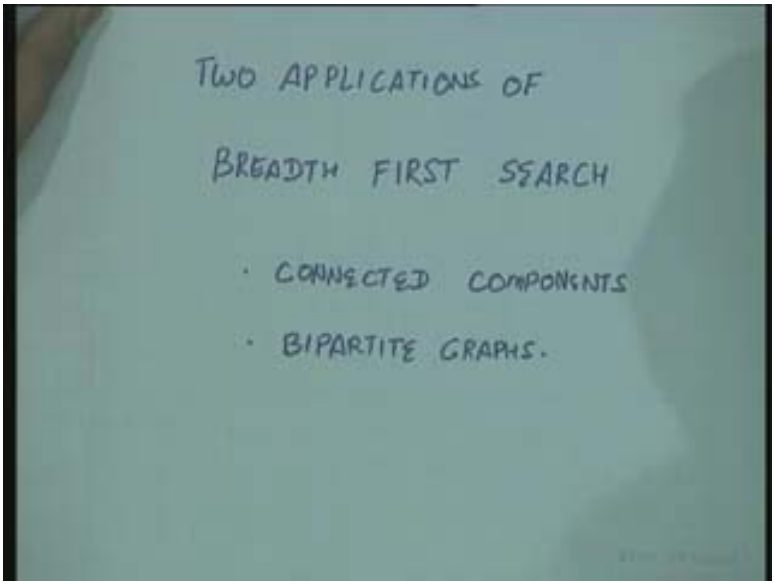
Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture – 26
Two application of Breadth First Search
-Connected components
-Bipartite Graphs

Today we are going to be looking at two applications of breadth first search. In particular I am going to look at applying breadth first search to finding out the connected components in a graph and a second application will be to checking if a graph is bipartite.

(Refer Slide Time: 01:20)



I define what bipartite means and then we will see how to apply breadth first search. Recall that a graph can be in many pieces, so I could have a graph which looks like that. How many connected components does this graph have? [Student: five] five [Hindi] connected components. What we want to do is have a procedure which will label these vertices. It will label every vertex here 1, say that this is in the first connected component. I have just arbitrarily called this as the first connected component. It will label every vertex here 2, it will label every vertex here 3, it will label this vertex 4 and it will label this vertex 5.

I want a procedure which returns this labeling of the vertices. I will have an array called label let's say and for each vertex, so this is the index corresponding to vertex v. At this location I should have 1, 2, 3, 4, 5, 6, 7 whatever is a number of connected components that number should appear here. So that by looking at this array, given any two vertices I can just in constant time determine if they are in the same connected component only. If they have the same label then that means that they are in the same connected component,

if they have different labels they are in different connected components. Question is how will I do such a thing. This is very useful procedure. First it is also counting the number of connected components your graph has. How will I do this, is the question that we are going to… [Student: do the, for each vertex and if the vertex are maintained one which counts vertexes which already have been traversed].

The standard things in any case in the first slide they said it is an application of breadth first search. The first thing you are going to do is to do some breadth first search somewhere [Hindi]. There is a notion of a starting vertex that we take one starting vertex and then we start doing our breadth first search from there. Suppose I take this as my starting vertex and starting doing the breadth first search, so what's going to happen? I am going to give each vertex a… so there also I was giving each vertex a label, a distance. Let's call that a distance label and let's call this component number.

There what I am going to do is all the vertices that I visit, if I do a breadth first search starting from this vertex, which all vertices am I going to visit? Only the vertices in this connected component. Is it clear to everyone? In the very first step, in the first round what I am going to do is I am going to look at all these three adjacent vertices and put them into the queue. These will get visited then when I take this out of the queue then I will put this into the queue and so on. I will end up putting all of these vertices into the queue, removing them and so all of these vertices will get visited. But I would not visit any of these vertices. I would visit only these vertices. As I am visiting these vertices, I can keep giving, assigning them a component number equal to one. I keep assigning whatever these vertices are let's say these vertices are some a, b, c, d, e and f and a appears here, b appears here and c appears here and d appears here and e appears here and f is here. I give each of these a connected component number of a one.

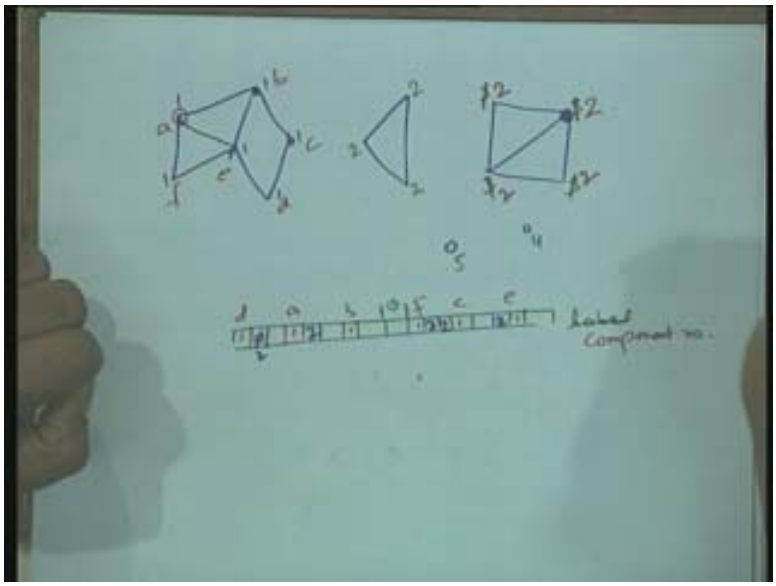
Now what should I do? [Student: implement the] initially what are the values that I give to each of these connected component number? Let's say initially everything is zero. Now after I finish this one BFS, some vertices have a connected component number of one and some others have connected component number of zero. Which are the ones which have zero? All of these. And which have a one? These. Now what should I do next? [Student: pick any of the] pick any vertex with zero, easier set then done. How do I pick any vertex with zero? I just start from here and find the first place, first vertex which is a zero. Then what do I do? Then I, let's say this was the first vertex which was at a zero, very first.

This is let's say this vertex here, so I start a BFS from here. [Hindi] there might be a two here, this would get label two, there might be a two here, this might also be two, this might also be two something like that. Now find the next zero. How do I find the next zero? Start from, so this is where the small thought is required. I should not start again from the end, start from where I found the last root vertex and continue from there, continue from the next location. Then once again when I find a zero, I start a breadth first search from there. Once again that will end up labeling certain vertices. [Hindi]

Once again I find the next zero and so on and on. This will at the end do what we wanted it do to. Question is how much time does it take? How much time did this breadth first search take? Number of edges plus the number of vertices. Of course number of edges is more than the number of vertices. Why? It's at least n minus 1. If a graph is connected then there is a spanning tree in the graph. If there is a spanning tree, the spanning tree itself has n minus 1 edges in it. [Hindi] Is the total turning time equal to order number of edges? [Student: order number is total vertices] Is this the right number? [Student yes] yes [Hindi] number of edges can be much smaller than the number of vertices in this entire graph.

The graph could have zero edges [student: suppose] so we have not counted the time required for traversing this array to find the next vertex from which to start our breadth first search. How much time does it take to traverse this array? Order n exactly, order n critically because we are not going back from the starting when searching. [Hindi] If we start always from the beginning then it will be order n times the number of components [student: only once] [Hindi] we have just made one scan of this array that's critical. The time required for that is order n.

(Refer Slide Time: 07:30)



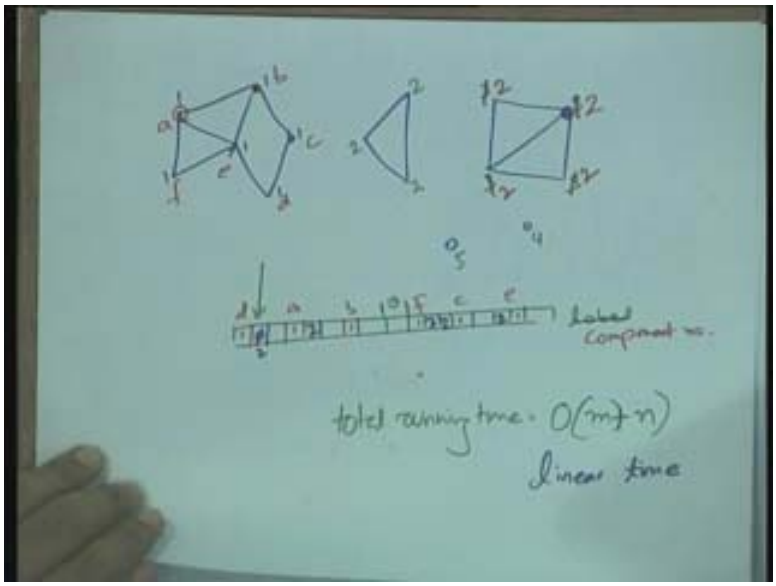
The right time bound should be order m plus n. You understand why am saying m plus n here and not just m? It is not the case that m is more than n, it is not either the case that n is more than m. I can't say that because this is not a connected graph over which I am doing the breadth first search. So this would be the right thing to say or if want to say maximum of m, n that is also okay. But just saying order m or order n is not correct. [Student: when you know a you started breadth first search then how do you label a without label b] No, I started my breadth first search from here. When I started my breadth first search from here let's say you are saying I want to start a breadth first search from here suppose I started my breadth first search from here. Then I looked at its adjacent vertices. These b, e and f were its adjacent vertices. These vertices I am going to

put them into the queue, at the time I put them into the queue I am also going to go to the location corresponding to the these vertices in this array and mark them one. [Student: sir but without like you are saying we go through the array work]

We go through the array only once for selecting the root vertex but for each one of these vertices, when I am visiting each of these vertices in my breadth first search, at that point I am going to go that particular location in this array and label that vertex. [Student: you are not counting the time] How much is that time? Constant for each of these vertices. So again number of vertices is n so again I would take order n time for that. For each one of these vertices, in any case I am spending a constant amount of time because I am putting this vertex into the queue, I am removing them from the queue. I am actually going to give them a distance label also, all though that distance label is not really required but I am going to access this vertex. I am going to spend a constant amount of time. So that in some sense, in that constant amount of time also accessing this array and updating that information here. [Student: sir is the location where the next element like k where you start way b is it know directly]

When I start from A, how do I know what b, e and f are? So think of b, e and f as numbers. When I look at the adjacency list representation of a, I have the adjacency list of a. I know what the 3 vertices adjacent to a are, they are let's say vertices numbered b is 2, 5 and 6. These are vertices number 2, 5 and 6. I will go to locations two, five and six in this array and make them one. You can just have a direct correspondence as numbers, if that's how you like to think of it. Clear to everyone? We can determine connected components in so much time, order m plus n, linear time. This is also called linear time.

(Refer Slide Time: 13:47)

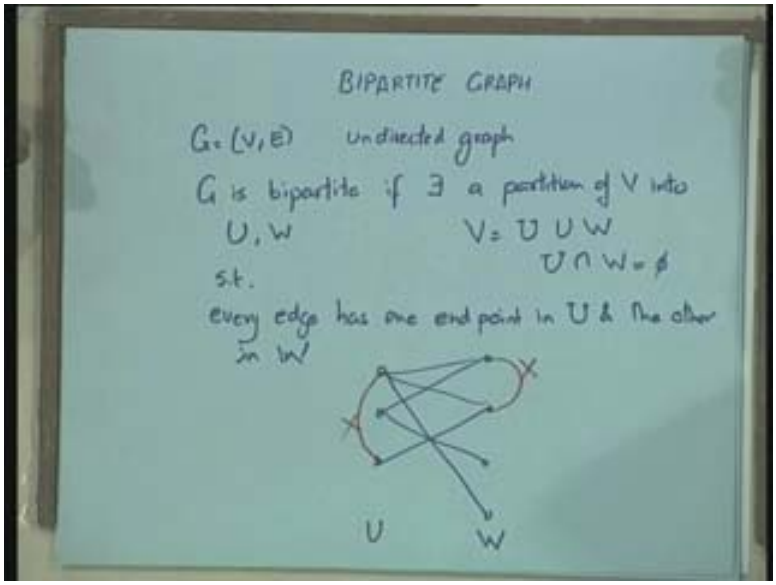


We have used this term before. I can remind you such an algorithm is called linear time algorithm. The next application we are going to look at is for what is called bipartite graphs.

First let me define for you what is a bipartite graph is. We are given a graph G, so recall a graph is given by a set of vertices and set of edges. We are talking of undirected graphs here. G is a bipartite graph if there exist a partition of V into U, W partition. [Hindi] What is partition mean? V equals U union W, U intersection W equals null. I can divide the set of vertices into two pieces, U and W such that every edge has one end point in U and the other in W.

Let me draw an example. [Hindi] this is an example of a bipartite graph. Every edge has one end in the set U and the other end in the set W, the other end point. That means there is no edge like this or like this. These edges are not there. There are no edges both of whose end points are in U or both of whose end points are in W. [Hindi]

(Refer Slide Time: 16:27)



Such graphs actually model a lot of things. One standard setting is these are boys, these are girls and these reflect their, whether they like each other or not. No, we are not saying they are getting married this way. We just trying to find out if they can get married or these could be jobs and these could be applicants to jobs, yes and a certain applicant is suitable for certain subset of jobs and you want to know whether for each job there is an applicant or for every applicant there is a job some such thing. Such graphs find a lot of applications in these kinds of settings.

Question, given a graph can you determine if it is a bipartite? Everyone understands what the question is. Given G is G bipartite, so suppose you said the answer is yes. Yes, G is bipartite. How will you convince me it is bipartite? [Student: no edge between the same lines levels with] He has a suggestion. What he says is do a breadth first search, after all that's the topic of this class. Do a breadth first search from which vertex? Any vertex. Let's do a breadth first search. I said starting from an arbitrary vertex, just to make sure so that you understand that there is nothing sacrosanct about the starting vertex [Hindi].

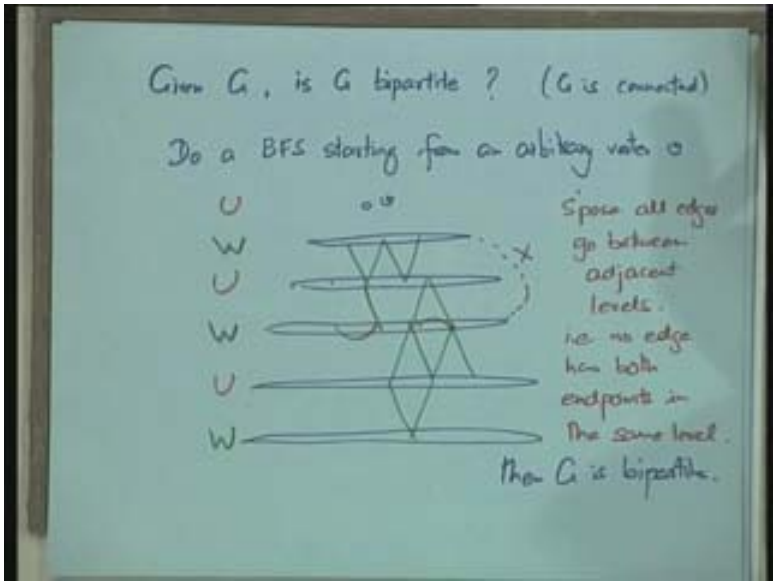
If you recall, breadth first search will divide the graph up into layers into levels. These will be the vertices at level 1, level 2, level 3, let's say level 4 and there were 5 levels. There was one other interesting thing that we have talked about as regards breadth first search. All the edges of the graph, we are not talking of the breadth first search tree. We are talking of the original graph. In the original graph there will be edges which go between adjacent levels or within a level. There will be edges of this kind which go between the adjacent levels and there might be some edges which go within a level. There would be no edges which jump levels, there cannot be an edge which goes like that. [Hindi] You should not confuse it with anything else. Everyone follows this. These brown edges are not there.

We have only the green and the red edges. When is this graph bipartite? Suppose these red edges were not there. Is this graph bipartite? Suppose there were no red edges. Which are the red edges? These are the red edges here. What do I mean by no red edges? There are no edges, suppose all edges go between adjacent levels. [Hindi] i.e. no edge has both end points in the same level. Suppose such was the case, what can you say? This implies the graph is bipartite. Does everyone see why? [Student: componental graph] [Hindi].
I am assuming that let's assume that it is a connected graph to simplify matters. Given G is G bipartite so let's assume G is connected. [Hindi] (Refer Slide Time: 21:50). We will have to check it for each connected component. If each connected component is bipartite then the graph is bipartite. If some one connected components is non-bipartite then the graph is not bipartite. I will come to all of this in a second.

For now let's assume we have given a connected graph and we have to determine if it is bipartite or not [Hindi] (Refer Slide Time: 22:14). Edges which go within the level. The claim is that the graph is bipartite now. Why? Because now you are going to take alternate levels on the u side [Hindi]. This is the bipartition. After all to show you that to convince you that the graph is bipartite what do I have to do? I have to show you a partition of a vertices such that every edge is going between one side, has its end point on the two sides of the partition.

Now if I do my partition in this manner [Hindi] then every edge is going from a vertex on the u side to a vertex on the w side. Yes, if these red edges were not there. But if these red edges were there then we can't say because then such an edge is going between a vertex on the w side and another vertex on the w side. If this were the case, all edges go between adjacent levels that is no edges has both end points within the same level then G is bipartite. But we have not solved the problem here. Suppose there is an edge which has both its end points in the same level. What then?

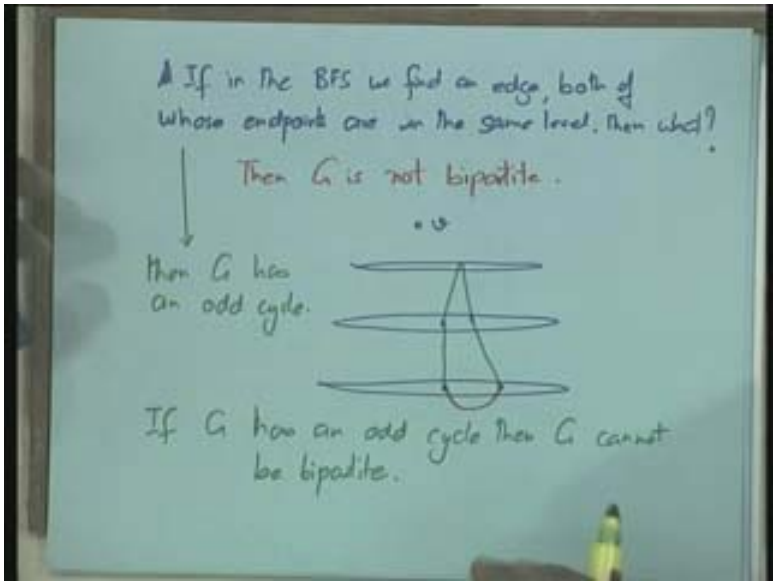
(Refer Slide Time: 24:09)



Suppose such an edge exits, does that mean that the graph is bipartite? You will not solve the problem. [Hindi] If in the BFS, we find an edge both of whose end points are in the same level, then what? What can we say then? Life cannot be too hard. Then what? Basically we have to say, the graph is not bipartite. Then G is not bipartite but why? [Student: you have partition between u and v then the vertex you started this BFS must be in some partition let's say u then this would definitely contradict because the next level must be in v, the next level must be in u. Ultimately it will contradict that] Proof is correct. I will give you a different proof though because it will bring out another aspect, another property of the bipartite graphs.

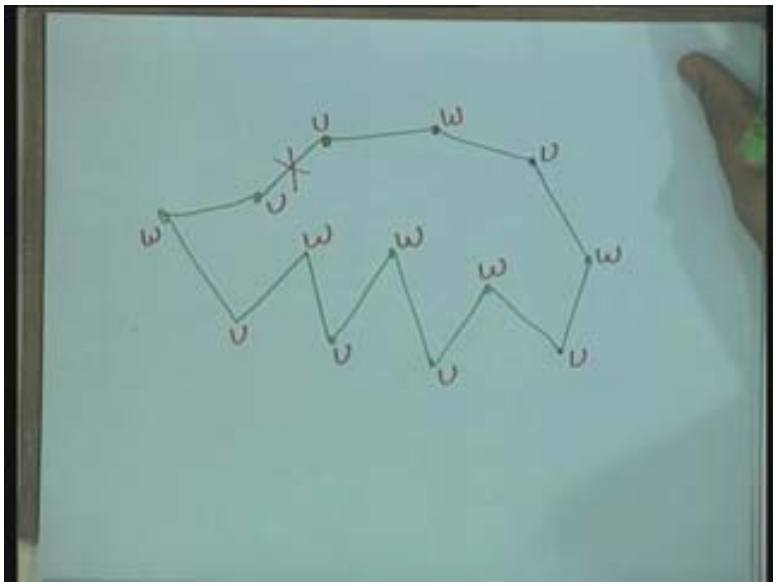
Suppose we had such an edge. We had an edge [Hindi] both of whose end points are in the same level [Hindi]. Now recall we had a notion of a breadth first search tree [Hindi]. What was the notion of a breadth first search tree? We said there is a tree and basically for each of these vertices, there is a predecessor. For each vertex there is a predecessor except for the starting vertex, the root vertex [Hindi]. Is this [Hindi] if such is the case then G has an odd cycle [Hindi]. Let's prove that fact now. If G has an odd cycle then G cannot be bipartite. I have not proved this fact, I will prove it. But if I prove it then we are done. [Hindi].

(Refer Slide Time: 28:29)



So proof by contradiction. Suppose the graph is bipartite what will happen? Suppose this vertex either it's on the u side or it's on the w side? Suppose it's on the u side then this vertex has to be on the w side. Then this has to be on u side, this has to be w, u, w, u, w u, w, u, w, u, u. [Hindi] I started with the u but this could easily have been a w and you would have to adjacent vertices which are both labeled w.

(Refer Slide Time: 29:45)



Everyone follows this proof. Let me show the previous slide once again. If the graph has an odd cycle then it cannot be bipartite and if we found an edge, both of those end points are in the same level then we have shown it has an odd cycle. This is the odd cycle in the graph [Hindi]. This is the general proof, this is not a proof by example because [Hindi] everyone follows this (Refer Slide Time: 30:20). In this manner you can use breadth first search to check if the graph is bipartite or not which is equivalent to checking, if the graph has an odd cycle or not.

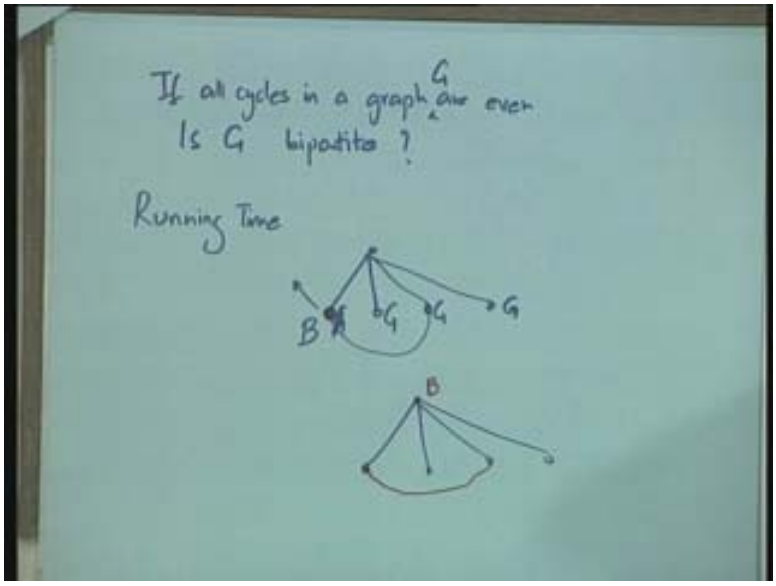
If the graph has an odd cycle then it cannot be bipartite. If the graph has no odd cycle then does it mean, it is bipartite? If all the cycles in the graph are even in length then does that mean that the graph is bipartite? [Hindi] My next question is if all cycles in a graph G are even. Is G bipartite? Yes or no? [Student: yes] Once again take your graph, do a breadth first search on the graph [Hindi] which is between the same level then there is an odd cycle. [Hindi] By taking alternate levels on one side, you have shown it to be bipartite. Yes?

How much time did our procedure take? The procedure to check if the graph is bipartite. How much time did it take [Hindi] and then what did we do? We did also something else. We checked if there was an edge both of whose end points were in the same level [Hindi]. Let me write [Hindi]. We will look at all the adjacent vertices then when I come to this vertex, what are the various things we were doing? We looked at all of these and we put them into the queue and we gave them a certain color. What was the color we gave them? Grey, so each of these vertices were colored grey. Then I removed one of the vertices from the queue. Let's say I removed this vertex from the queue and I looked at its neighbor's. If any of the neighbor's is grey then we can stop the procedure [Hindi] (Refer Slide Time: 34:18). If the neighbor is white, grey or black [Hindi].

I will repeat. All you need to do is to check if the other end point of the vertex is grey or not. Just grey, if it is grey stop, exit, reboot. The graph is not bipartite. And that's the only thing we need to check. Why is that? The reason for that is if the other point is black then that does not mean that the graph is not bipartite because you could have such an edge.
When I am looking at, so let me draw this picture again because this is getting cluttered. When I am looking at this vertex, then it has its neighbor's one of which is this, which is already colored black. I cannot use black, the other end point being black has a test for non bipartiteness. Because this could be the entire graph for all I care. This is clearly bipartite. But to say that this vertex, when I am looking at this vertex it has a neighbor which is black, use that to say it is non-bipartite, would be wrong.

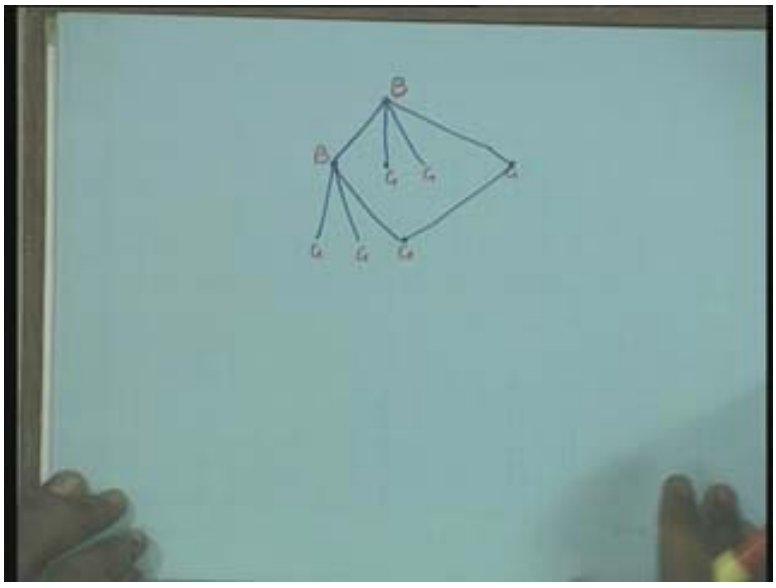
Now what we are saying is that suppose there is an edge like this. When there is an edge like this, then when I am considering one of its end points for the first time, then the other end point; since it has not been considered yet, would be colored grey. Yes. Since it would be colored grey, we would be able to identify that edge as an edge which is running within the same level. [Hindi] They will also get into queue. [Student: (Refer Slide Time: 36:55)] here is a valid point. I made a mistake. Does everyone understand his question? No. let's look at this picture for now. [Hindi]

(Refer Slide Time: 36:39)



What is the point? Depends upon me where the point is. This is the thing that has been. So this should be black clearly. This should be black because it has been expanded out and then since it has been expanded out all of these guys are in the queue. They are all grey. What are the color of these guys? Because I have not touched them yet, they are in the queue. They are grey. Now who can tell me, what I am trying to do? If now suppose I look at this vertex next and this has an edge going here. The other end point of this edge is grey, so I might say it is not bipartite so that is not right.

(Refer Slide Time: 39:24)



Simply level [Hindi]. We are keeping level numbers so that quantity is important. That level number business is important. We are keeping the level numbers of the vertices. If the level number is the same, say the graph is non-bipartite. If the number level is different, continue. Clear? One thing I wanted to do today which I think, I did not do very well in the last class was to argue that the level number of the vertex that your breadth first search procedure is giving you the shortest path. Some of you perhaps understood it. But I think I did not do it very well.

Let's do it once since we have some time today. Let me write down the formal statement. what I want to argue is that in a breadth first search, starting from vertex v, the level number of vertex u is the length of the shortest path from v to u. What are we saying? We are saying that I started a breadth first search from vertex v, I got a bunch of levels. Let's say vertex u is sitting in this level 4; level 1, 2, 3, 4. Then the shortest path… what do you mean by shortest path? Let's define what shortest path means. You all understand what a path is. So between v and u, there could be many paths in the graph. It could be very complicated. Each of these paths has a certain length. What is the length of the path? It's just the number of edges on the path, let's say.

For now we will just keep that as a definition. Later we will modify this definition. So for instance if I were to look at this path, this has 1, 2, 3, 4, 5, 6; 6 edges on it so it has a length of 6. If I had gone like this, it was even longer, it had length 7. If I had gone like this 1, 2, 3, 4, 5; this had length 5. But there is also a path of length 4, this is 1, 2, 3, 4. It is not an important here exactly which it is but you understand that there could be many different paths in a graph between two vertices. The shortest path is just the path which has the least number of edges on it.

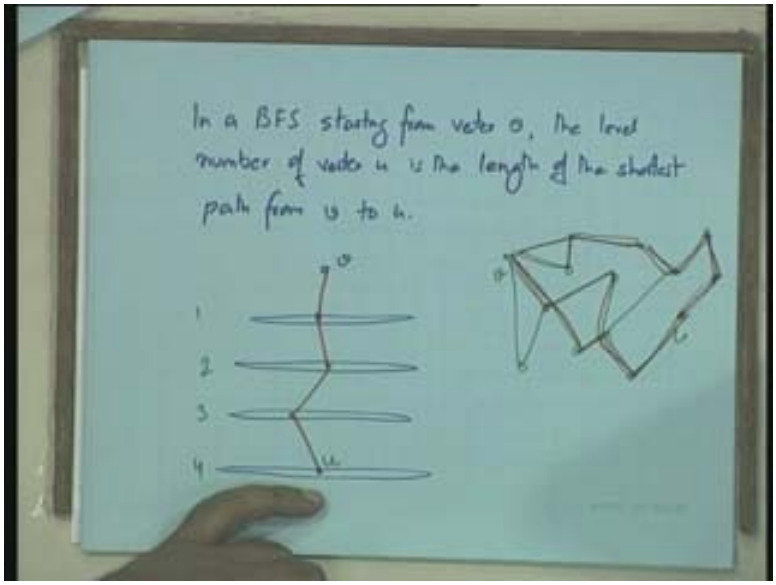
It is relevant clearly because if these are certain roads or some such thing, you would like to travel the minimum possible. Although I have not kept any distances here but let's say these just reflect the number of hops that you are taking. Say suppose this is some computer network, you want to talk between v and u. Let's say most of the time is spent in a, when you have to go through one node. This is the link between this computer and this computer, this is link between this and this is the link between this and this. So information travels very fast on link. But at a node, at a computer it has to be processed and forwarded and stuff like that. There is a lot of time wastage here. Clearly you would like to minimize. This is called the number of hops then. This path has 6 hops on it. So you would like to pick a path which has the smallest number of hops because then you are information travels very quick. You would like to find the shortest path.

The claim is that if this vertex u is at level four then the shortest path from v to u is of length 4. First let's show that there is a path of length 4. How do I show there is a path of length 4? Very simple. I start from u, I take its predecessor. Its predecessor would be some vertex there. Then I take its predecessor. Its predecessor would be some vertex at the previous level. You understand what predecessor means? Let me recall. Predecessor is the vertex which gave due to which this guy got its label.

Clearly that is the vertex sitting here. The predecessor of this and I would get to the predecessor of this and clearly the predecessor of all of these vertices at the first level is the root vertex itself. What is the length of this path? It's just the number of levels and its four. There is a path of length 4. Can there be a path of length 3? [Student:] If there is a path of length 3, then we will have to jump a level which violates the fact that this is a breadth first search. This is a partition it gives by breadth first search. [Hindi]

You understand? If there is a path of length three, then that path cannot visit all these. It has to come here, it has to start from here. But then it cannot visit all these 3 levels. It has to jump over one of these. But if it jumps a level then it's a path, then it violates our breadth first search property. That shows that this process gives you also the shortest path and the shortest path is just the distance, the level number of each vertices. If I wanted to find the shortest path from a certain vertex, all I have to do is do a breadth first search and the level number would give me the length of the shortest path.

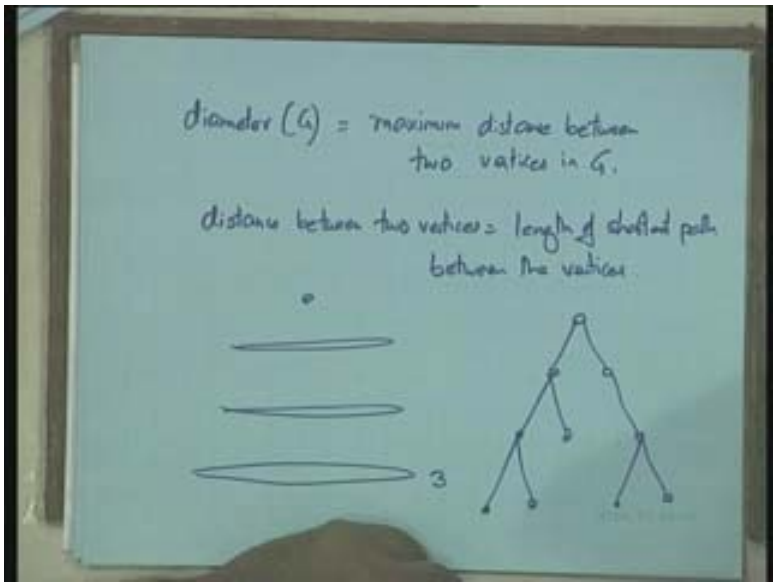
(Refer Slide Time: 45:33)



Now one last thing I want to do with you is again applications of breadth first search. It's a very simple application using this fact that we just understood. I give you a graph and I define the diameter of a graph. What do you think the diameter of a graph should be? Maximum distance between two vertices along the shortest path between those two vertices. Maximum distance between 2 vertices in G, let's just say that. What does distance between 2 vertices means? Distance between two vertices equals length of shortest path. We are not saying that the diameter is the length of the longest path between two vertices. That is a very hard quantity to compute. That's impossible to compute, more or less. We are not talking of that. Diameter is very specifically defined maximum distance between two vertices and the distance between two vertices is the length of the shortest path.

How am I going to find the diameter of the graph? [Hindi] What is the answer? BFS, just do a BFS and let's say the largest level number is 3. The diameter of the graph is 3, that's not true. We will have to do a BFS on every node and then find the maximum level. You understand why this is wrong? Does everyone understand why this is wrong? Because I could have a graph which looks exactly like this. Suppose this is my graph, so it's exactly this. If I did a BFS from here, I would get exactly level 3. But the diameter of this graph is not 3. The diameter of this graph is 6. The diameter is 6 because it is the maximum that we are interrupting. So to get the diameter, I would be able to get to the diameter if I started my BFS either from this vertex or from this vertex or from this vertex or from this. You started from one of these, then the maximum level number would have given me the right information. But I did not know which these vertices are, so I will have to try it out for every vertex. How much time would determining the diameter take? MN time. We are assuming that the graph is connected.

(Refer Slide Time: 49:22)

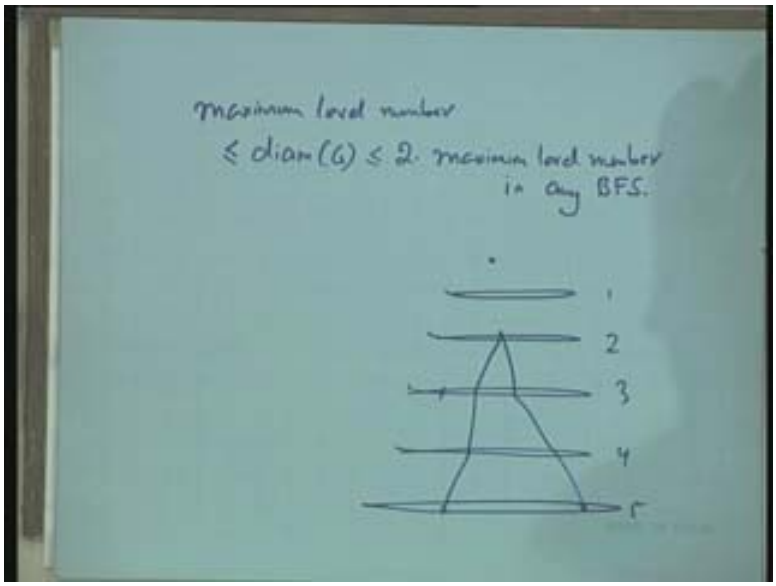


So it will take, why because each of these breadth first search will take M time and we are doing N. For a disconnected graph it is not defined, let's say it is infinity. For each connected component you can define the notion. But if I do a depth first search from one vertex and I look at the largest level number, it doesn't tell me what the diameter is exactly. But it does tell me approximately what the diameter is. It can be at most half the diameter, the maximum level number is at least half the diameter. Yes. The diameter cannot be more than two times the maximum level number for any BFS. You understand? Diameter of the graph is going to be less than or equal to two times the maximum level number in any breadth first search. You understand why? [Hindi]

The claim is diameter cannot be more than 10. No 2 vertices are more than 10 units of apart. Why? Take any 2 vertices. [Hindi] Since this is true for every pair of vertices, the diameter cannot be less than 10, cannot be more than 10. Diameter is less than 2 times the maximum level number and diameter is greater than… what is the diameter greater than?

Maximum level. Diameter we know is at least 5. Why? Because there is this vertex here and one vertex here [Hindi]. So by doing one BFS, you can get an approximation to the diameter. [Hindi] You know the diameter is at least 6 and at most 12 [Hindi].

(Refer Slide Time: 52:46)



It's clear to everyone? So with that we will end today's discussion on breadth first search. In the next class we are going to look at depth first search.