

Complex Network: Theory and Application.
Prof. Animesh Mukherjee
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 13
Community Analysis – II

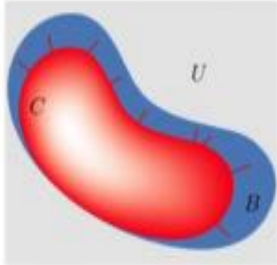
Welcome back to this session on Community Analysis. Last day we have already defined what a community means, both philosophically as well as we have tried to give a quantitative definition also. And then we have seen that there are various techniques by which you can do community detection, we have been looking into agglomerative techniques.

Then as we proceed in the next few lectures we will see other different methods like the Bisection method and the Spectral bisection methods for doing community analysis.

(Refer Slide Time: 00:56)

Local algorithm based on agglomeration
Bagrow, J. Stat. Mech., 2008, also: Bagrow, Bollt, Phys. Rev. E, 2005

- Agglomerate nodes one at a time
- Maintain two groups, the **community C** and the **border B**
- nodes in **B** have been explored but are not yet in **C**
- Move nodes from **B to C** in specified order, **Outwardness** (Ω)



So, today we will start off with a Local algorithm based on agglomeration. Have a look at this slide, so basically this algorithm was proposed by Bagrow in 2008. Now the basic idea of this algorithm is that in a network you actually keep track of two sets; one set is the set C which is the community, and the other set is the set B which is the border from

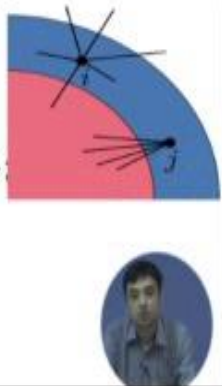
where new nodes enter into the community. And in this way at every iteration you actually agglomerate. Now, the decision of putting a node from the border to the community is based on a metric called Outwardness.

Basically, this is a local metric. So you look at the local properties of the nodes that are in the border and then based on that you make a decision whether you should include that particular node in the community or not. And in this way you keep on including newer and newer nodes in the community and the community grows. And then finally, you can keep track of this whole process and draw the agglomerative picture of the community structure in the form of a dendrogram.

(Refer Slide Time: 02:19)

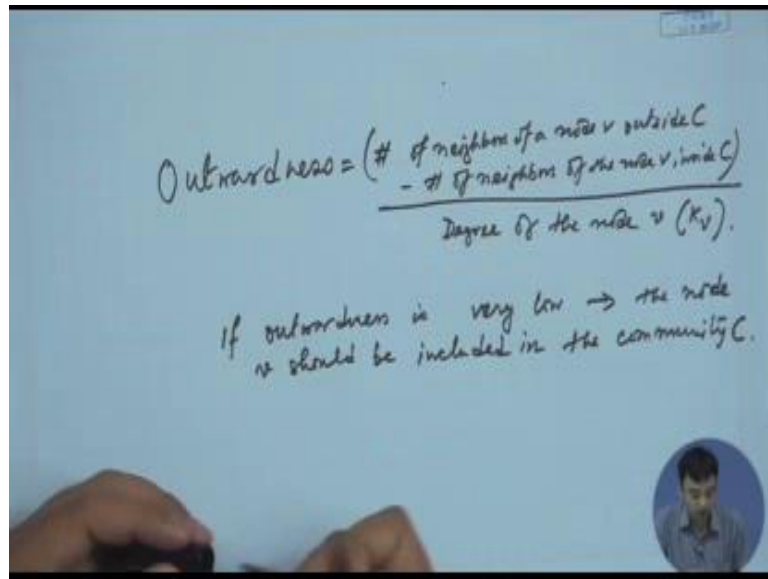
Local algorithm based on agglomeration
Bagrow, J. Stat. Mech., 2008, also: Bagrow, Bollt, Phys. Rev. E, 2005

- $\Omega_v = (\# \text{ of neighbors of } v \text{ outside } C - \# \text{ of neighbors of } v \text{ inside } C) / k_v$
- $\Omega_i = 2/3, \Omega_j = -1$
- Algorithm
 1. Choose starting node s : $C = \{s\}$, $B = \{\text{neighbors of } s\}$;
 2. Add $v \in B$ to C , where $\Omega_v = \min\{\Omega\}$;
 3. Update B, Ω 's, repeat from 2;



Now the first that we need to do is to define these local metric of outwardness. So how do we define this local metric? This metric is defined as follows.

(Refer Slide Time: 02:30)



Outwardness =
$$\frac{(\# \text{ of neighbors of a node } v \text{ outside } C - \# \text{ of neighbors of node } v, \text{ inside } C)}{\text{Degree of the node } v (K_v)}$$

If outwardness is very low \rightarrow the node v should be included in the community C .

You basically compute this as the number of neighbors of a node v outside C minus the number of neighbors of the node v inside C , this difference divided by the degree of the node v which may be denoted by k_v . Basically, you see what is the fraction of edges from the node v that is going inside the community, and what is the by fraction of edges that is lying in the border. And you make a difference of these two and you express these are the ratio of the total number of edges that go out of the node v that is the degree of v . So you express this as in terms of this ratio. Therefore, if outwardness is very low then that would mean that the node v should be included in the community C . So, that is the basic idea.

Now let us look at the slides. Here, for ease of understanding I have put a very simple example. Let us consider these two nodes; node i and node j . And we can compute the outwardness score of each of these nodes. For instance, for node i the number of edges that are going inside the community is only 1, whereas that are outside the community are 1, 2, 3 and 4, so you make a difference of 4 minus 1 that is 3 by the total; sorry 1, 2, 3, 4 and 5, so the number of edges going out is 5, whereas the number of edges going inside the community is 1. So you make a difference 5 minus 1 that is 4 that is normalize by the total degree of the node i which is 6 in this case. So, that makes the outwardness score as 4 by 6 which is 2 by 3. So, I write ω which is a notation for outwardness

score, ω_i is equal to 2 by 3 for this case.

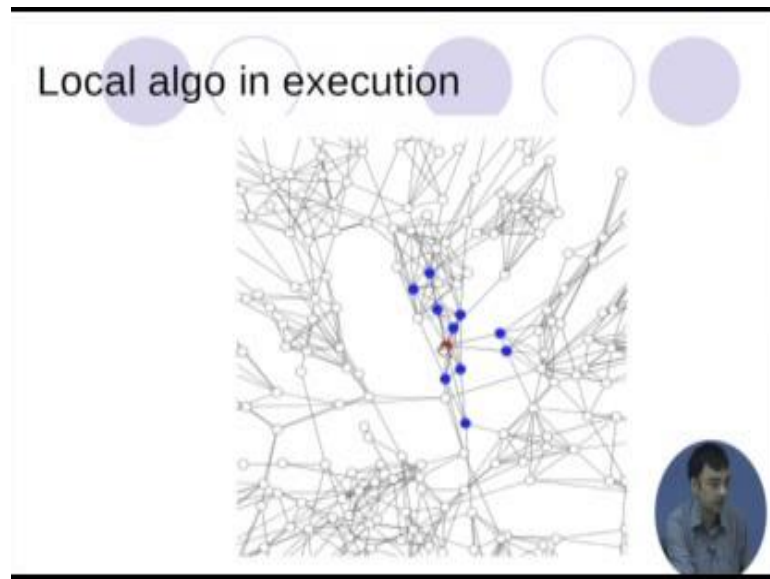
Similarly, you can compute the outwardness score for the node j . So, for node j you see all the edges are going inside the community. Basically, you can also intuitively understand that node j is a much better candidate to be inserted in the community than node i , because all its edges are going inside the community. Here, if you calculate the outwardness the degree is equal to 1, 2, 3, 4, and also the total number of edges going inside the community is 4. And there is no edge going outside the community, so it is 0 minus 4 by 4 which is minus 1.

Therefore, the agglomerative (Refer Time: 06:01) algorithm is summarized in the next three steps of the slides. In the first step what you do, you choose a starting node you arbitrarily choose a starting node s and you put this s in the community C . Now B is basically the neighbors of s , B which is the (Refer Time: 06:21) set of border nodes is nothing but the one distance neighbors of s .

Now you inspect every individual node from B , say you inspect every individual node v from B , and you check among all these nodes in v which is the node that has the minimum outwardness score. And depending on that the say if it is the node v that has the minimum outwardness score, for instance in this case considering these two nodes j has the minimum outwardness. So, you include j inside the community.

Accordingly you update the community set. So now, your community will become s comma j , whereas B will be the neighbors of s plus the neighbors of j . And now you repeat the computation of the outwardness and you keep doing so unless and until you have included all the nodes in the network. In the next slides I show you a simulation of this process.

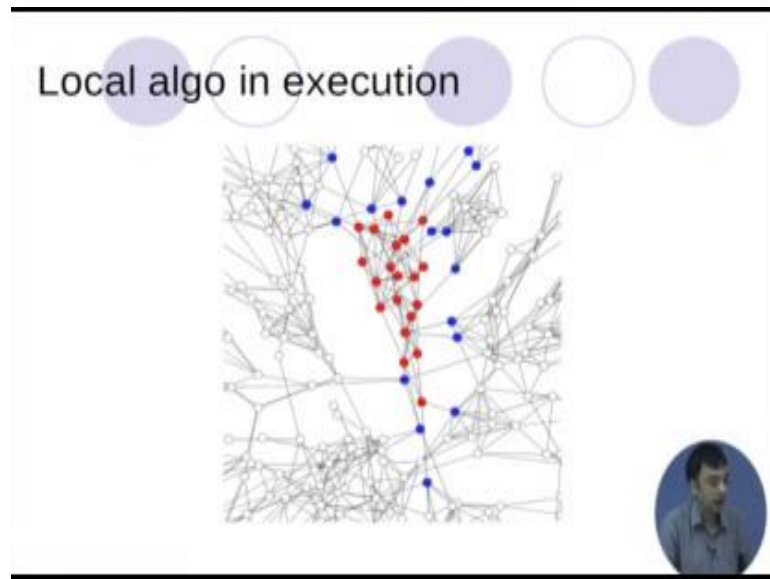
(Refer Slide Time: 07:25)



So, let us take this example network. The red node that you see at the center is basically the only node in the community in the first step and all the blue nodes are its one distance neighbors. Now from this step if you execute this algorithm, in the next step you have 1, 2, 3, 4, 5 nodes which get inside the community because of their low outwardness score the other nodes become the border node. And you keep on doing this continuously until and unless you have actually covered the entire network, you have actually subsumed all the nodes in the network.

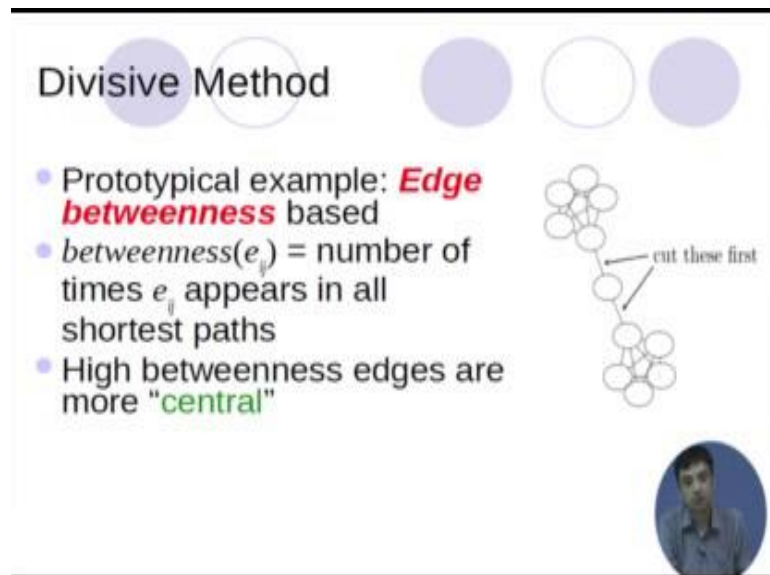
And in this way after the process is complete you know exactly, what are the nodes that got inserted into the community one by one and then from this information you can actually construct the dendrogram, which gives you a fantastic notion of the community structure.

(Refer Slide Time: 08:22)



So, this is the second example of agglomerative approach based on local measure of outwardness.

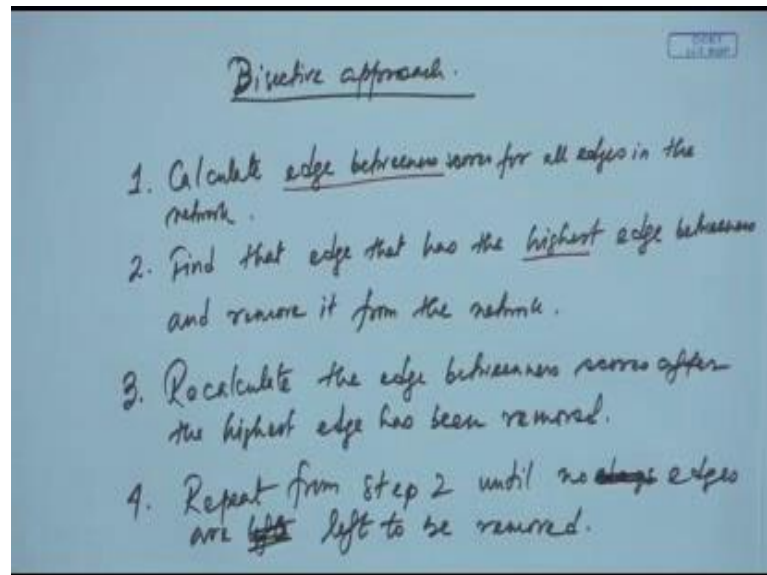
(Refer Slide Time: 08:35)



The third approach that we will look into will be on based on the idea of Bisection. So far we have only looked into agglomerative approaches now we will look into one

Bisective Approach.

(Refer Slide Time: 08:43)



As I already told you earlier that in the bisective approach what you have is basically, you start with the full network with all its nodes and edges now you have to find out candidate edges which are actually standing at the border of say two communities, two or more communities and you basically cut these edges in order to decompose the network into structures or communities. The question here is therefore, how to identify edges that stand at the border of two or more communities, basically bridge edges. So how do I find out or compute bridge edges.

So, one of the classic examples was proposed by Newman and Girvan one of his students, so they proposed this idea of edge betweenness. If you remember we discussed the idea of betweenness centrality in our earlier lectures. We computed betweenness centrality by estimating what is the total number of shortest paths that actually passed through that node between any pair of vertices by the total number of shortest paths between that pair of vertices. That was the basic idea of betweenness centrality.

And actually if there is a node in the network through which a large number of shortest paths passed then, that means this is like an articulation point in the network or a cut point

in the network. That is like it actually connects two basic. So, that is what we saw earlier that it connects probably one or more modular components of the network. That is the only individual node in the network which actually connects multiple paths of the network.

Here, we actually translate that idea of betweenness but now to edges rather than looking into nodes we now look into the number of shortest paths that actually passed through a particular edge rather than a node. Now if there is a large number of shortest path passing through a particular edge then probably that edge is actually sitting as bridge edge between various components. So that is the basic idea. So, two components get separated by a edge if the edge betweenness of that particular edge is high, that is the hypothesis. Based on this hypothesis Girvan and Newman actually proposed a very simple algorithm.

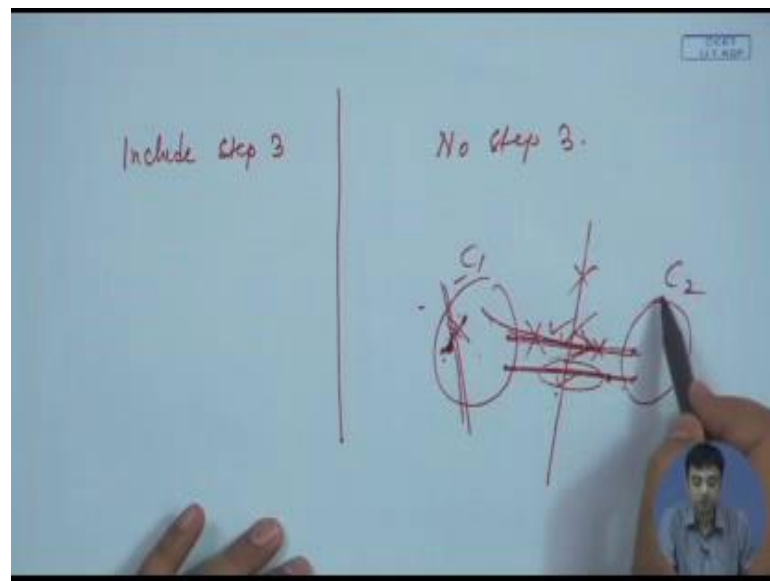
The steps of the algorithm can be summarized as follows. So step 1 is, calculate edge betweenness scores for all edges in the network. The next step is to find the edge that has the highest edge betweenness and remove it from the network. The third step is to recalculate the edge betweenness scores after the highest edge has been removed. And number 4 is repeat from step 2 until no edges are left to be removed. It is a very simple algorithm if you see. It comprises 4 steps; the first step is actually you calculate the edge betweenness for every individual edge in the network.

Now you rank the edges based on the edge betweenness and remove the edge which has the highest edge betweenness, then you again after you have removed that edge you again recalculate the edge betweenness of all the other edges in the network and you repeat the same process from the step 2 until and unless you have no edges left. In this way if you think intuitively you are actually bisecting the network iteratively and dividing it into smaller and smaller sub paths. And if you keep track of this execution you can again draw the dendrogram accurately.

So, now there is a question here. If you look at the third step this is actually the most expensive step of this process, it is one of the very, very expensive steps. In general if you have to calculate the edge betweenness you have to actually calculate the shortest

path between every pair of nodes in order to calculate the edge betweenness for individual edge. Now that itself is time consuming. And on top of this we will actually compute the times but before that intuitively also on top of that computation you also have to re calculate this edge betweenness every time, but this actually is very important for the performance of the algorithm.

(Refer Slide Time: 15:26)



Why? Because, actually the authors did two types of two variations of the algorithm; in one where they had include step 3 and in the other they did not have the step 3, no step 3. That means, in the first case they were recalculating the edge betweenness after every removal of edge, whereas in the other version of the algorithm they actually calculated the edge betweenness once and then based on that they kept on removing the edges.

Now, these second method might be problematic. Because, say for instance there are 2 components like this and say this is c_1 and this is c_2 . And say there are actually two edges connecting them, but by some means what happens is that most of the shortest path between these two components flow through this particular edge, then this particular edge will have a high betweenness and it will get removed. But if you then do not recalculate the edge betweenness, if you depend on the older values of these the edge betweenness this edge will never be identified until and unless it is very late.

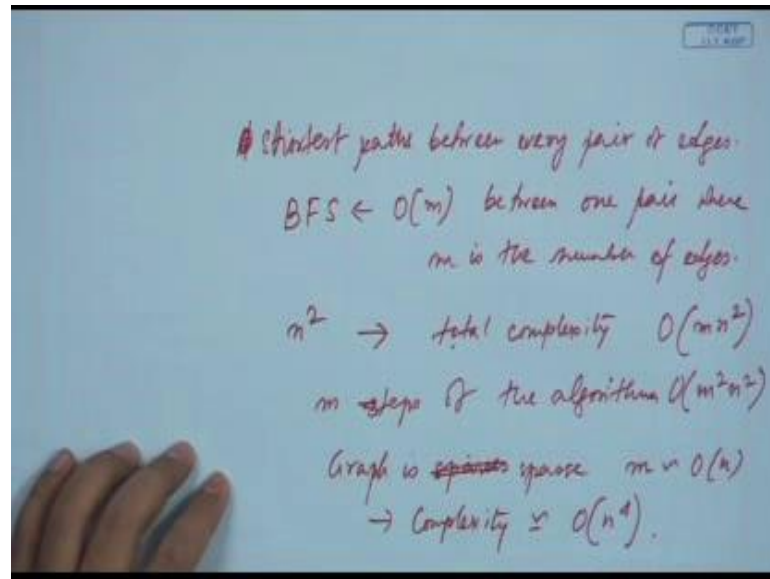
So you might not identify this edge and there might be some internal edges which have a higher betweenness somehow that would get removed even before this particular edge gets removed. So, more practical division of the network is missed even before you could actually identify it. Because, if you have a small edge like this inside one of the sub communities which actually divide, this sub community in further communities even before we could have discovered these two larger more logically correct partitioning, this is a problem.

So, the I iterate the problem once again; basically what happen is let us say that there are two edges that connect these two communities. Now fortunately or unfortunately most of the shortest paths tend to passed through one of these. So these actually rank high in edge betweenness and you comfortably remove it. But then the other edge never gets tracked, because you are not recomputing if you had you recomputed next time these edge would qualify as the edge through which most of the shortest paths pass, but this you cannot identify because you are not recalculating, and there might be some other internal edges which have higher betweenness than this particular edge earlier.

And that gets removed and that might fracture this particular component c_1 further even before we have identified a more logical partitioning between c_1 and c_2 . So, this is a very practical problem and that is why the process of recalculation is very, very important.

So now, if you look at the complexity of the algorithm this is a very, very time hungry algorithm, because see for every iteration you have to be calculate all pair shortest paths and that you have to do for a large number of iterations at least as long as there are no edges in the network. This is a very time hungry process. And the question is like if you have a very large draft say, billion of nodes even if you can make a little bit more efficient that actually helps. So for the next part of the lecture we will try to see how you can make the algorithm efficient by reducing some computation over it.

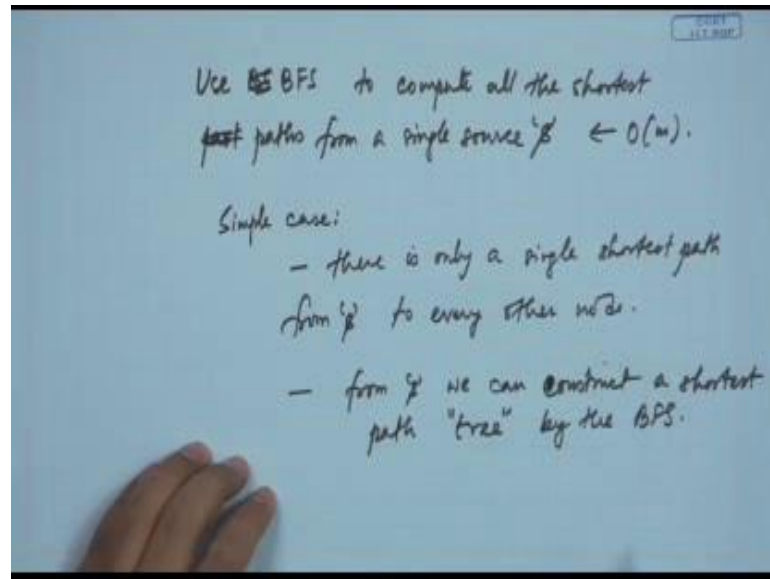
(Refer Slide Time: 19:42)



As we have said in general if you look at this algorithm you have to find out all pair shortest paths between every pair of edges. Now if you use say simple BFS algorithm to compute the shortest path between a single pair then, the BFS algorithm will be order m between one pair where m is the number of edges. If you n square such pairs then the total computation time the total complexity should be order of $m n$ square because there are n square such pairs. Now you have to run this iteration for m steps of the algorithm which would lead to a total complexity of order m square n square.

Now even if we assume that the graph is sparse in general as we have seen that real world network has sparse, so graph is sparse that would mean that m is roughly of the order of n . That means, in general the complexity of the algorithm is roughly equal to order n^4 . So, this is a very time hungry algorithm as you see. Therefore, in the rest of the lecture we will try to see how we can make this algorithm efficient. We will basically try to reduce this complexity to at least one order less and we will try to reduce it to order n cube.

(Refer Slide Time: 22:12)



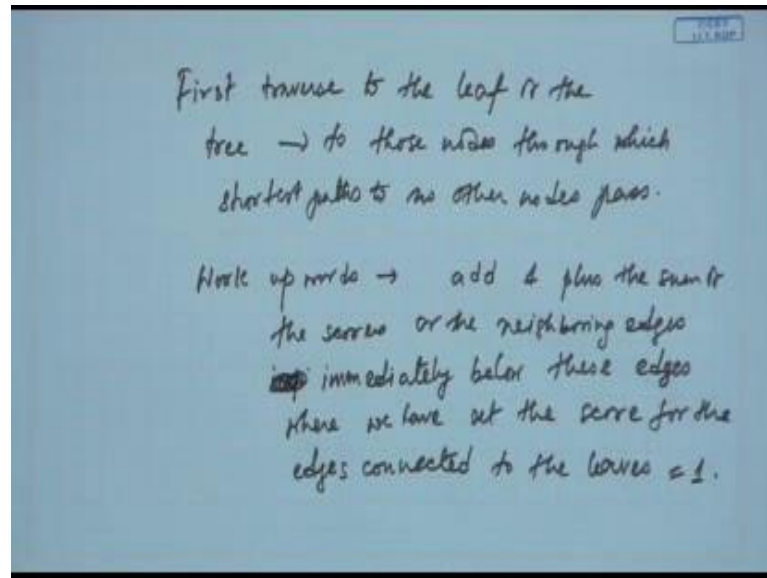
Now in order to do this we actually use the concept that, when you use BFS to compute all the shortest paths from a single source s . In one BFS you can actually find out all the shortest paths starting from a source node s , so this is the idea that we will use and this actually will take you order m time. And we will try to see how we can leverage on this in order to compute the edge betweenness centrality for every individual edge.

Now, we will start off with a simple case and then we will try to see the more general case. So this simple case is the following: so what you do is that, we assume here that there is only a single shortest path from s to every other node. Note that this is not true in general, but we will start off with this simple assumption and then we will try to make a generalized version of this. We will see how we solve this simple case and then we will see the more general case which addresses every possibility.

So, we assume that there is only a single shortest path from s to every other node. That means, from s we can construct a shortest path "tree". Since there is only a single shortest path, if we are assuming that there is only a single shortest path to every other node from s then we can construct, so this would actually result in a shortest path tree and construct shortest path tree by the BFS. Now given this shortest path tree the

question is how to calculate the edge betweenness. Again that is not very difficult. We will look at an example, but let us first write the algorithm. The algorithm is very simple.

(Refer Slide Time: 25:11)

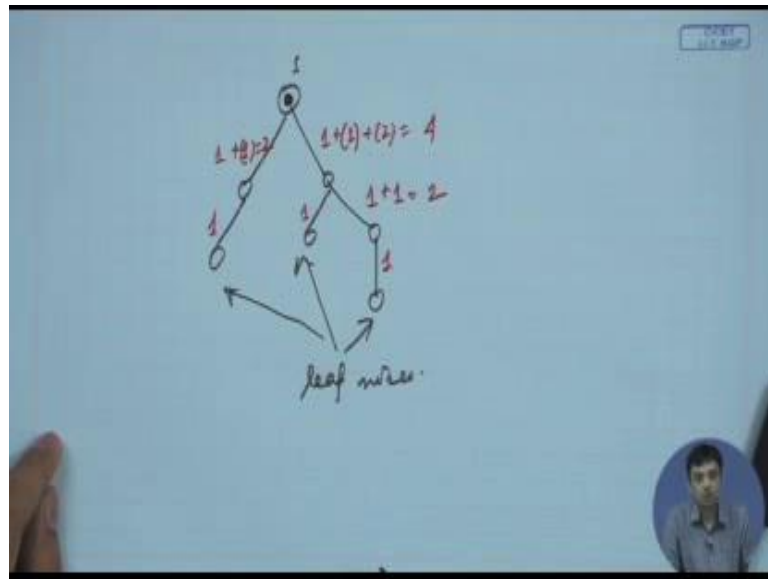


So, you first traverse to the leaf of the tree that is to those nodes through which shortest paths to no other nodes pass. You basically go to the leaf of this tree. The leaf nodes of this tree are those nodes through which no shortest paths pass through any other node in the tree. It is a very very simple definition of leaf nodes in a tree.

So now, from there we have to work upwards, and in every step what you do is you add 1 plus the sum of the scores on the neighboring edges immediately below these edges, where we have set the score for the edges connected to the leaves equal to 1. So, basically to the edges that are connected directly to the leaf we set a score to 1 and then we work upwards and at every upward step what we do is for every individual edge that we add 1 plus whatever score comes from the immediate lower level of the edges. In this way you compute these scores

And basically, if you see intuitively this course is nothing but the representative edge betweenness or the path counts, basically the number of shortest paths passing through that edge. So let us take a small example and see.

(Refer Slide Time: 28:10)



So, let us have this node s here. This is a leaf, this is a leaf, and this is a leaf, these are leaf nodes. To each of the leaf nodes what we do to the edges directly attached to them we set a score of 1. For instance, this particular edge we work upward we have 1 plus this weight 1 that is equal to 2. For this edge, we have 1 plus this edge weight 1 plus this edge weight 2 that is equal to 4. And for this edge, we have 1 plus this edge weight equal to 2. In this way you basically compute the path counts that pass through every individual edge and this actually is proportional to the edge betweenness.

Now, you do it from all the sources considering every node in the network as sources and therefore after you have finished this step you find the total sum of all the path counts across a particular edge that actually gives it its contribution to the edge betweenness. So, in this way you can compute the edge betweenness in case it is a shortest path tree.

So, in the next part of the lecture we will see the more generalized case.