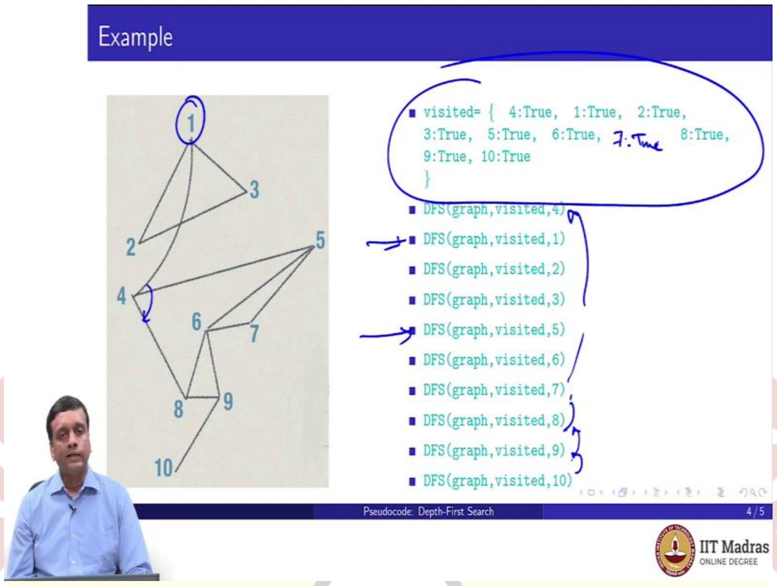
Example



- visited= { 4:True, 1:True, 2:True, 3:True, 5:True, 6:True, 8:True, 9:True }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)
- DFS(graph,visited,6)
- DFS(graph,visited,7)
- DFS(graph,visited,8)
- DFS(graph,visited,9)

Pseudocode: Depth-First Search 4 / 5

IIT Madras ONLINE DEGREE



Example

- visited= { 4:True, 1:True, 2:True, 3:True, 5:True, 6:True, 7:True 8:True, 9:True, 10:True }
- DFS(graph,visited,4)
- DFS(graph,visited,1)
- DFS(graph,visited,2)
- DFS(graph,visited,3)
- DFS(graph,visited,5)
- DFS(graph,visited,6)
- DFS(graph,visited,7)
- DFS(graph,visited,8)
- DFS(graph,visited,9)
- DFS(graph,visited,10)

Pseudocode: Depth-First Search 4 / 5

IIT Madras ONLINE DEGREE

So, let us look at this example. So I want to start DFS and find out what all I can reach from vertex 4. So, I start with this empty dictionary visited nothing is been visited so far and I call DFS with this vertex. So the first thing that happens is that visited 4 get set to be true and then I have to look at the neighbours of 4 so the neighbours of 4 are 1, 5 and 8.

So let us assume that we always explore the neighbours and kind of increasing order. Remember we are going through the columns from 0 to n (())(07:34). So here is actually 1 to n where does not matter. So we are going through the columns from smallest to largest. So I will choose to visit 1. So I set visited a 4 to be true and I choose to visit 1.

So now I go into a recursive call I have suspended the call for 4 and I have gone into a call exploring the neighbours of 1. So the first thing that happens is that I have to set that 1 is visited and then I have to look at the neighbours of 1 which are 1, 2, 3 I mean 2, 3 and 4. If I go in order then the first thing I pick up is 2. So when I go into 1 I will set 1 to 2 and I will now start exploring 2.

Once I explore 2 then I will set 2 to true and I have to know explore the neighbours of 2 which are 1 3 when I exclude the neighbours of 2 the first neighbour is 1 but 1 is already visited. So, I will skip 1. The next neighbour is 3 3 is not visited. So I will call 3. So now I go to 3. So now at 3. So, having DFS of 3 I said 3 to be true saying is visited and now I have to look at its neighbours which are 1 and 2.

But now 1 and 2 are both visited already. So, I cannot do anything from 3. So, the DFS call for 3 returns with no change except for setting 3 to be true and now what happens is that this returns back to the DFS call for 2. Now, 2 has no change left because the next vertex that I have to explore from 2 is 4 which is already visited. So 2 will return to 1.

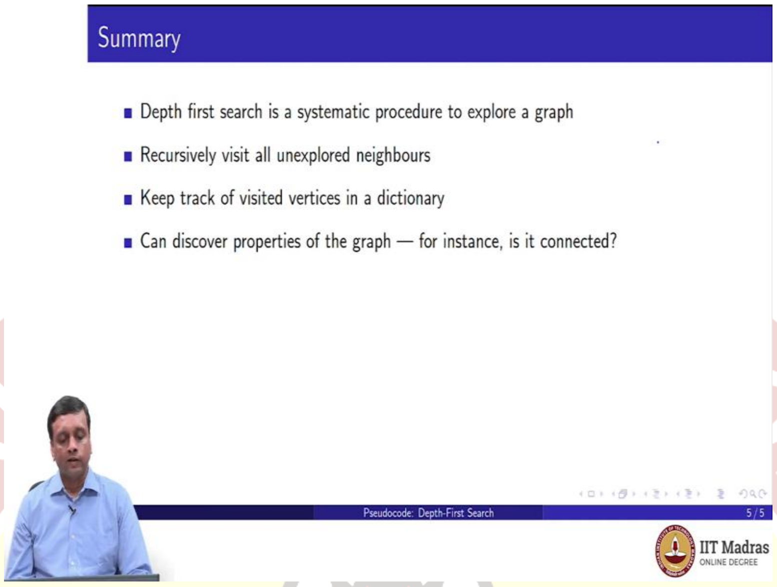
And now if the call for 1 has no vertices to explore because I have already explored 2 I went from 1 to 2. But now if I look at 3 3 was visited via 2 4 visited because for was the one that called 1 to begin with so 4 will come back so I will come back to 4. So after calling 3 I will come back to 4 again and now remember that 4 had these neighbours so it will pick the next neighbour of 4, which is not visited which is 5.

So I will now go from here to 5. So when I call 5 then 5 becomes true and now 5 has neighbours 6, 4, 5, 4, 6 sorry 4, 6 and 7. So since 4 is already visited it will call 6. So, I will set 6 to true then I will go to 6 and I will look at its neighbours 7 and 8 so it will call 7 so 7 to true now when I come to 7, it has neighbours 5 and 6 but both 5 and 6 have already been done. So, there is nothing further to do with 7, so I come back to 6 and then I go from 6 and I go to its next neighbour 8. And then 8 will visit 9 and 9 will visit 10.

Now, at this point everything is marked let everything is marked visited. So when I look at the neighbours of 10, there is only 9, which is already visited. So, I will come back to 9. Now look at 9 it has no more un-visit neighbour 6 8 and 10 are all visited. So we will come back to 8 8 has no more neighbours to visit 4 6 and 9 are visited it will come back and so on.

So eventually I will come back to 4 and now remember I had done 4 to 1 and then I had done 4 to 5 so in principle now I can go from 4 to 8 but 4 will find that 8 is already been visited so it will say okay now I skip 8 and then I finish. So the DFS called to 4 will finally come back with this list of visited vertices and since this list has every key in some added in some random order 4, 1, 2 ,3, 5, 6, 8, 9, 10 there should in a 7 somewhere which I seem to lost. So since this thing has all the keys it you can actually verify by looking at this picture, of course manually that you can reach every vertex from 4.

(Refer Slide Time: 11:31)



Summary

- Depth first search is a systematic procedure to explore a graph
- Recursively visit all unexplored neighbours
- Keep track of visited vertices in a dictionary
- Can discover properties of the graph — for instance, is it connected?

Pseudocode: Depth-First Search
5 / 5
IIT Madras
ONLINE DEGREE

So depth-first search is a recursive procedure mean it is not necessary recursive but it is easiest to describe it recursively is a systematic way to explore a graph by recursively visiting all the unexplored neighbours and keeping track of the visited vertices in our case we use a dictionary. So, we added each node that we visited to this dictionary and we kept updating this dictionary and returning the updated dictionary from depth for search so that after each call to depth first search we have the updated status of which nodes have been visited through the recursive call.

And one example one simple example, we saw is that once we get back the list of visited vertices through the initial call we know whether or not the graph is connected. If every key every vertex is present is a key. It is connected there are lots of other interesting things that you can do with that first search which you will find out in other courses and later on it is not necessary to do it now before us therefore depth for search is basically a way of finding out what all is connected to a given vertex.