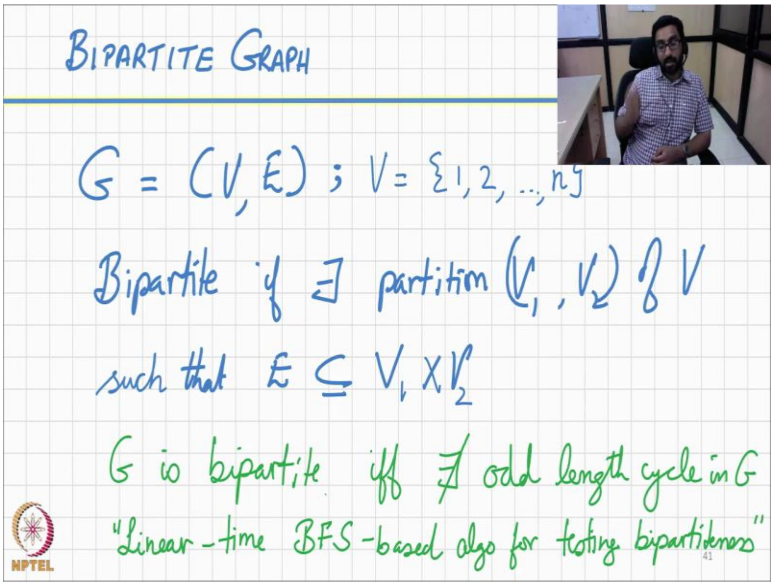
Algorithms for Big Data

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

Lecture – 41
Testing Bipartiteness

In this last segment, we are going to look at the problem of testing whether a given graph is bipartite or not. And of course, we are again in the dense craft model which means that the type of queries, we can ask are of the form, is there in edge between these 2 vertex U and v.

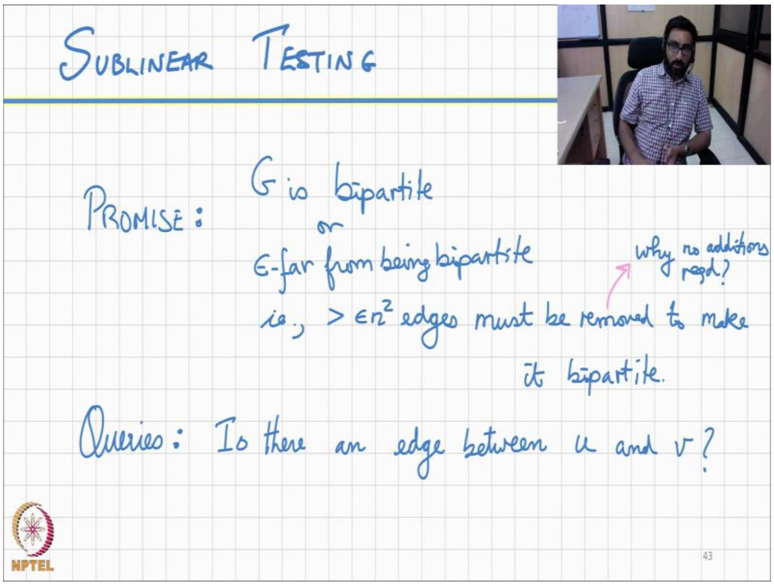
(Refer Slide Time: 00:39)



And you see those queries, we must ask the question whether the graph is bipartite type that is remind us, what we mean by a graph being a bipartite graph such a graph must have a partition of its vertices into sets v 1 and v 2, such that the edges are sub sets of v 1 cross v 2. So, the edges can only be of the form where they connect 1 vertex from the 1 and 1 vertex from v 2, but unlike the biclique does not have to be exactly v 1 cross v 2 and well known property of bipartite graph is that there it does not exist any cycle in the graph G.

In fact, these 2 statements are equivalent, the statement of G is a bipartite graph is equivalent to saying that the graph does not have an odd length cycle an exploiting this property. We can arrive at a linear time shall we say BFS based algorithm testing bipartiteness basically just test whether there is any cycle that is of odd length then it, but obviously, this is for our purpose because serve executing BFS would mean that we need linear time then any number of queries and that is not acceptable.

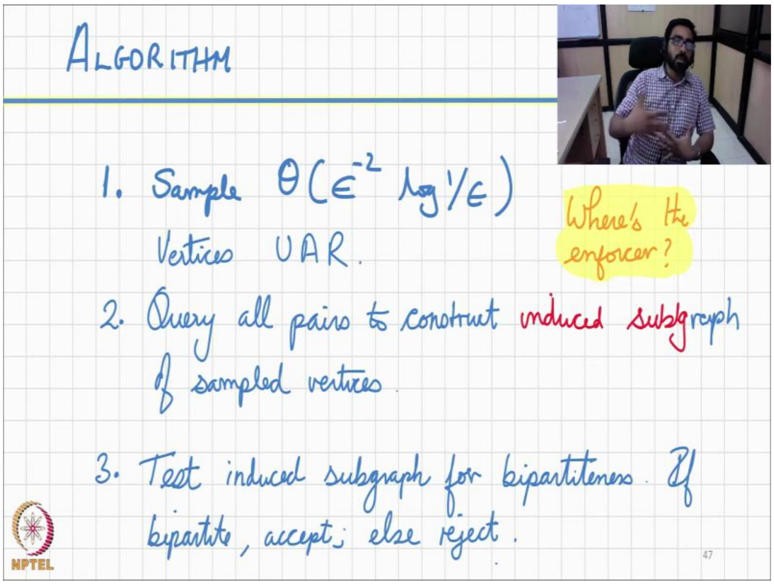
(Refer Slide Time: 02:23)



So, we have to be in the realm of sub linear testing and for this purpose, we will need a promise and the promise should be that either the graph G is bipartite or its epsilon far from being bipartite graph which means that more than epsilon n square edges must be removed in order to make the graph bipartite.

Notice that we only need to remove edges in order to make it bipartite because we really do not need to add any new edge because for bipartite graph. The edge set e only has to be a sub set of v 1 cross v 2 and the as we mentioned earlier we are only allowed queries of the form is there an edge between 2 vertices say vertex U and vertex v. So, let us further little look at the algorithm to test where the graph is bipartite in this sub linear setting.

(Refer Slide Time: 03:42)

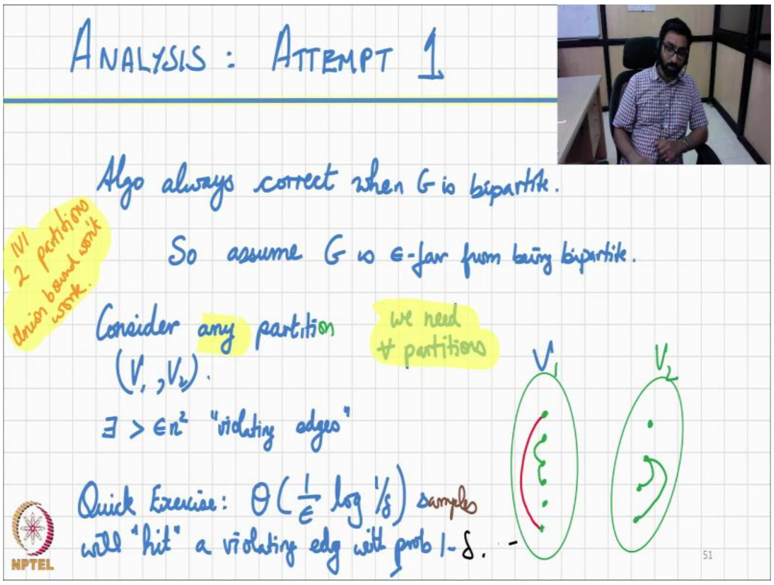


So, algorithm is very simple. We simply sample data of 1 over epsilon square times log 1 over epsilon number of vertices uniformly at random the constant hidden within the data annotation can be reverse engineered later. So, we are not going to write much about the constancies. In fact, this whole segment we are going to purposefully be a bit sloppy in order to focus on understanding what is going on and we will work out these constancies part of our exercises later.

So, we sample these many vertices and after we have sampled these vertices we will query all possible pairs vertices from the this sample and in and by doing that we will construct the induced sub graph of those sampled vertices and then we will just test whether that induced sub graph is a bipartite graph or not and this we can do using BFS member. We can we have a linear time algorithm using BFS, but at this notice that this is a linear time algorithm on a graph that is of size roughly square of epsilon to the minus 2 log 1 over epsilon and. So, that is still very much sub linear in terms of the larger graph that we are concerned about.

Now, we test the bipartiteness of this induced sub graph and if it is bipartite we accept otherwise we reject and of course, a verdict of accept or reject must hold in this in the larger input graph on n vertices. So, you may wonder where is the enforced part in the biclique case we had a clear enforced step, and a test step as a out in this case the enforced step and the test step are both blended and really it is the analysis that we separate them out. So, we will see that in a little while, but for a now let us on purpose make the missed step a wrong attempt to few will just a sort of get a sense for why we even need this enforce and test approach to analyzing this algorithm.

(Refer Slide Time: 06:23)



Notice, of course, of the algorithm always going to be correct when g is bipartite and. So, in like in many other such properties, property testing context we do not care, much about getting the right answers when the property holds we are more concerned about being able to reject when the property does not hold. So, for that purpose we are going to assume that the graph G is far epsilon far from being bipartite and now we have to show that the algorithm will reject such a graph with some good probability under that assumption consider a partition v 1 and v 2, we guarantee that the more than epsilon n square violating edges and these are edges there are going between vertices there are both in v 1 or both in v 2.

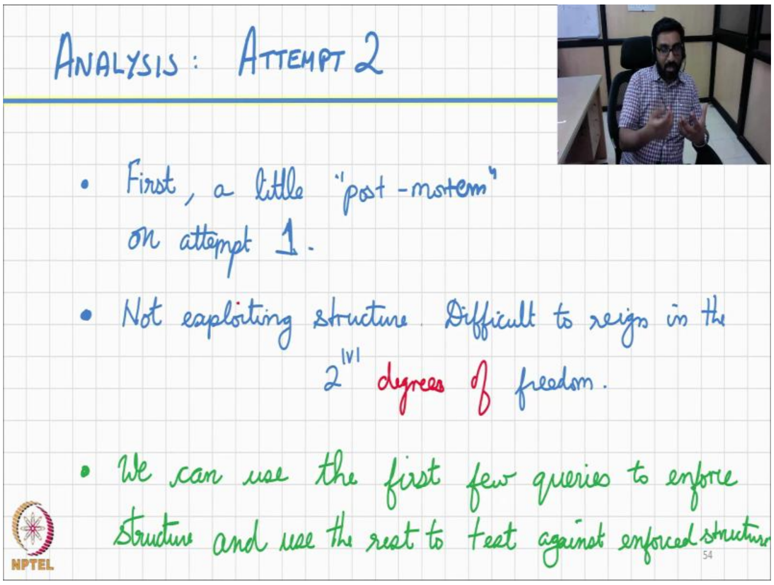
Here is a quick exercise for you, what you need to show that if you sample 1 over epsilon times log 1 our delta samples you will hit such a violating edge with probability at least 1 minus delta. So, for this you will have to look at the probability that each sample and here we are sampling edges in some sense each sample be a violating edge with probability epsilon and. So, within each sample will not be violating edge with probability 1 minus epsilon and we have 1 over epsilon times log 1 over delta such samples. What is the probability that all of them do not reveal a violating edge that will be we need to show that will be probe that will be of probability delta.

So, with probability 1 minus delta we will be able to find a violating edge. So, spend some time to work out those detail ones you convinced yourself that is the case lets step back and look at what we have achieved we have shown for a specific partition that we will be able to hit a violating edge with probability at least 1 minus delta that is greats for 1 partition.

But we really need to be concerned about every possible partition and here lies the problem because a number of partitions is a 2 raise to the cardinality of the vertex set and union bound is the only option at a disposal right now and that is not going to work. So, the probability of the failing is at most delta, but then if you more supply 2 raise to the cardinality of v times delta and still want that to be at small constant and delta must be extremely small and if you think about it delta has to be like 1 over 2 raise to the cardinality of v then log of 1 over delta is going to be log of 2 raise to the cardinality of v and that is going to be linear number of samples linear in the number of vertices that is just an unacceptable number of sample we want our algorithm to be able to upgrade under far few number of sample.

So, with that being the case we have to think about how we can we can have an enforce part and then test against what this being enforced in the enforced part.

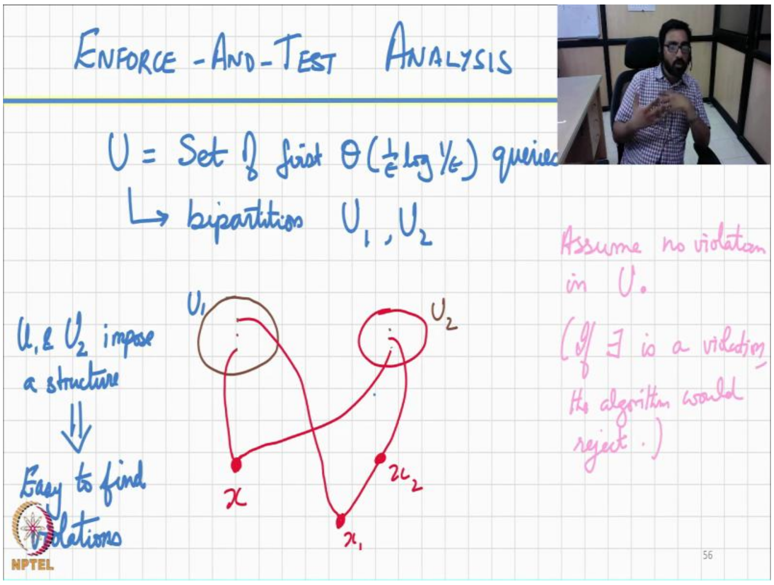
(Refer Slide Time: 11:16)



So, this leads us to the second correct attempt. So, just a little quick post mortem on attempt 1 the failed attempt we were not able to reign in the 2 power cardinality of v number of degrees of freedom of that we had in partitioning the vertex set. So, we have the there is lot of structure which we need to able to exploit and. So, we are going to do that following remember the enforced part is implicit what we are going to do is just the first few queries as enforce our queries, and thought those first few queries will create a structure, and then the rest we used to test against that structure and this is the quite nicely analogize to the balls and bins way of analyzing that we had before.

The algorithm i mean, the process was same for all 2 squared of n number of balls, but for the analysis purposes we set the first 2 first squared of n balls separately created structure, and then looked at what happened through the next squared of n balls. And this, what we are doing here is very similar the first few queries were going to enforce a structure, and then on 1 even though the algorithm does not change we in the analysis we look at the structure that is enforced by the first few queries and then use a subsequent queries to test against that enforced structure.

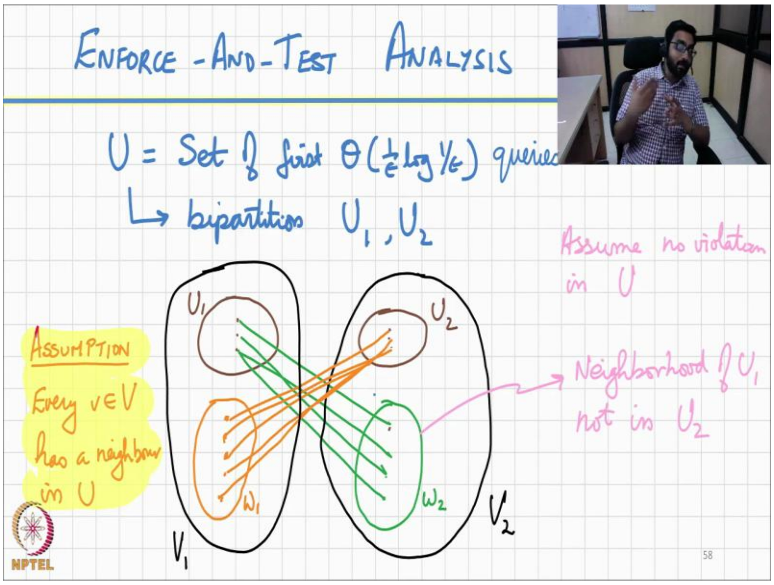
(Refer Slide Time: 13:12)



So, let us look at it bit more carefully. So, we are going to call the set of first data of 1 over epsilon log 1 over epsilon queries as the set U these this is the part that is going to enforce a structure.

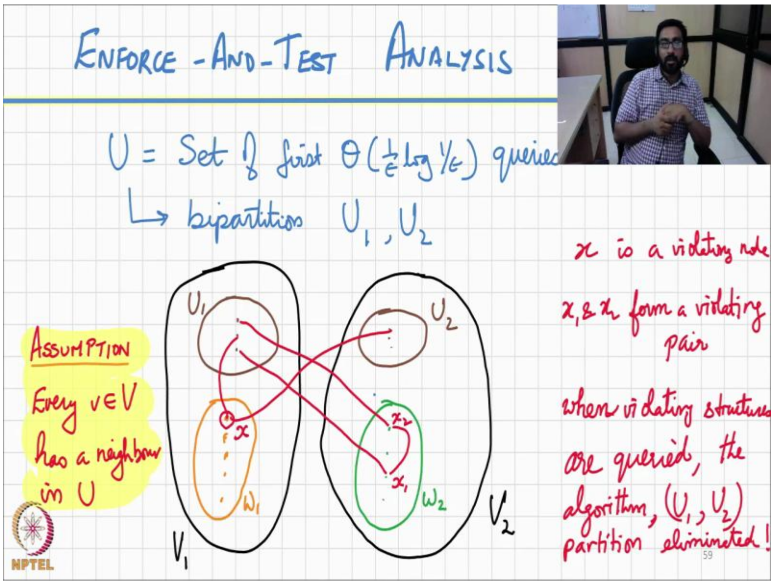
Let us pick by partition U 1 comma U 2 of the set U in. So, the moment we have U 1 and a U 2 we are going to assume that U 1 and U 2 a had chosen such that there are no edges between vertices in U 1 or no edges between vertices in U 2, if we cannot find such a partition then were already done the graph is no longer bipartite graph. So, we found a violation, but let us assume that we could not find violation and. So, we have this U 1 and U 2 how could we subsequently find and this creates a structure some vertices are now in the left in U 1 and some vertices in the right in U 2 there are lot more vertices in the graph, but how do we find an violation against the structure there are 2 ways in which this can happen 1 is you can find in the vertex x such that x has a neighbor in U 1 and the neighbor in U 2 if that is the case then U 1 and U 2 cannot be a valid partition. So, that is 1 way to find a violation there now a second way to find a violation either. So, you find an edge x 1 comma x 2, but x 1 happens to be neighbor of some vertex in U 1 and x 2 happens to be a neighbor of some vertex in U 2 and this again would be a violation.

(Refer Slide Time: 15:27)



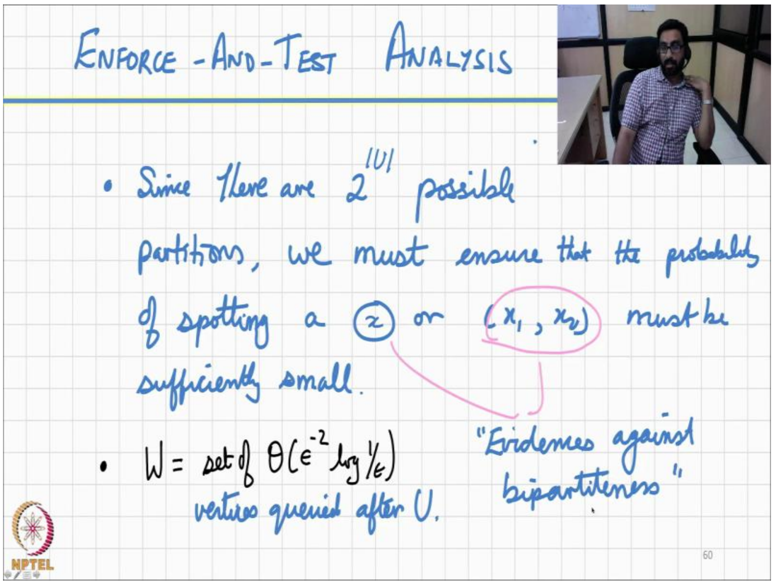
So, our goal with rest of the queries is to be able to find 1 such violation and remember U 1 and U 2 create a structure. So, what we are going to do is look at the settle set U 1 and in particular spot all the neighbors of set U 1, let us call that set w 2 along with U 2 we bunch that together and call it is set v 2 this is the 1 part of the partition the entire graph and all other vertices we place in v 1. So, U 1 and w 1 shown here in orange together from v 1 and you are going make a simplifying assumption that is certainly not true in general, but however, it to keep analysis simple and it is actually a fairly easy assumption to remove later. So, we are going to remove that assumption as part of an exercise, but for now we are going to make this assumption that every vertex other than the ones in U 1 and U 2 - have a neighbor in either in U 1 or in U 2.

(Refer Slide Time: 16:59)



So, the rest of the analysis is going to be based on that assumption and lets recall that vertex x would be a violating node basically in evidence that this partition will not work as the bi partition if it is a neighbor to a vertex in U 1 and also a neighbor to vertex in U 2 over another way to find the violation would be a pair of vertices x 1 and x 2 such that 1 of them as such that both of them neighbors of U 1. So, they actually have to be in set w 2, but they in the pairing an edge between them n a violation and when either 1 of these violating structures are queried the algorithm essentially eliminates this partition U 1 comma U 2.

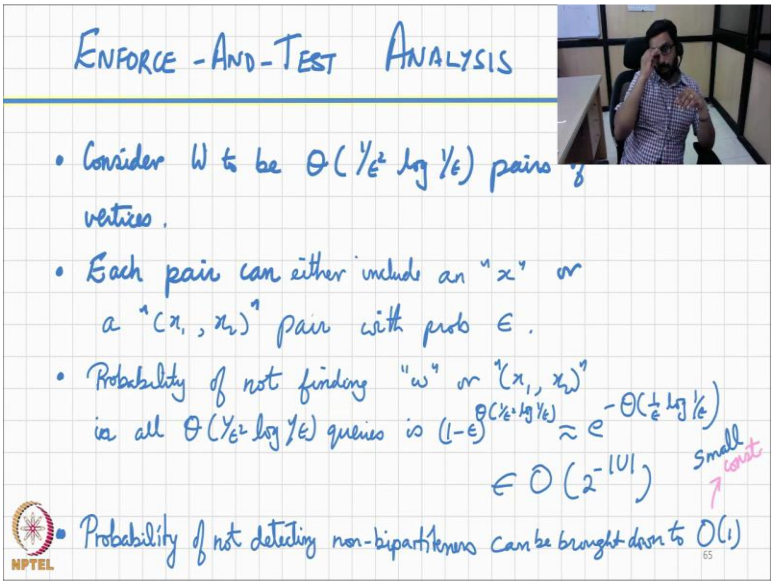
(Refer Slide Time: 17:55)



But keep in mind that there are total of 2 raise to the cardinality of the U number of possible partitions of the set u. Now, we must ensure that not only is this set U 1 comma U 2 is eliminated we must ensure that our queries are strong enough to eliminate every partition possible partition of the set U and keep in mind that this whole thing is operating under the assumption that the graph is. In fact, far from being expand from being a bipartite graph and. So, we need to the second part the test part should be strong enough to reject all these 2 power U possible partition and the goal is for each of these 2 power U possible partition should be able to find a violating structure either x or the edge x 1 comma x 2.

Now, for this purpose we set the test part that is the set w which is the next set of data of epsilon to the minus 2 log 1 over epsilon number of vertices queried after the U and we treat them as pairs because we want to either an x or x 1 comma x 2 for simplicity we are going to treat this set w as pairs. So, each pair could either include an x as evidence against bipartiteness or include a pair x 1 comma x 2 which will again be evidence against the bipartiteness.

(Refer Slide Time: 19:45)

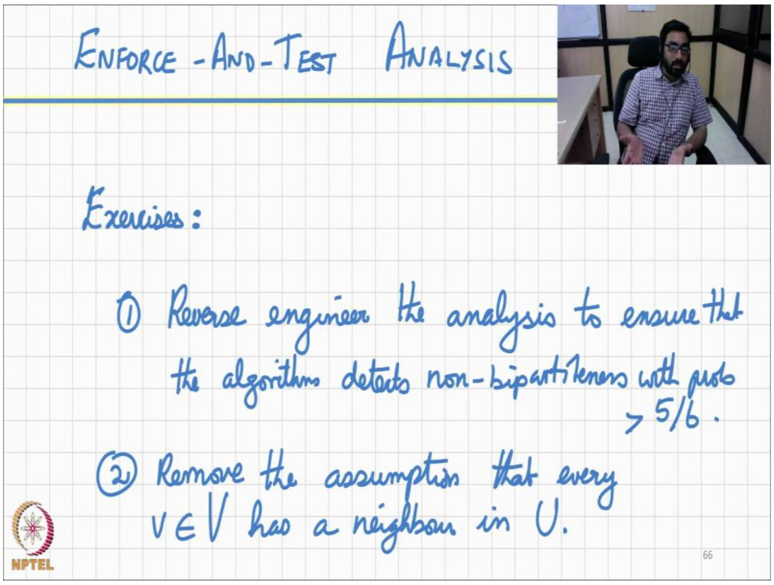


So, now what is the probability of a not finding an x. So, this must be an x by the way what is the probability of not finding an x or an x 1 edges from x 1 comma x 2 these are witnesses against this particular partition U 1 comma U 2 in all of the 1 over epsilon square log 1 over epsilon query this is the set of all queries in w the second set of the test set, and that is going to be 1 minus epsilon raise to the data of 1 over epsilon square log 1 over epsilon which we can work it out will be o of 2 raise to the minus U and of course, we with the appropriate constancy embedded in to this these data annotation we can bring this probability down to a small constant and this will be this small constant will be the probability of not detecting bipartiteness.

So, this gives us the overall idea. So, in the first part we basically remember we were focusing on graphs that are epsilon far from being bipartite and the first part is the first set of U queries and that we have basically enforce a structure and there are now we focus on 2 power cardinality of U number of partition U pick partition U ensure that with some very low probability o of 2 raise to the minus cardinality of U probability U ensure that the bipartiteness, we missed the bipartiteness, we miss finding evidence for the bipartiteness.

With such a small probability which means that we will missed we will not detect the nonbipartiteness with again very slow probability even when we are consider all the 2 power cardinality of U number of partitions and. So, over all the probability that we will not detect will be brought down to a constant and that is exact a small constant that is our final goal which we have achieved. But there are few details need to be worked out.

(Refer Slide Time: 22:33)



So, there are 2-2 exercises that we need to be complete first 1 we need to reverse the engineer analysis to ensure the algorithm detects non-bipartiteness with some reasonably good probability, say 5 over 6 and then in this step is fairly easy then the interesting challenge would be to remove the assumption that we made. Remember, we made the assumption that all vertices that are not in U are neighbors of U; we need to ensure that that assumption is removed and that will again will be an exercise that we will work out.

So, this brings us to the end of our lecture on this enforce and test technique for property testing that especially is useful in the context of dense craft property testing problem.