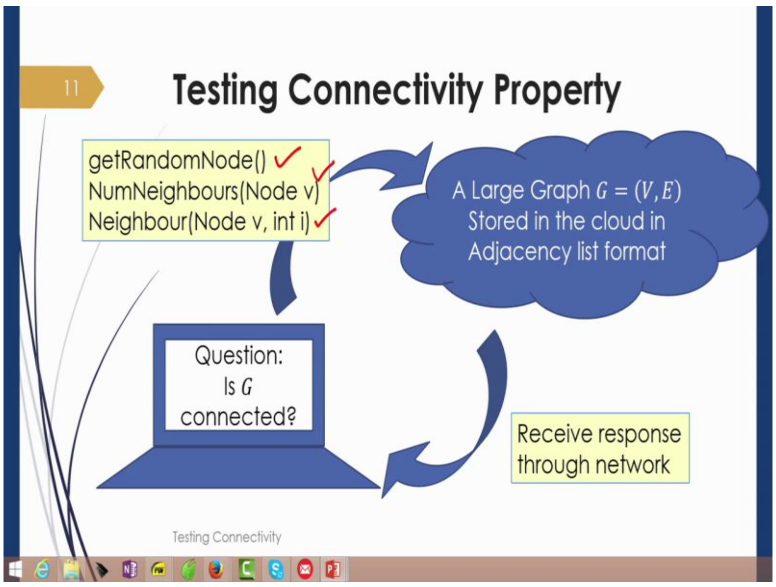
Algorithms for Big Data
Prof. John Ebenezer Augustine
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 38
Testing Connectivity

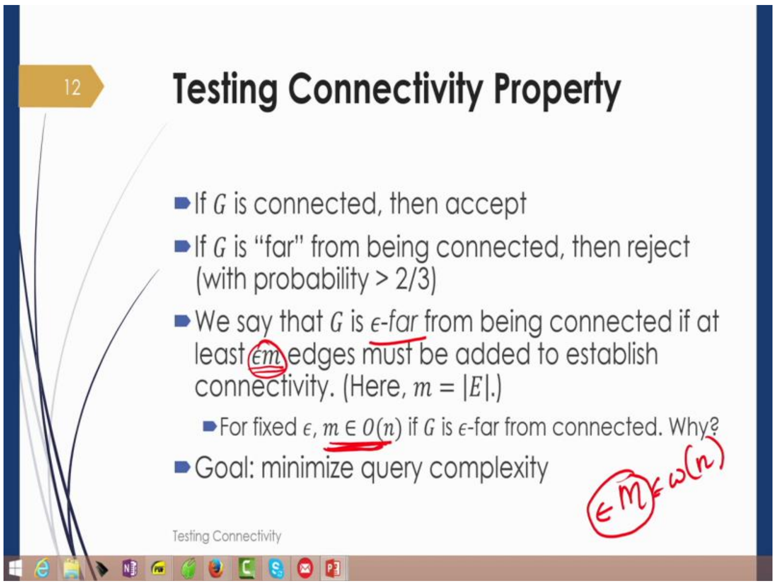
Hello everybody. Let us start the second segment of our lecture on property testing. We are interested in designing an algorithm and analyzing it, the test whether a given graph is connected and recall that the graph will be stored in some sort of a cloud.

(Refer Slide Time: 00:36)


11
Testing Connectivity Property
getRandomNode() ✓
NumNeighbours(Node v) ✓
Neighbour(Node v, int i) ✓
A Large Graph $G = (V, E)$ Stored in the cloud in Adjacency list format
Question: Is G connected?
Receive response through network
Testing Connectivity

So, we will not have full access to the graph rather we will have access to the graph by a queries and in particular, we have three types of queries that we can send to the cloud. The first query will request a random node in the graph, the second query request for the number of neighbors of a particular node v and the third query request for the say the fifth neighbor of a particular node v. These are the types of queries that we can make on these large graph stored in the cloud and the cloud will respond with the appropriate responses and we need to test whether this graph is connected with as few queries as possible.

(Refer Slide Time: 01:42)



12

Testing Connectivity Property

- If G is connected, then accept
- If G is "far" from being connected, then reject (with probability > 2/3)
- We say that G is ϵ-*far* from being connected if at least ϵm edges must be added to establish connectivity. (Here, $m = |E|$.)
  - For fixed ϵ, $m \in O(n)$ if G is ϵ-far from connected. Why?
- Goal: minimize query complexity

$\epsilon m \in \omega(n)$

Testing Connectivity

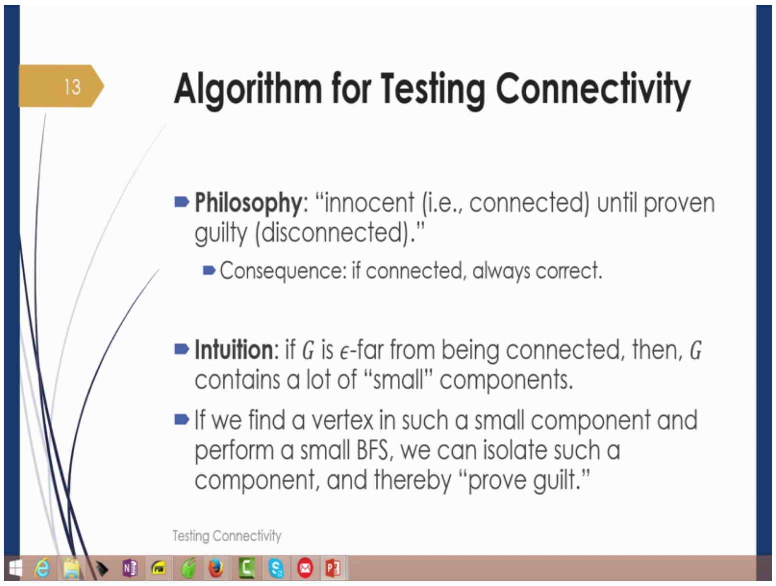
Of course, when the graph is connected we want to accept it. Normally, property testing only requires us to accept this scenario with probably the at least two thirds. In our context however, we are going to design an algorithm that is always going to accept a graph if it is connected. So, in order words this algorithm that we are going to designing is going to have a one sided error. If, however, the graph is far from being connected the algorithm we design we are going to design now will reject the graph with probability at least two thirds and what is it mean for a graph be far from being connected.

Well, this fix with the notion of being far from having a property that we already discussed in the first segment. The graph is epsilon far from being connected if at least epsilon m edges must be added to establish connectivity. So, the graph is highly fragmented if it is epsilon far from being connected and in order to defragment and if you where and get the graph to be connected, we will have to add edges and in particular we will have to add epsilon m edges in order to establish connectivity this is one little except of that we need to note.

Let us assume that epsilon is fixed and let us assume that the graph is epsilon far from being connected then we can conclude that the number of edges is only of the order of n. Now, why is this case? Suppose, m is little omega of n, in this case epsilon m is also going to be little omega of n in which case we immediately notice that this epsilon m number of edges is going to be sufficient for us to get the graph to be connected.

So, we simply cannot have the graph that is epsilon far from being connected having number of edges that is little omega of n. So, as a result we can say, if a soon that we are dealing with a parse graph with at most O of n edges. Our goal is, of course, to minimize the query complexity, basically the number of queries that we need in order to be able to answer whether graph is connected or not.

(Refer Slide Time: 05:02)



13

Algorithm for Testing Connectivity

- **Philosophy**: "innocent (i.e., connected) until proven guilty (disconnected)."
  - Consequence: if connected, always correct.
- **Intuition**: if G is ϵ-far from being connected, then, G contains a lot of "small" components.
- If we find a vertex in such a small component and perform a small BFS, we can isolate such a component, and thereby "prove guilt."

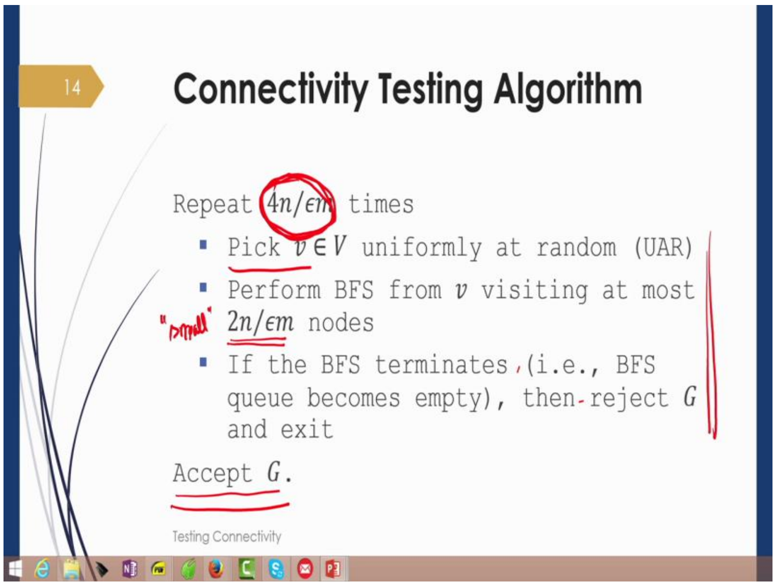
Testing Connectivity

A main philosophy behind our algorithm is the following. We are going to assume this sort of innocent until proven guilty, paradigm of thought meaning we are going to assume that the graph is connected and we are going to seek evidence that is not connected and until we find such evidence. We are going to say that the graph is connected, and when we do find some evidence not connected that is only then will be say that the graph is not connected as a consequence this the algorithm that we design will always be correct, when the graph is connected because we cannot find evidence which is not connected. So, and more over if we say that the graph is disconnected then we will only say that is disconnected.

When we have clear evidence it is actually disconnected. So, let us a little deeper look at intuition behind the algorithms, if the graph is epsilon far being connected then the graph will contain a lot of small components because a lot of edges need to be added to graph before we can get it connected which means that the graph is very, very fragmented and because the graph is fragmented, we want to say that not only other lots of components, there are lots of small components and so random sample Intel, the set of vertices is could very well with the good probability land us on such a small component.

Now, if we land on such a small component then we should be able to find evidence for this graph being disconnected because if we land in a small component a very small BFS should tell us, whether we are in a small component or a large component and so you are lucky enough to land in a small component and we perform the small BFS, we should be able to prove the guilt meaning, we have being able to prove that the graph is disconnected and this is the main intuition. So, let us look at the algorithm in greater detail.

(Refer Slide Time: 07:39)


14

Connectivity Testing Algorithm

```
Repeat 4n/εm times
    ▪ Pick v ∈ V uniformly at random (UAR)
    ▪ Perform BFS from v visiting at most 2n/εm nodes
    ▪ If the BFS terminates, (i.e., BFS queue becomes empty), then reject G and exit
Accept G.
```

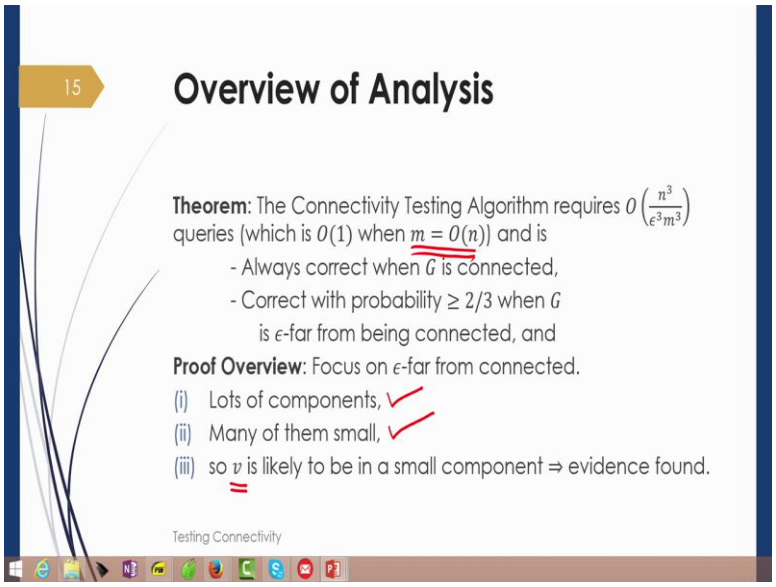
"small"

Testing Connectivity

The algorithm is iteration over a set of steps, for now, I were let us ignore the number of or at least let us not get distracted by the number of reputations. So, we are going to perform some number of reputations and each reputation is the following. We are going to pick a vertex v uniformly at random and this is done under the hope that we will are random choice will land in a small component and from that vertex v, we will perform BFS visiting at most some number of vertices and this is designed to be this is over notion of small.

We will formalize these things shortly, we perform a BFS and that visit at most these many numbers of nodes, if the BFS terminates that, if the BFS queue becomes empty; that means, we have actually reached a small component in which case we will reject g because we now know that the graph G is disconnected now. After these many of reputations, if we are not able to find any evidence at all that the graph G is disconnected then we will come out of the loop and we will accept G. So, this is the algorithm and as you can notice the algorithm itself is very, very simple. We now have to formally prove that are algorithm in deed accepts all connected graphs and rejects graphs that are far from being connected.

(Refer Slide Time: 09:46)



15

Overview of Analysis

Theorem: The Connectivity Testing Algorithm requires $O\left(\frac{n^3}{\epsilon^3 m^3}\right)$ queries (which is $O(1)$ when $\underline{m = O(n)}$) and is

- Always correct when G is connected,
- Correct with probability $\geq 2/3$ when G is ϵ-far from being connected, and

Proof Overview: Focus on ϵ-far from connected.

(i) Lots of components, ✓
(ii) Many of them small, ✓
(iii) so $\underline{v}$ is likely to be in a small component $\Rightarrow$ evidence found.

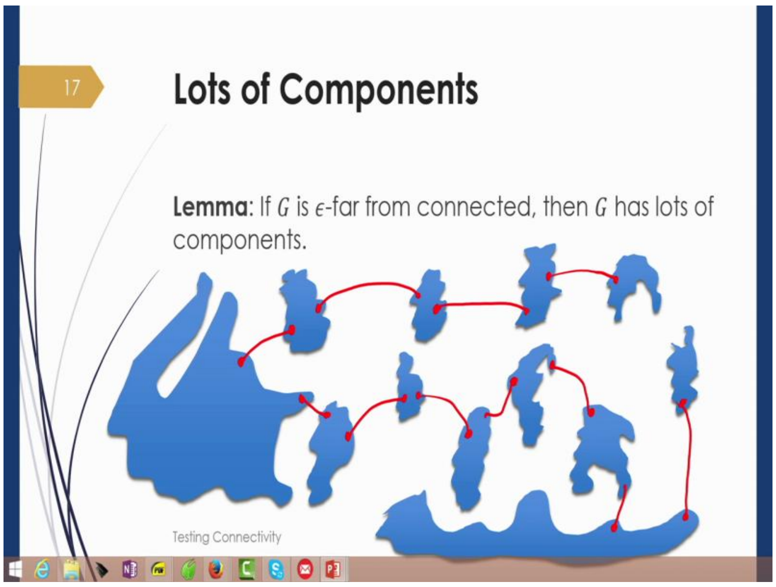
Testing Connectivity

So, in particular this is the theorem that we need to prove the connectivity testing algorithm requires O of n cubed over epsilon cubed n cubed number of queries and just a quick note about this, when m is O of n, the number queries is actually only O of 1 assuming epsilon is a fixed constant. Now, with that many queries our algorithm we need to show that our algorithm is always correct, when G is connected and it is correct with probability at least two thirds, when G, the practice epsilon far from being connected this is the theorem that we are going to prove and the overview of the proof is as follows.

We all is quite obvious this algorithm this always going be correct when the graph is connected simply because it is, if cannot find any evidence of being disconnected. So, our focus therefore, is going to be on graphs that are far epsilon, far from being connected. So, the sequence of steps in this proof is as follows, first we will prove that there are lots of components in the graph h, if its epsilon far from being connected and then we will go on to show that many of those components are actually small and will, of course, define small quite likely and then will go on to show that a randomly chosen v is likely to be in a small component.

Therefore, leading us to the evidence that we are seeking and we will is in the third step. We will work out the probability with which we will land in a small component and therefore, the probability with which our overall algorithm is going to be correct. So, this is the overview of our proof. Let us go on through the first step in this proof sequence.

(Refer Slide Time: 12:01)


17

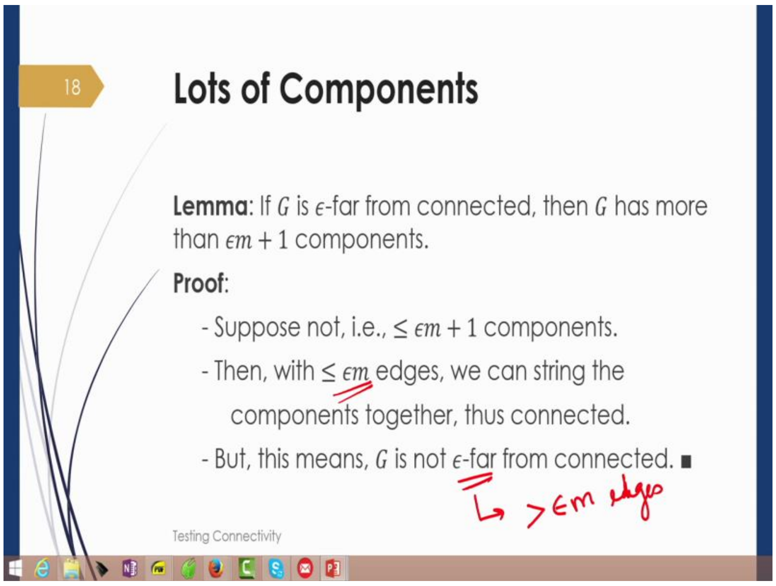
Lots of Components

Lemma: If G is ϵ-far from connected, then G has lots of components.

Testing Connectivity

Our first lemma is to show if the graph G is epsilon far from connected then G has lots of components and we have illustrated that by this figure shows all these components and here seen this the quick intuition. If it is epsilon far then we are required to add at least epsilon m edges before the graph can be connected and now, if very few components we can simply string them together as shown in this figure and we should be able to connect the entire graph. So, the proof relies on quantifying the number of such edges that we can add and therefore, the number of components that we should that the graph should have. So, let us be a little bit formal now.

(Refer Slide Time: 13:03)



18

Lots of Components

Lemma: If G is ϵ-far from connected, then G has more than $\epsilon m + 1$ components.

Proof:

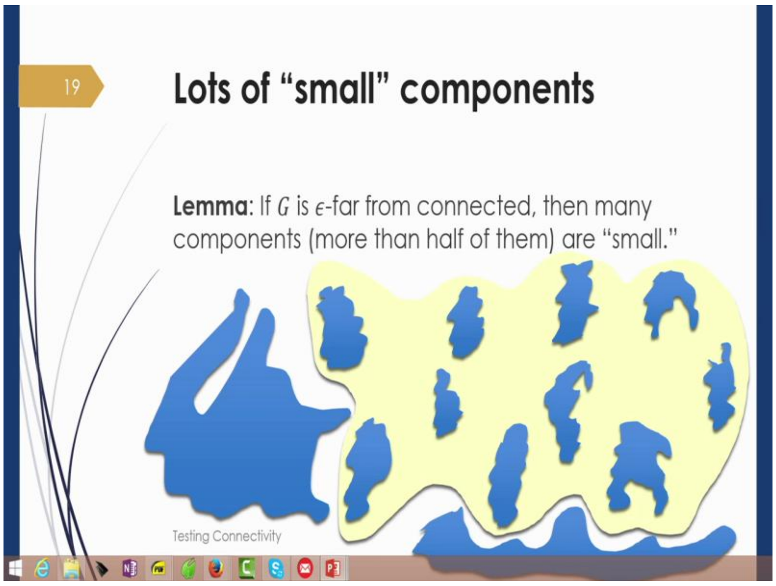
- Suppose not, i.e., $\leq \epsilon m + 1$ components.
- Then, with $\leq \epsilon m$ edges, we can string the components together, thus connected.
- But, this means, G is not ϵ-far from connected. ∎

$\hookrightarrow > \epsilon m$ edges

Testing Connectivity

If the graph G is epsilon far form connected then G has moved than epsilon m plus 1 connected components and the proof is as follows. Suppose, there is, if number of connected components is at most epsilon n plus 1 then at most epsilon m edges are to get string the components together and therefore, get them to be connected that this would mean that the graph G is not epsilon far connected because epsilon far would mean that we have to, we require more than epsilon m edges to for connectivity. So, this establishes the contradiction that we are seeking and therefore, we can conclude the lemma.

(Refer Slide Time: 14:01)

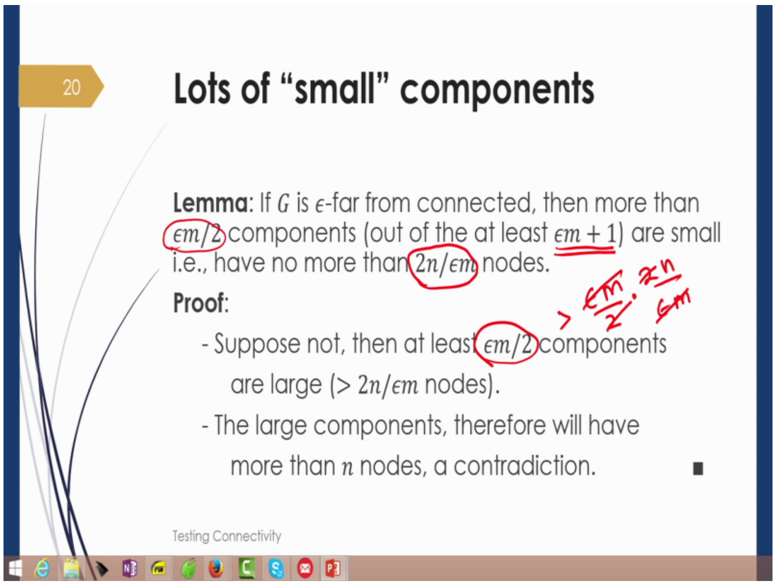


Lots of "small" components

Lemma: If G is ϵ-far from connected, then many components (more than half of them) are "small."

Now, that we know that graph that is epsilon far from being connected has lots of components. We want to show that not only do we have lots of components, but more over we want to show that we have lots of small components.

(Refer Slide Time: 14:24)



Lots of "small" components

Lemma: If G is ϵ-far from connected, then more than $\epsilon m/2$ components (out of the at least $\underline{\underline{\epsilon m + 1}}$) are small i.e., have no more than $2n/\epsilon m$ nodes.

$> \frac{\epsilon m}{2} \cdot \frac{2n}{\epsilon m}$

Proof:

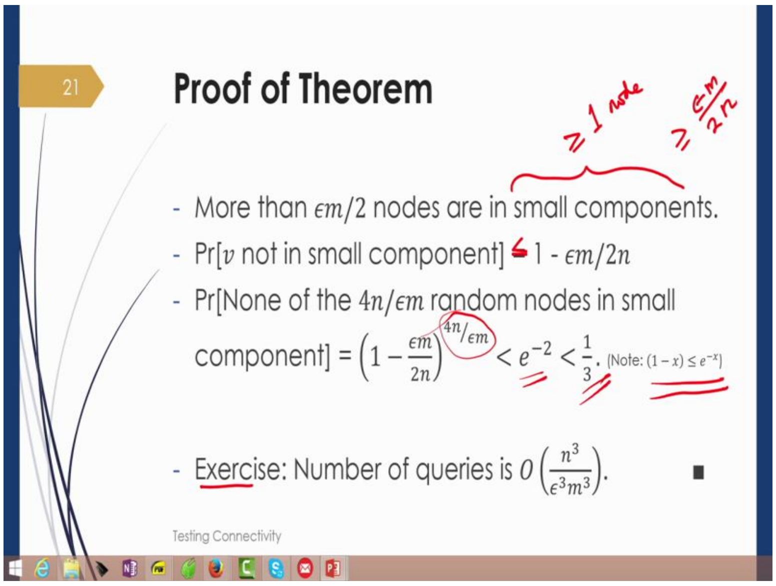
- Suppose not, then at least $\epsilon m/2$ components are large ($> 2n/\epsilon m$ nodes).
- The large components, therefore will have more than n nodes, a contradiction. ∎

Let us be able to more precise with what we want, If G is epsilon far from connected when more than epsilon m by 2 components. Now, recall that there are at least epsilon n plus 1 or rather than more than epsilon m plus 1 component. In this graph and are we want to show that more than epsilon m by 2 about half of them at least are small and what do we mean by small these are components that have no more than 2 n over epsilon m nodes in them. Let us go into the proof now.

Again, we use proof contradiction suppose not; that means, there are at least epsilon m by 2 components that are large. Now, if there are fewer than epsilon m by 2 small components then they must be epsilon at least epsilon m by two large components and there are components with strictly more than 2 n over epsilon m nodes.

So, how many nodes are there in these large components? All this work it's going to be well we have at least epsilon, I mean epsilon n by 2 component and each one of those components has more than 2 n over epsilon m about. So, that is 2 n over epsilon m. So, just that accounts for more than n nodes and that is simply a contradiction again. The graph itself is to be started-off with only n nodes. So, you we cannot have at least epsilon m by 2 large components, which mean that more than epsilon m by 2 components must be small.

(Refer Slide Time: 16:37)



Proof of Theorem

- More than $\epsilon m/2$ nodes are in $\overbrace{\text{small components}}^{\geq 1 \text{ node} \quad \geq \frac{\epsilon m}{2n}}$.
- Pr[v not in small component] $\leq$ 1 - $\epsilon m/2n$
- Pr[None of the $4n/\epsilon m$ random nodes in small component] = $\left(1-\frac{\epsilon m}{2n}\right)^{4n/\epsilon m} < e^{-2} < \frac{1}{3}$. (Note: $(1-x) \leq e^{-x}$)

- Exercise: Number of queries is $O\left(\frac{n^3}{\epsilon^3 m^3}\right)$. ■

Testing Connectivity

So, with these as in our background, we are now ready to prove the main theorem. So, just recall more than epsilon m by 2 nodes are in small components and each of these small components must have at least one node. Each one of these must have at least one node. So, what is the probability of a randomly chosen vertex v, not being in a small component? Well it must be at most 1 minus epsilon m by 2 n, why is that? Because what is the probability that we will vertex v will be in a small component well each of these small component has at least one vertex and there are epsilon m by 2such vertices at least.

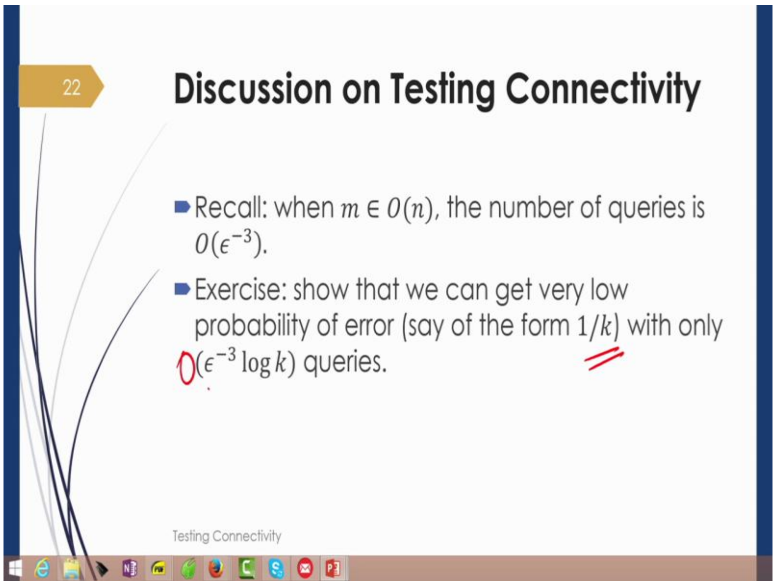
So, probability of landing in such a small component is going to be greater than or equal to epsilon m by 2 divided by the total number vertices that we have which is this is the probability of landing in a small component. So, the probability that v is not in a small component is at most one minus epsilon m by two n. So, this is one particular vertex v that was chosen uniformly at random.

If you recall the algorithm we repeated this 4 n over epsilon m number of times. So, what is a? This is the. So, let us look at the probability that none of the 4 n over epsilon m random nodes will land in small components. Now, this is really bad because we missed all the small components not once, not twice, but 4 n over epsilon n times. So, this is a really bad event, but what is the probability of that bad event well probability that it occurs once is 1 minus epsilon m by 2 n, but 4 n over epsilon m number of times means that the four n over epsilon m goes to the exponent.

So, if we apply this in equality minus x is at most e to the minus x this can be. This probability can be brought down to e or the minus 2 and of course, e being something like - 2.71 so on. So, etcetera we is easy to see e to the minus 2 is less than one third which is the probability that people not find evidence all over random vertex probes landed in in large components not in small components.

So, we missed this is the probability this is an upper bound in the probability that we need all the small components not once twice, but all four n over epsilon n times and. So, this is the probability of missing. So, the probability that we actually are algorithm will correctly land in a small component and find evidence that we are. In fact, in a disconnected graph is therefore, going to be at least two-thirds. So, that completes the part of the proof regarding the correctness of the algorithm. The number of queries is quite easy to show and. So, that is going to be an exercise for you.

(Refer Slide Time: 20:44)



22

Discussion on Testing Connectivity

- Recall: when $m \in O(n)$, the number of queries is $O(\epsilon^{-3})$.
- Exercise: show that we can get very low probability of error (say of the form $1/k$) with only $O(\epsilon^{-3} \log k)$ queries.

Testing Connectivity

Before we conclude let us have a quick discussion on what we have studied. So, for just to remind you, when m is O of n, the number queries is only O of epsilon to the minus 3 and if epsilon is also fixed, we are taking about a constant number of queries and with this constant number of queries, we are able to with reasonable probability detect whether the graph is connected or epsilon far from being connected and this works straight we are able to get probability of correctness of at least two thirds, but is actually not very difficult get that the probability of correctness down to a much smaller value.

For example, we can get the probability down to something also form 1 by k for any arbitrary k value, we just epsilon to the minus 3 with O of epsilon to the minus 3 times lock k queries. So, this will be a small exercise for you to show that this is in fact, true.

With that, let us conclude the second segment of our lecture on property testing our third segment is going to be on linearity testing.

Thank you very much