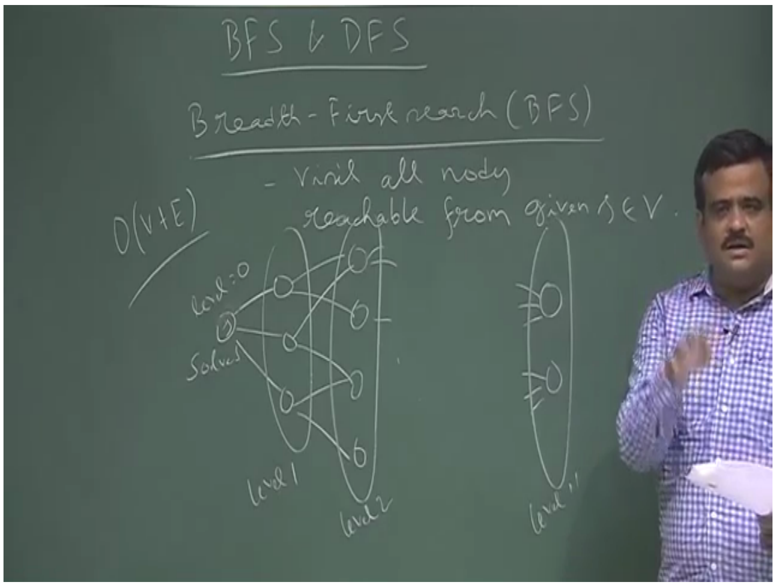
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 41
BFS & DFS

So, first is breadth first search or BFS Breadth-First Search or BFS. So, this is basically the idea is to. So, you are given a graph. So, idea is to visit all the vertices you have to explore all the nodes and a reachable form, so you have to start with a vertex a reachable from given vertex s form v and we just level this vertex as 0 level. So, it is a level wise.

(Refer Slide Time: 00:20)



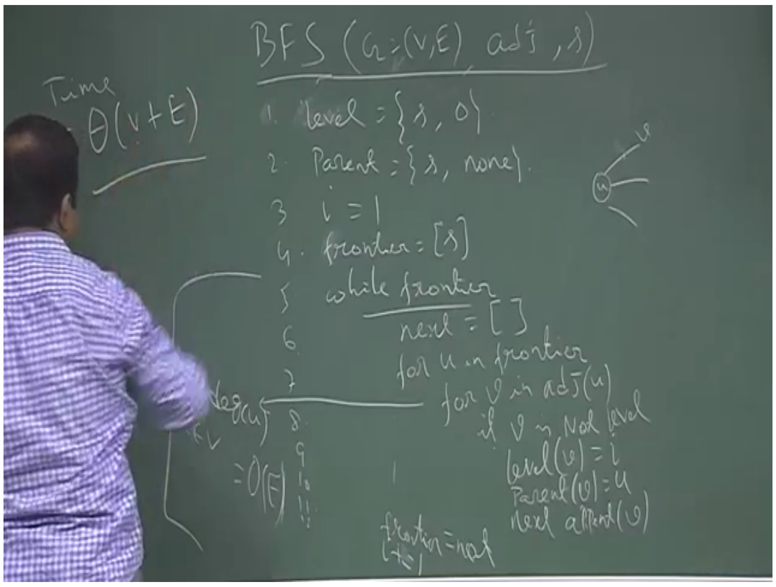
So, we start with a vertex is like in for the Rubik’s cube what we have seen in the last class the Rubik’s cube the solve state this each position each state is a vertex. So, solve steady is a vertex and from the solve steady if you change one 90 degree move change. So, that is the one move change we can go to the solved state like this. So, this is our S this is the solved state for the Rubik’s cube and these are the vertex which is just one step away from this all state and these are the vertex which are just 2 state away from the solve state like this and like this.

So, these are the vertex final vertex which are basically 11 stay 11 step away from the solved state. So, this is basically so we start. So, this is basically level 0 level 0 vertex these are level 1 these are level 2 like this. So, these are level 11 if it is 2 by 2 Rubik’s cube then the dimension is 11 the per length of this is 11 that is the god number. So, this is the level next level; that means, level 1 and level 2 vertex then this is the level 11 vertex like this. So, this is basically level wise search and this we want to do it in V plus E times if V is the dimension of the vertex V is the number of vertices and E is the number of h.

So, this we want to do in V plus E times that is the goal. So, basically we will looked all the vertices which is all the vertices reachable form S and we will just look at the adjacency list of each vertex. So, we start from S we look we first look at the adjacency list of all vertices form S which is adjacent to S. So, that will assigned as level 1 I mean S is level 0, then again when we capture all the vertices in level 1 then we go for level 2. So, those are basically adjacent to the vertex which is in level one. So, those will be in the level 2, so like this. But we need to careful to avoid the repetition so we that we have to avoid the duplicate.

So, we a vertex which is already is mark that should not mark again. So, let us write the code for this. So, the idea is basically level by level search which are reachable form s.

(Refer Slide Time: 03:57)



```
BFS (G=(V,E) adj , s)
1. level = {s, 0}
2. Parent = {s, none}
3  i = 1
4. frontier = [s]
5  while frontier
6      next = [ ]
7      for u in frontier
8          for v in adj(u)
9              if v in Not level
10                 level(v) = i
11                 Parent(v) = u
                   next append(v)
       frontier = next
       i += 1
```

So, this is BFS by input is a graph G and the this is the vertex said G is basically V comma E and we have the adjutancy list and we have a starting vertex. So, that is also input of the BFS we have to start from at a. So, what we do? We level, this is the code, we level the vertex s as 0 and the parent of s as none because that is the starting vertex and then we assign the i to be 1, i will be indicate as the level and we assign another set called frontier we assigned frontier which is initially as s and this we have a loop while frontier is empty while frontier; that means, while frontier is not empty. So, this is basically a python code.

So, next is basically next with would empty and then we take the element from the frontier for u in frontier and we take the all adjutancy vertex of u in u that is v say we take all the adjutancy vertex of. So, u is in frontier this is u u is by initially u is s, you take all the all the nodes which are adjacent to u. So, this is the vertex v; v vertex and if v is not level then you have to level v if v is not level so; that means, v is not mark or not seen basically we have to avoid the duplicate v if v is not seen then level of v level of v will be i and parent of v is u and we have to append this v in the next.

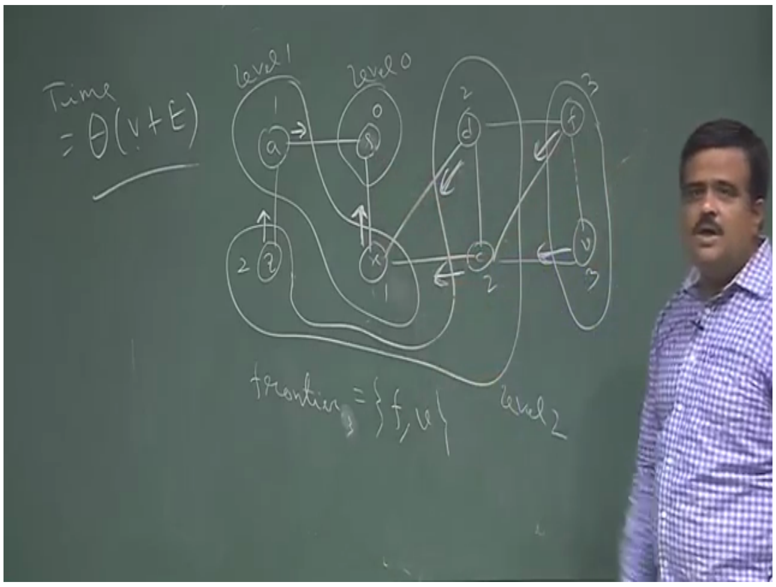
Basically next append this v we have to append v to the next, for the next level vertex this is 6 7 8 9 10 11 say up to 12 and then after this after this the frontier will be next and i is equal to i plus plus, i is equal to i plus plus. So, that to indicate for the next level vertices, this is the pseudo code this is basically pascal code this is sorry python this is the python language, this is the code basically. So, what we are doing we are starting with the vertices and we have leveling the vertices 0 and the parent of S is null because that is the starting vertex and then we have putting this S in frontier and then why in frontier is not empty we have to do this ok.

And we for each this we have to we are having a set next and what we are doing we are taking a element from the frontier because frontier is not empty that is why you are going in this loop we are taking a element in front from frontier that is u and we take all the adjutancy vertex form a vertex with u, this is v. And now v is not level you have to level v y i and then we put this and parent of v, v assigned to u because v is the parent of v and we put this v to the next set next to indicate that is the these are the next level vertices and we put next after completing this v assigned this next set to the frontier and we repeat this again and the for the next level will be i plus 1. So, this is the idea.

So, let us take an example. So, what is the time complexity for this we will come to back within that time complexity yeah so, but we what is the time complexity. So, this is we are doing the basically what this is the degree of the vertices. So, this is basically degree of u and this is basically we are doing while frontier. So, v times and so this is basically summation of u belongs to v. So, this is by again by (Refer Time: 09:33) order of E and we are marking all the vertices; that means, we are viewing all the vertices the time will be order of V plus E because we are visiting all the vertices. So, that is why time will be order of V plus E ok.

Now, let us take an example of this algorithm. So, suppose we have a graph like this a say s which at x d c f v and these are the say edges it. So, on say this one this one suppose we have a h from here this, this.

(Refer Slide Time: 10:01)



Suppose this is a graph this is a input and we have to run the BFS breadth first search on this input. So, it is a level by level search.

So, we start with the vertex a vertex s, s is the source vertex. So, what we do? The frontier is basically frontier is basically s initially so; that means, we level this vertices 0 level vertex. Now we check frontier is empty know frontier is not empty. So, we extract v, this is the u vertex now you check all the adjutancy vertex of this. So, adjutancy vertex of this is this 2. So, this is 1 2. So, this is basically sorry 1. So, this is basically this 2 are adjutancy base. So, this is the next level, now, is 1, this is the next level vertices.

So, this is basically frontier i is equal 0 and, so this is the 0 level and this is the level 1 so; that means, frontier of 1 level i is equal to 1 is basically a comma x . So, now, this is the 0 level this is 1 level. So, this is level 0. Now we have to now frontier is this now we have to check all the vertices, you take this vertex if you take this vertex who are the adjutancy list this is the in adjutancy list, but this is already done this is already leveled. So, we want to avoid the duplicate. So, we are not doing anything on this, this is not level.

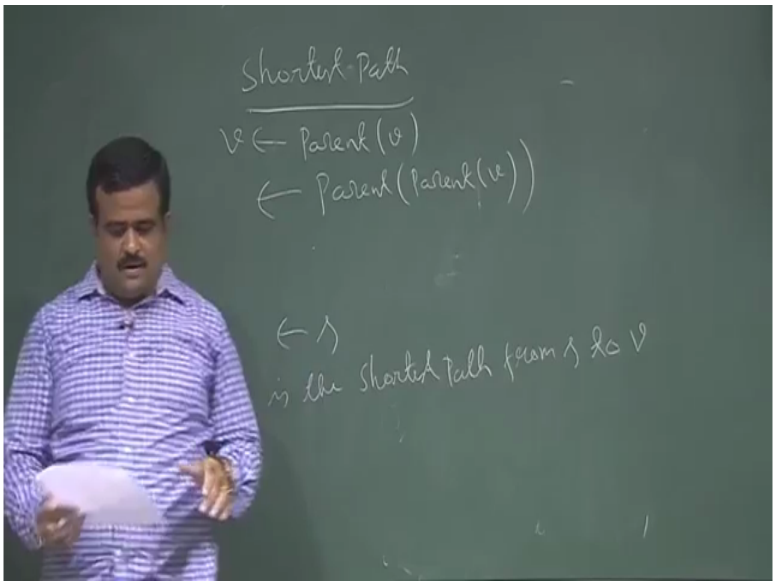
So, you have to gong to level this by 2 now, i is 2 basically now go to here. So, here we have how many vertices this is also there, but this is already leveled. So, you are not touching this. So, these are this 2 vertices. So, this is basically level 2 level 2 so; that means, basically we have now frontier is frontier of level 2 is basically now this set. So, these are this words is next it convert to the, so z d c, this set, so this set is basically level 2 vertices and now this is our now this is our next and this will be in frontier and i is now i plus 1. So, i is now 3.

Now, look at all the vertices from here these are our u for this vertex this is already done for this, this vertex this is not done. So, you have to put this level 3 this is done this is done this is already not done we have to put level 3 and all are for this vertex this is already level this is level. So, they all are done. So, now, this is basically now the frontier. So, basically frontier 3 is basically f comma v and this is basically level 4 level 3 vertex, now i is four this is level 3 vertices.

Now we put the arrow for this parent we have a parent pointer. So, if you put this arrow for this parent pointer. So, this parent of this is basically s because this was capture from s and we are then we capture this also. So, parent of this is basically s. So, this array is basically parent pointer and parent of this is basically a and parent of this is basically this one and parent of this is this one how we are capturing how we are leveling so that way. Now this basically we capture from here. So, we could capture from here then the arrow should go like this, but we capture from here. So, this is the arrow and this is the arrow. So, this is the parent pointer.

Now, this parent pointer has a importance. So, what is that? Now parent pointer we will give us the shortest path from that node to s. So, if you follow this arrow. So, if you start from s a so we follow this arrow this this. So, this will give the shortest path from that node to this node. So, this is the parent pointer advantage. So, how to find the shortest path? So, basically if we want to find the shortest path from s to the a f to s. So, we have to follow this arrow.

(Refer Slide Time: 15:55)

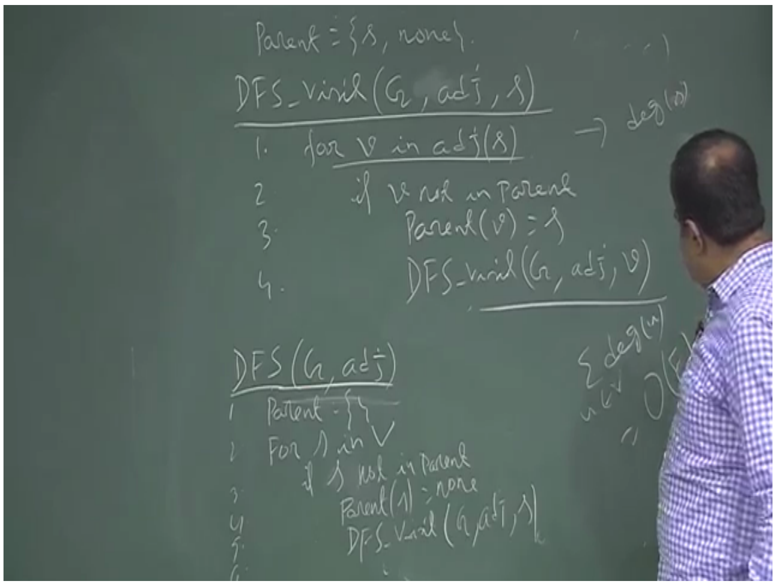


So, basically the shortest path finding shortest path, so v now we will list to parent of v now again parent of parent of v. So, dot dot dot and I will finally, will reach to s.

Now, this is the shortest path from s to this is the shortest path from s to v is the this can be proved shortest path in the sense that there is no way, but shortest path is means number of edges we are visiting is less in that sense. So, this is basically called breadth first search and it is time complexity is V plus E.

Now we will talk about depth first search. So, this is also another way to explore a graph. So, this is called DFS. So, when solved it is called DFS.

(Refer Slide Time: 17:16)



```
Parent = {s, none}.
DFS_visit(G, adj, s)
1.  for v in adj(s)                → deg(u)
2.      if v not in Parent
3.          Parent(v) = s
4.          DFS_visit(G, adj, v)

DFS(G, adj)
1   Parent = {}
2   For s in V
        if s not in Parent
3           Parent(s) = none
4           DFS_visit(G, adj, s)
5
```

$\sum_{u \in V} deg(u)$

$= O(E)$

So, DFS has many application like h classification and topological sorting we will see about those. So, just to, this is basically it is basically the idea is is basically we this is back tracking tracking we follow all the paths until we stuck, we start from a vertex we follow all the path we just keep on going until we stuck. So, this is say a nice example of match. So, if you just follow if you stuck then we will come back to there and we try for other path. So, that is the way. So, we will do the back tracking, tracking because we may stuck. So, we follow the path until we got stuck. So, follow path until we stuck.

Now, once we stuck, once we stuck then we will back track we will come back to that previous position and then we choose another path and we continue until we stuck there like this. So, we start from here say we keep on going until you stuck there is no way to go then we come back here then we export the other path from here again we if we stuck we will come back we will explore the other path from here like this. So, back tracking is necessary here. So, recursively we will do this. So, back tracking we need to do and this will co this is a recursive explore a recursive call recursively we explore the graph because when we again stuck we will come back there we will choose the another path there, but we have to careful that there should not be any repetition there should not be any duplicate repeat of the vertices. So, if a vertex is mark then it should be done.

So, let us just write the pseudo code for this. This is also in write try to write in python. So, this is basically. So, parent of. So, we just, so, it has 2 code 1 is DFS visit which is basically we have a graph input and we have a adjacency list and here we have a source vertex. So, DFS visit is having a input with source vertex. So, for this parent of the source vertices null. So, what we do which for which take all the adjacency list of s and if it is knows a explore that; that means, parent we are using the parent to indicate that whether we have seen this vertex before or not if it is not seeing.

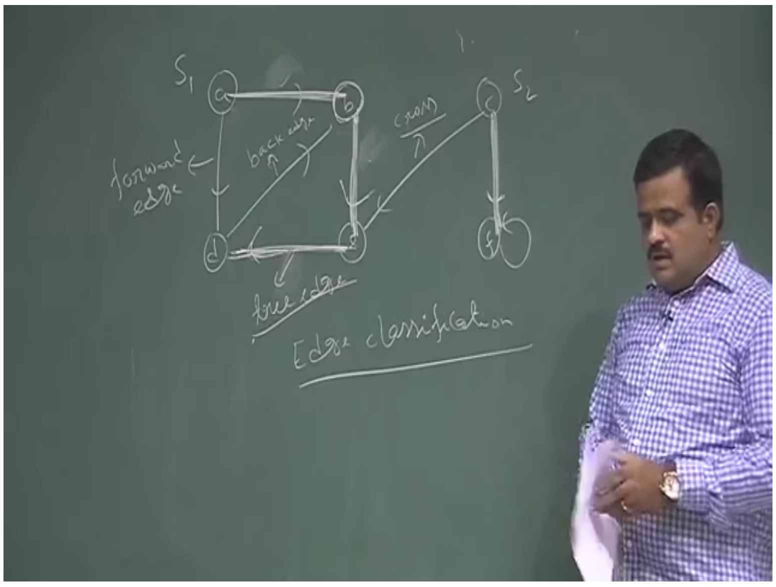
So, if v not in parent; that means, if v is not seen before then will level this v parent of v has s and we call DFS base if this is the recursive call on this v. So, graph adjacency list and on this v. So, this is the recursive call, very simple code. So, this is the recursive call. So, when we just follow that. So, we start with s. So, is this is s we start with s we go to v then again we call this DFS it is visit from v if v is not done and we keep on doing.

Once we start we will come to here we will choose another vertex v. So, this is for all the vertex which are adjacent to s. So, once we explore with this will. So, that is why it is a recursive call and it is back tracking. So, this is with the one vertex now the final the main code is basically DFS visit DFS which is taking the graph and the adjacency list. So, here is no. So, knows starting vertex. So, we just having parent as empty and then we choose a arbitrary vertex as s, for s in v if s is not in parent if sorry s not in parent; that means, s is not explored then what we do we just assign parent of s as null and we call this DFS visit on s.

DFS visit on G adjacency vertex and s, very simple code. So, we just start any arbitrary vertex as s then we if s is not mark we have to chose a vertex which is not mark then we will explore this. Let us take an example. So, before that what is the time complexity for this? So, again time complexity will be depending on this is basically or degree of s basically and this is doing for all the vertices. So, basically summation of degree of u, u belongs to capital V. So, this is basically order of E and since we are visiting all the vertices. So, this is basically a time is basically order of V plus E because we have to we have to visit all the vertices.

Now, let us take an quick example. So, we want to take a quick in some suppose we have a vertex a b say d e we have c vertex over here and so we have a f vertex ok.

(Refer Slide Time: 24:38)



And then suppose we have this edges this one, this one, this one this is a directed graph it can act from un directed graph also this one this one and this one and we have a self h over here suppose this is the giving here and we want to explore the DFS on this . So, now, we have to choose vertex or starting with suppose we choose these as a s.

So, this is a, we denote by s 1. So, this is our starting vertex say s. So, now, we just need to find out. So, now, we have to start from s we need to explore keep on exploring with the all the adjacency vertices. So, of we choose this, this is the number 1 then this is we call again DFS on this. So, this is having adjacency vertex this, we go their. So, again this is adjacent with this we go their. So, this is say 2 this is 3 this is the ordering will be visit the now we this is we have to call DFS is visit, visit from this, but this is having a vertex adjacent to this which is already in the parent which is already marked. So, we stop here.

Once we stop here we come back here we have to see all the adjacency vertex of s. So, that is the back tracking tracking. So, we explore a path we come back because we stuck here we come back here then we see this is the adjacent another adjacency vertex of v so, but this is already in this is already in the parent. So, so we are not doing anything for this. So, this is the way. So, this is the S G S 1.

Now we choose another vertex from this graph which is not done; that means, which is not in the mark at a parent. So, this is basically the vertex. So, we may start with any of this vertex. So, we base start with this S 2 if you start with the S 2 then this is our new s. So, this is the vertex, but this is already in this is already marked this is this, this parent mark is already done. So, this is explored.

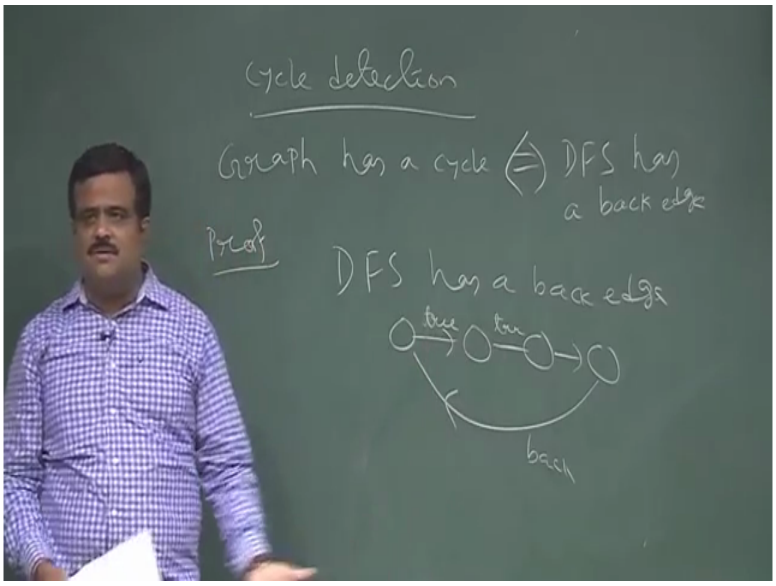
Now, this is not explored. So, we have to explore this, this is basically 3. So, this is basically not explored. So, this is we have to. So, the there is no need of this number in anyway. So, this is, now, this we mark the parent of this as this. So, this is also mark as a parent now we call this, but these as only you self look which is already done. So, we stop then is no other vertex left out. So, this is the v f S search. So, we just start with a vertex and we keep on going until we stuck once we stuck we will come back to that vertex we explore the other path like this. So, this is the depth for search.

So, now, there is. So, now, we can label the basically we can label the edges. So, we can classify the edge. So, this is call edge classification we can classify the edges how? Now these edge basically form say tree. So, these edge will be form say tree I mean if it is connected then it is form a tree otherwise it will form a forest. So, this edges are called tree edge which forms the parent basically tree edge, this edge call tree edge which basically for this bold color edge which is basically forms the parent. And other non tree edge like this is a non tree edge say this one this one is a non tree edge this is basically called forward edge non tree is either forward edge back edge or cross edge.

So, why this is called forward edge? Because we have these ordering of that this, so this is coming after this. So, this is a edge form this 2 this ordering. So, that is why it is called forward edge because this is the edge we have visited first and this is the edge after that. So, this is the edge form that this ordering is which we have which vertex we have visited first earlier than which vertex we are visited later on. So, those type of edge is called forward edge, but these edge if you look at these edge these edge is basically going from this to this these we have visited later on and these we have visited before this. So, this is called back edge, this is called back edge. And these edge which is connecting the 2 tree different. So, this is called cross edge and other than back edge and forward edge any edge is called cross edge. So, that is the classification of edge.

So, this will help us to have some application like how to find the cycle detection of the edge. So, let us just quickly have that.

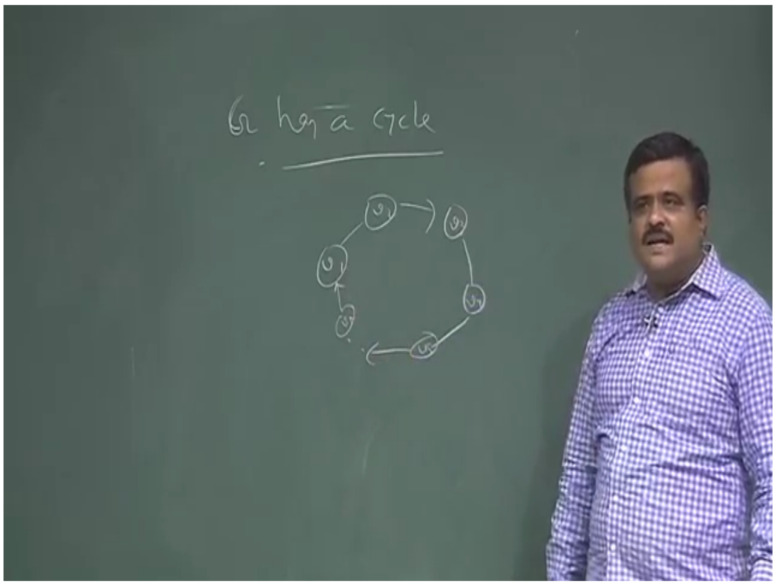
(Refer Slide Time: 30:55)



So, one application is cycle detection how to detect a graph is having a cycle or not. So, given a graph, this is the theorem. So, if there is a back edge then there is a cycle. So, given a graph has a cycle implies if we run the DFS the DFS has a back edge. This is vice versa. So, if DFS has a back edge then it has a cycle.

Now let us prove, then let us try to proof this. So, first of all we assume the DFS has a back edge. So, we are first proving this side DFS has a back edge. So; that means, what? That means, we have this time things and this is going this these are the tree edge and we have a edge like this. So, this is back edge. So, these are the tree edge the forming the tree, tree has tree edge and this is the back it say so obviously, this is the back edge this is cycle because this is a back edge. So, it will form a cycle. So, that is quite clear. So, if the DFS has a back edge then there is a cycle. So, if this way you can detect the cycle now other way if there is a cycle then how to show the DFS has a back edge, so other way around. So, that is also interesting.

(Refer Slide Time: 32:45)



Suppose we graph has a cycle G has a cycle. Now, we have to, the DFS has a back edge suppose this is a cycle say we have say v 1, v 2, v 3 like this v 4, v 5 dot dot dot say v k like this dot dot dot suppose there is a cycle, it start with v 1 and it going to v 2. Now we do not know in our DFS algorithm which vertex we select to start. So, if we select say for simplicity we can say we are selecting that vertex v 1 if we select this vertex v 1 then it will keep on going these are all tree edge like this, these are all tree edge these are all tree edge like this, these are all tree edge, but this will be back edge because this cannot be tree edge because this is already done. So, in back case this will be the back edge. So, in that case this will be the back edge. So, this is one application to find the cycle in a graph there are other applications also like topological sorting and job scheduling algorithm. So, those are in the text book.

Thank you.