NPTEL
NPTEL ONLINE CERTIFICATION COURSE

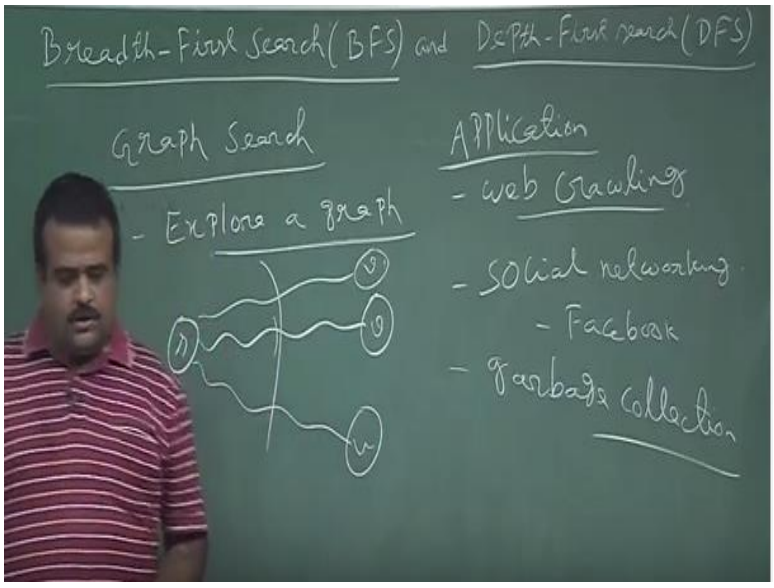
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhpadhyay
Department of Mathematics
IIT Kharagpur

Lecture 17: BFS and DFS

Okay, so we will talk about BFS, DFS this is basically the problem of graph search.

(Refer Slide Time: 00:27)



So basically we want to explore a graph, exploring a graph so how can we explore a graph. So suppose we have two vertices, two vertices A and B and we want to have, we want to see the, what are the path from A to B, we want to find the path of A to B.

So this is only of exploring a graph, not only B, we want to explore all the path from A is to any other vertices like this. So this is also one way to explore a graph. So we want to see from A

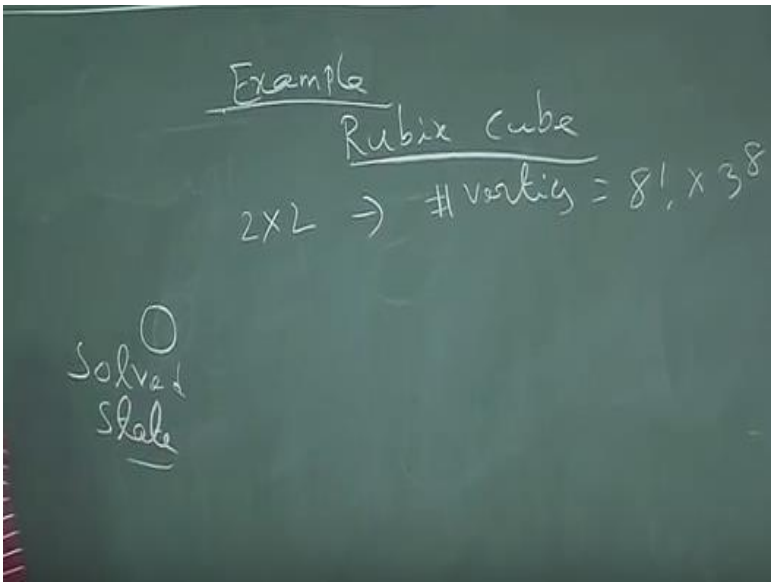
what are the vertices we can reach, so this is one way of exploring a graph. So not only A we want to see what are the, we want to explore all the vertices, okay.

Now these are some application, graph search, so first application would be like web crawling, so this is basically used in Google, so suppose I design a new webpage and Google has to get that webpage into their database. So how to explore that, so Google has to add new and new database, a new and new webpage.

So Google will do that, this such DFS okay. Now another maybe the social network, social network thing, okay. So friend finder like in Facebook, so how to find a friend. So you have to explore that and another application could be say garbage collection. So now it is every modern programming language is having this garbage collector, so we do not need to see the memory we have used, so it will automatically check by the program whether something is using or not.

So it will explore if it is not using for long time, so it will free that space for further use that is the garbage collection. So there are many applications of graph search, so today we will talk about this two way, how we can search a graph, how we can explore a graph okay. So this is, we start with the BFS, breadth-first search.

(Refer Slide Time: 03:35)

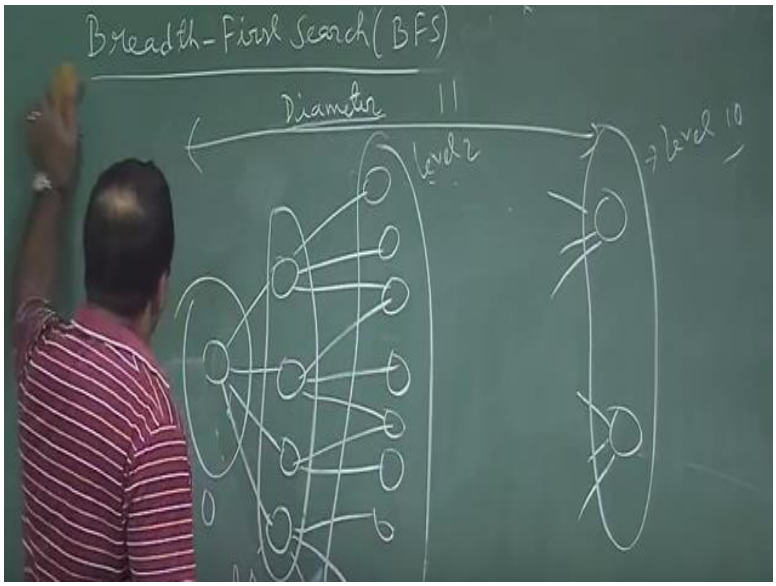


So we will take an example of Rubic cube or the pocket cube okay. So this is a 3/3 Rubic cube so it has a solid state and if we, so each position is we can define a state. So we can have a graph where each position is the state. Now if we change it then it will give us another state by just one move. So this is a state.

So it has a solid state okay, so now, so how many, so if it is a 2/2 Rubic cube then how many states are there basically, the number of states are basically that is basically number of vertices, it is basically factor of 8 x $3_{8,}$ it is huge it is basically 2, 6, 4, 5, 3, 9, 5, 3, 0 so this many states okay. Now how we can form the graph based on this vertices, so these are the possible states this is on state it will change little bit this is another state, this is like this.

So this is the state okay. So now how we can, so this is, suppose this is a solved state so this is a solved state or we refer to a solved particle, solves vertex that means everything is matching all the rate over here like this okay.

(Refer Slide Time: 05:20)



Then we define, we take other state which are just one step hired from the solved state. So we take a solved state we just make a one move.

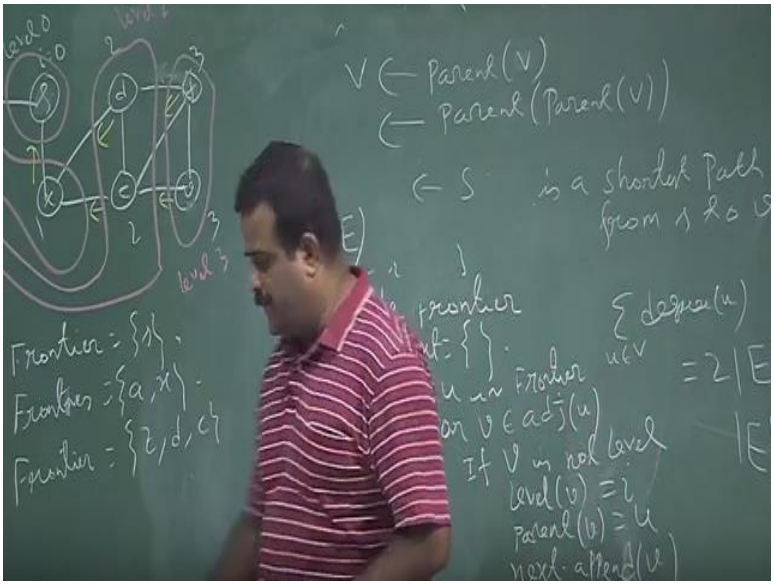
Then those are basically these states, so just a one move, one move from the solved state. And then we take another vertex, another state which is basically two moves from the solved state like this. So these are basically two moves, two move so to come to the solve sate we have to give two move we fall this and then this so like we continue so at the end we have the partitions like this so what is the this depth if it is 2/2 rubric cube this is 11 basically so whenever we are in ant state we can go to the solve state by 11 move this is called diameter or t is also called god number so this is the number of step we require from a any given state suppose we are here we can come to the solve state if we fall this path.

So this is one way to explore a graph this is basically BFS break first search we are reform this we are reaching to the all other vertices like this okay so this example of BFS so now we will formally we have the code for BFS so to have the code let us just define the label so these are basically we are defining these are the nodes which are in level 0 that means they are already in the starting stage.

So we take the solve state at the starting stage and these are in level one that means we need to have a one move to go to the solve state which are directly connected so these are the all adjacency vertices of this node and they are in all level two like this so if it is 2/2 rubic cube this are in all level 11 I mean 11 diameters so level 10 the last level okay so now let us write the code for the BFS.

So it is level wise exploring level by level so we first start with a source vertex is and then we explore all the adjacency vertices those are in the level one and then from there we explore all the adjacency of those vertices which are in level then slowly we will explore the graph so that is the idea.

(Refer Slide Time: 08:49)



So let us write the code BFS so we have these are vertices says we have a graph we have adjacency list and we have source vertices okay.

So let me write the code so level the source as 0 because source as that 0 level that I define parent of S is the in all and then we start the level 1 and the previous level is i-1 so pervious level means we at level we start with level 0 then level 1 and so on and the we put the frontier we put

So there okay now while frontier is not entity okay we take the next as initialized MT and then for V is frontier we take a vertex front initially frontier is basically continuing is B frontier what we do so we check all the adjacency vertex of I am sorry u is in frontier we take all adjacency vertex are this node.

So suppose we start with S so this is our u now we take all the adjacency vertex of this and we level them by I so level of v is i-1 so we level if it's I not level if v is not level because we do not want any repetition not level then we level of DY i so this is the code and we make point where parent of B is basically u this we require to have shortest path from this v to the source what we will explain so now next is basically we append in the next we append this D into the next so this is 6 , 7, 8, 9, 10, 11, 12 may be.

Okay then in 13 what we are doing okay so we just taking the frontier as a next and we take i=++,i+,i=++ for the next level, so this is the this is typically python code so this is what it is telling we start with a source then we and in the level the source as is 0 level and then slowly we will explore the graph, let us take an example, So suppose we have a graph like this we have a, s, z, x, d, f, c, v so suppose this is our partition and we have a just like this, this, okay. So this is our graph given graph.

And we have to apply the we have to explore this graph based on the this BFS, okay so do we start with this node this is our level 0 node so i = 0 so this we denote by level 0 node, okay. So now so our initialize our frontier is S this type, now frontier is not empty next is empty over here so we just take this U, so U is basically S now we take all the adjacent in node of E basically S, so these are the two adjacent in node of this.

So we level them by 1 this two and we put it into the frontier basically we put it into the next and next will be the frontier, so now the frontier is basically these two node AX and this is basically our level 1 node, okay. Now we take now frontier still node empty so we take a vertex from the frontier if we take as level 1 node, okay. Now we take now frontier still node empty so we take a vertex from the frontier if we take say yeah if we just take A then we take all the adjacent and vertices of A.

And we level them so z we level them 2 and if we take x we level them these to 2 and this is the next level this is the next level so this is the level two this is next level vertices and the frontier will become now I mean that is the next we have to know the nodes in S so this will become basically J, D, C okay, Now we take all the element so we take J so for J this is the adjacent C vertex which is already level.

So we will not do that again, now we take D for D this is the node sop we level it 3 and then for C this is the node we level it 3 this is the level 3 vertices, okay. And we also do the this Spherion mark the parent pointer which will give us the sort us for basically so from A so parent pointer of this is basically this node and the parent pointer of this is basically S and parent pointer of this is basically this node and this, this.

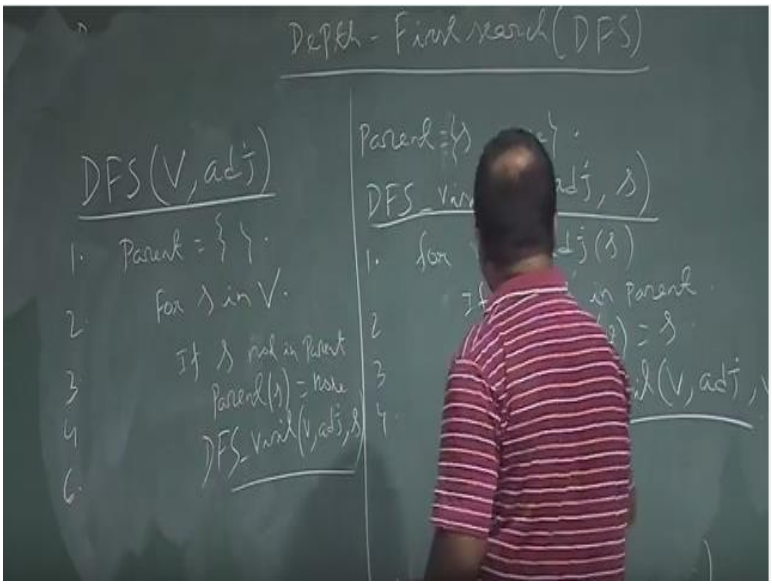
And parent pointer of this is basically I mean it depends how we expecting if we are expecting from D so this is basically this node and parent pointer off this is basically this node if we explore through D then it will get like this, okay. So this is the parent pointers so now what is the use of this parent pointer? We want to reach to suppose we have the factors V we want to reach from V to S, so how we can reach from V to S?

We just follow this parent pointer link an these will give us a shortest path, if you just take this explored this link so we will just explored this link like V then parent of V then parent of parent of V like this so slowly will these two case, okay. And now this is basically a shortest path this is a shortest path from A to V, okay. So we just follow this pointer and we get the shortest point so this is the BFS and what is the time complexity for this?

So the time complexity will be order of, so degree so what is the sum of the degree of u this is basically I mean if it is a directed graph, if it is unexpected graph it is 2|E| this is basically [indiscernible][00:18:24] and if it is directed in the, this is basically order of E, and we are visiting the vertices, so the time will be sort BFS it is basically V+E, okay so this is the way we just explore all the vertices. But every time we have to afford the duplicate, so that there will be once we visited a node we should not visited again if we have at the duplicate.

Okay, so now this is also giving the shortest path, if we just follow this parent pointer to that is and so this also solve the problem of the Rubic cube, so for Rubic cube if we start from any vertices, any vertices and if we know that position in that picture then we start from there we follow this path and we come back to the solve state. Okay, they there are almost if this 2/2 Rubic cube, this is 3/3 Rubic cube if it is 2/2 Rubic cube it is just 11 step we can use to solve state, 11 step, okay. Okay now we will talk about DFS.

(Refer Slide Time: 20:07)



So we talk about DFS or depth finite search this is also one into explore of okay, so this is coming from the problem what is called, exploring a rate, okay so we, we start exploring a means like we, we go to some, some way and if we start there we will come back again and then from this is sort of back checking, so we keep on continue, we keep on going until we stuck, if we stuck means there is no, where to go from there, so then we will come back that is the back tracking. We come back to the BFS position and then we try to explore the other path, so that is the exploring the mean.

So now, so this is basically we recursively, we recursively explore graph and we have to do the recursively, we have to do the back tracking if necessary, back tracking as necessary, okay. So

this is a recursive call, so let us just write the pseudo code for this, so we take the parent of, parent of s as null and we call this function DFS is it, this is pseudo code, this is a sub routine, so we have a graph input and s.

So this is basically actually DFS is we have to explore all the vertices. Now where we have exploring this with the vertices s, so if the graph is connected if you can explore each vertices from s then one sort it is done, but it the graph as disjoin connect, disjoin component then we have to do it all the vertices. So for this, so this is if you start with s and if you explore this, so what is the code, code is like this, so for, so basically the idea is this is s, so these are the all adjacently vertices of this v, so we keep on explore this from v.

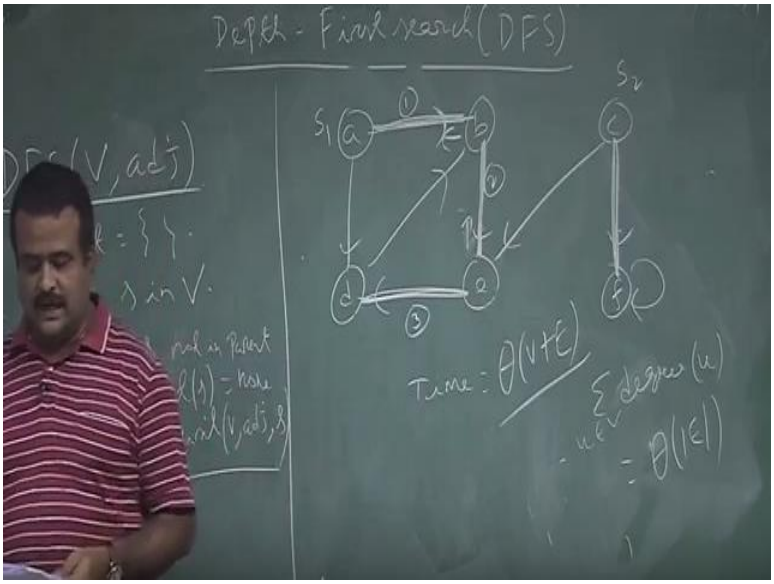
So we start with s we go to v then we keep on explore from v so that v also effect to have the reputation so we will keep track of that, so it is just write the code, so for v in adjacent instead of case, then we have check if v is, if v is not in parent that means v is not explore yet, then we mark parent of v which is basically u and we call this, this is a recursive call.

They DFS visit this is the state of parts and this is the okay this is the recursive so if it is not yet explore will again do the recursive call on this so we will continue this until we start if we start will come back and again we will take a point from the parent and we will explain that is the idea so this is the see code we have to call this code in the DFS the main code is DFS and we have a graph.

And the adjacent cell is this is the main code and this will call this pseudo code okay, so what we do we take the parent, so initially parent is empty and we take a note from this for s and V, we take capital we can start with any unknown, we take a note from s, from v and we check whether it is not explored if V is not in parent, if sorry if S is not in parent then we take the parent of S as null and we call this DFS visit, D, adjacent and in this base so this is the sub routine we will call for this, so this will do for all the vertices.

So we take a arbitrary vertex, we try to explore all the component connected to S and then if it is covered all the point till otherwise we have to take the another component. So we will take an example for this how it is working.

(Refer Slide Time: 26:30)



Okay, so let us take a graph, this is a, b, c, d, e, f, and suppose these are the a edges, suppose these are the edges okay, and there is a self here so suppose this is a graph so this is a we have to execute this DFS on this graph.

Okay, so suppose we take a vertex s, s could be any arbitrary vertex so we start with this vertex, we defined as S0 okay, so now this is our a, this is our S so now we take all the adjacency vertex, we take the adjacency vertex of this so this we can take or this we can take, if you take this then this is the number one edge so we cover this and we put it into the this parent then we keep on exploring, so we take these again and then we cover this, this is the number two and then we have a parent pointer like this, like this and then from here.

Where this is the adjacency vertex so every time we check whether this is already visited or not, so d is not visited so we will export these dates so this is, is number 3 and then from d this is the

vertex so but these has already explore so we will not export this vertex again, so then so we have to come back so we will come back here we check whether e has any other adjacency vertex or not, no, so we come back here, so we check whether d, b is having any other adjacency vertex or not, we will come back here, we check a is having any other adjacency vertex or not.

So yes a is having a adjacency vertex which is basically d but d is already visited, it is already covered okay. So now this, this part of the graph is over, now we have to take another vertex which is, say we can take this one S2 and we take adjacency vertex of this so we check this one so this one is already explored so we will not cover this. Now we take this one so this one we have to explore, so f now we take from S we have f so f is already explored we are doing anything and then from f there is no other vertices adjacent to it so we will go back here or we check any other vertex adjacency list of c or not.

Then we stop okay, so this is the basically DFS, now what is the time complexity for this, so basically time is we are exploring all the vertices and we for each vertices we are exploring the adjacency list, adjacency list degree, that is called degree how many vertex are there in the adjacency list, that is called degree, so this is basically we know this is basically order of e by hand setting them up, so this is basically and we are visiting all the vertices, this is basically order of v + e. Now these has many application this be a, this problem like we can classify the ah and we can find the cycle.

Whether there is any cycle in this graph, cycle testing or we can do the topological sort, so this is basically the two way graph exploring, one is BFS and other one is DFS, thank you.