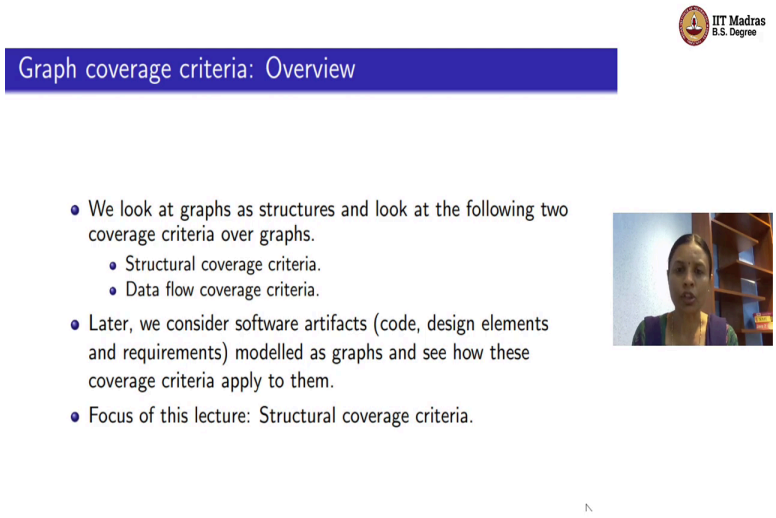
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information, Bangalore
Week: 2 Lecture: 3
Structural Graph Coverage Criteria

(Refer Slide Time: 00:15)

Hello, everyone. Welcome back to Week 2, we saw the basics of graphs then we saw some basic algorithms on graphs. Now, it is time to see how we are going to use graphs in testing. So, in this lecture, and in the next one, I am not going to tell you about how graphs are derived from software artifacts.

But instead, we continue to look at graphs as they are, and see some coverage criteria that is, purely dependent on the structure of a graph. So, we call them a structural coverage criteria. In next week, we will see actual code segments, how to map them as graphs, and how to apply these structural coverage criteria on those code segments. So, this would still continue to be purely graph theory oriented.

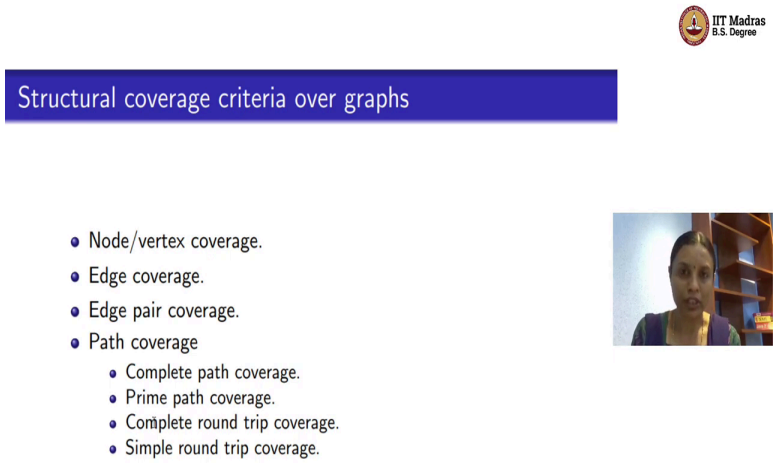
(Refer Slide Time: 01:08)



So, as I told you, we would like to look at graphs that model software artifacts, they could model control flow and methods, called graphs in code, they could model design elements requirements. And then we would apply test requirements on those software artifacts modeled as graphs generate test cases to meet those test requirements.

Test requirements can broadly be of two kinds when we consider them over graphs. One is purely based on the structure of the graphs, we call that a structural coverage criteria. The next would be to augment the nodes and the edges of a graph with information about variables and then consider the flow of the value of the variables, we call them as data flow coverage criteria. To start with in this lecture, and in the next one, we will look at structural coverage criteria.

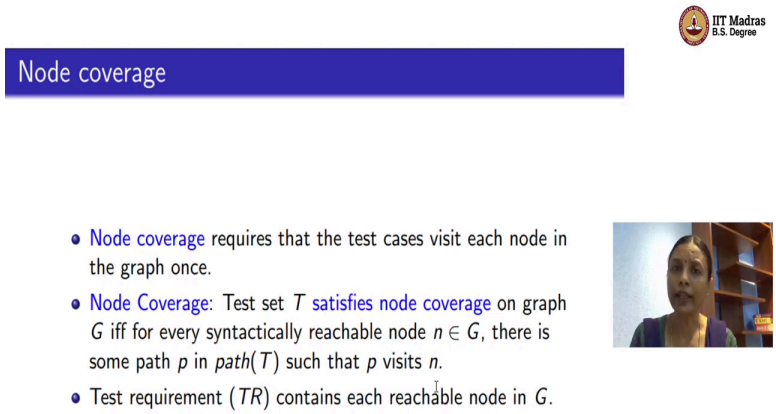
(Refer Slide Time: 02:06)



So, when I look at a graph, we all know what a graph is, what are all the various components, structural entities that you find in a graph, it has nodes, or vertices, we use these two terms synonymously, it has edges, then it could have set of edges, set of nodes, set of vertices, then graphs have paths of various kinds.

So, that is about what we are also going to use when it comes to their use and testing. So, this slide lists these structural coverage criteria that you will see over graphs, so we can cover for nodes and vertices, we can cover for edges we can cover pairs of edges. And when it comes to paths, we could generously say execute all the paths that is complete path coverage we will define a notion called prime paths. Which are very useful for testing loops in this lecture. And then (to) two entities that relate to round trips and detours. I will tell them about I tell them to you very soon.

(Refer Slide Time: 03:11)

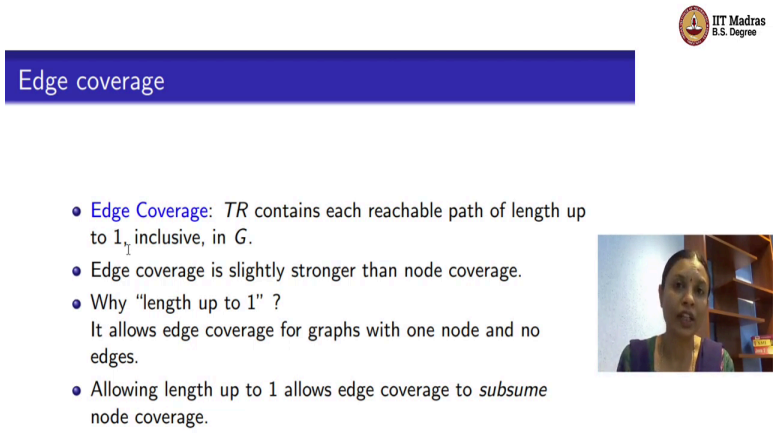


So, no nodes in graphs for now, do not worry about where the graph comes from, what do the nodes in the graph correspond to we will see that next week, but assume that you have a graph modeling a software artifact. So, that graph has nodes and it has edges it has paths. So, my simplest test requirement could be node coverage.

What is it say as a test requirement, right test cases that will visit every node once in the graph at least once in the graph. So, cover every node that is right a set of test cases, how to test cases for graphs look like they are path that begin at an initial vertex and end at a final vertex. So, what is being said a, node coverage is that right a set of test cases for a test requirement, which is node coverage, node coverage says the test cases should visit every node once.

So, that is what is given in the definition here. So, we say a test set or a set of test path T satisfies node coverage as a test requirement on a graph if and only, read iff as if and only if, for every node that is reachable in the graph syntactically reachable means there is a path from the initial vertex to that node, there is some path in this test set T such that, that path visits that node. So, just let me summarize in simple terms test requirement is node coverage. That is right a set of test cases that will visit every node at least once. How do the test cases look like? They are basically test paths that will visit every node at least once that is node coverage for you.

(Refer Slide Time: 04:57)



The next is edge coverage. What is edge coverage say the test requirements for edge coverage as the name indicates, visit every edge at least once instead of writing it as visit every edge at least once if you notice the first bullet in this slide, we write visit each reachable path of length up to 1. So, what is the length of an edge? Length of an edge is just the edge right two vertices (connecting) connected by an edge.

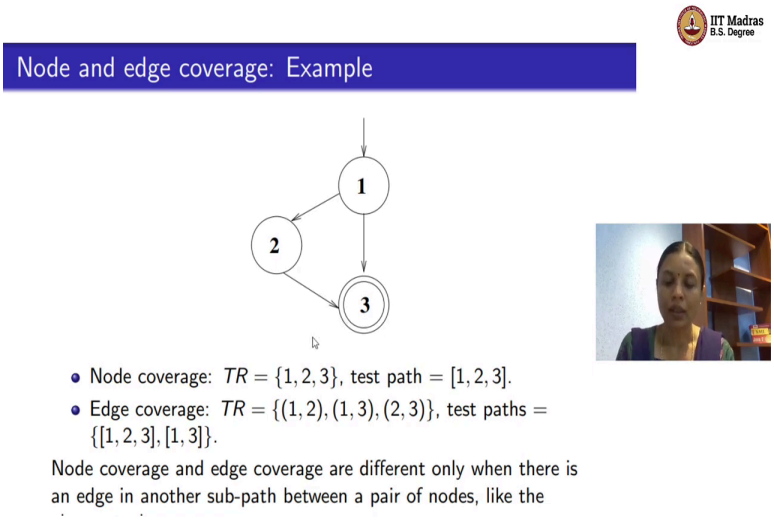
So, edge can be thought of as a path of length 1, instead of saying edge coverage, which is visit each edge or visit each path of length 1, we are saying visit each path of length up to 1. So, why do we need up to 1 that is because for reasons that matter to us in testing, we would like edge coverage to always, subsume node coverage. So, suppose I have graph derived from a software artifact.

And I say my test requirement is to achieve edge coverage in the graph. By default, for convenience, I would also like to assume that it also achieved node coverage that is whenever I satisfy a set of tests path are written to achieve edge coverage, the same set of test paths should also achieve no coverage. This is purely for convenience.

And because I need this to be true, what I say is edge coverage as a test requirement contains test cases that have to execute each path of length up to 1 path of length 0, which corresponds to nodes. So, achieves node coverage paths of length 1 which corresponds to edges, so it achieves

edge coverage. The simple reason is I want edge coverage to subsume node coverage that is as a set of test cases written to visit every edge should also correspond to the set of test cases written to visit every node. So, that is why I have edge coverage, so, the next.

(Refer Slide Time: 06:59)

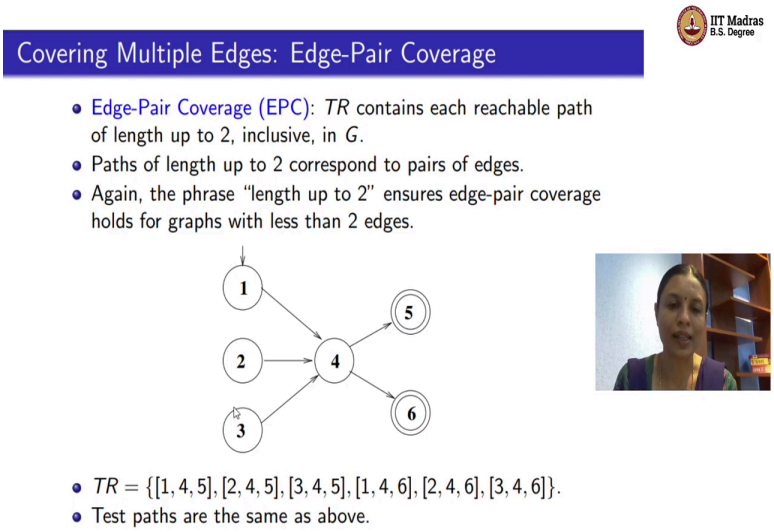


So, before we move on to the next one, this slide has an example. So, on top here is this small graph, very small graph. So, it has three vertices 1, 2, and 3, as you know by now, how to read it. Vertex 1 is the initial vertex; vertex 3 is the final vertex. So, how many nodes does this graph have 1 2 3, 3 nodes how many edges does this graph have edge from 1 to 2, edge from 1 to 3 and edge from 2 to 3, so three edges. 1 2 3

So, node coverage test requirement is just basically the set of nodes 1 2, and 3 test path can be what? Actually, there is a mistake here, test path should be (1, 2), and then (1,3), it is not just this because there is no path that goes is (1, 2, 3), (1, 2, 3) is a path, 1 to 2, 2 to 3 is a path so, correct. So, that covers node coverage so, this is correct.

Edge coverage is what is the test requirement is basically the set of all edges, as we just said there are three edges, 1 to 2, 1 to 3 and 2 to 3. So, the test path that goes as 1 to 2 to 3 will cover the edges 1 2, and 2 3, but then the edge 1 2 will be left out. So, for that I add this test path 1 3. So, for edge coverage, the TR or the test requirement is all the set of edges, the test paths that achieved this test requirement are these two paths path 1 2 3, and path 1 2, is this clear.

(Refer Slide Time: 08:32)



So, the next coverage criteria, is edge pair coverage. So, as the name indicates cover pairs of edges. So, you might be wondering, why would I need to cover pairs of edges, many times you will realize that when I look at control flow graph of methods and functions, the graph will have such a structure as you see in the slide.

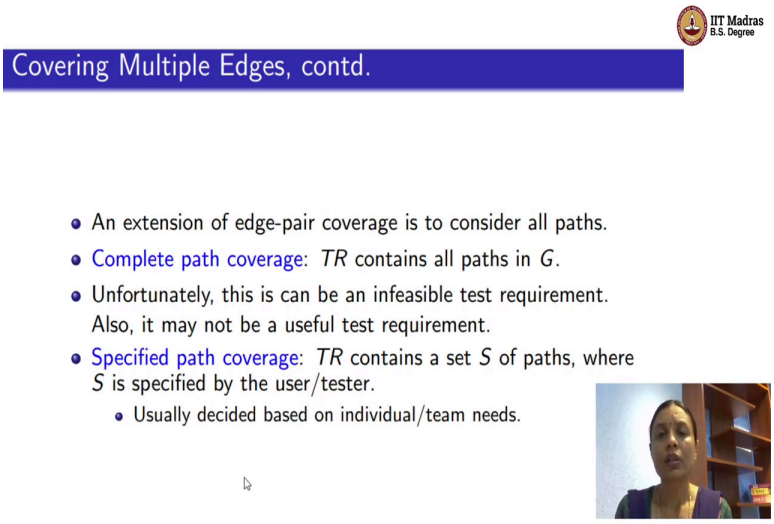
So, there will be several vertices edges that converge from several different vertices into one vertex, and then there will be edges going out of it. So, this kind of structure in the control flow graph makes it very naturally amenable to write a test requirement which says cover pairs of edges because I could do paths of length 2.

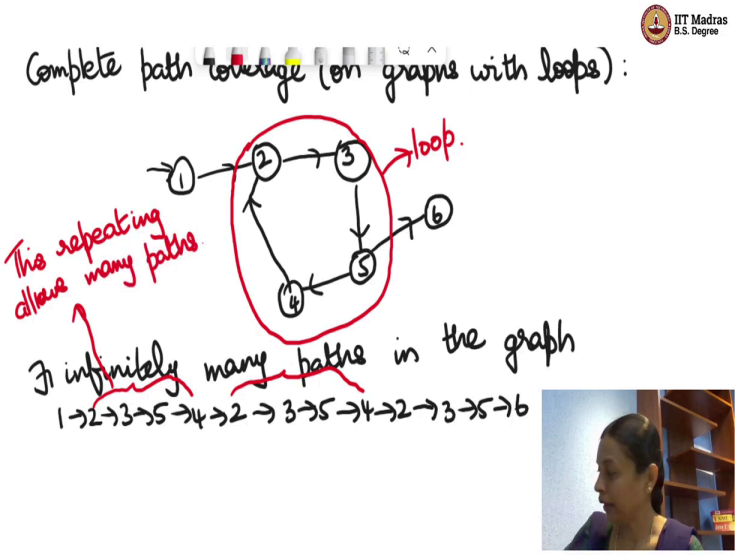
So, I could do (1, 4, 5), (1, 4, 6), (2, 4, 5), (2, 4, 6), (3, 4, 5) and (3, 4, 6) as again, like just done for node and edge coverage. I would like edge pair coverage to subsume both edge coverage and node coverage. So, I extend paths of length 2 from the test requirement to write paths of length up to 2. That is mainly because I would like edge pair coverage to subsume edge coverage and node coverage.

So, this example is what we just walked through for this example graph where 1 is the initial node 5 is the final node, there are 5 and 6 are final nodes there are totally 6 nodes, and 5 edges, the test requirements for edge pair coverage here, actually should enumerate all the edges and all the vertices. But then subsumption is there. So, I just write paths of length 2 only. So, these are

all possible paths of length 2, I just read them out to you. So, the test paths in this example happened to be the same as the test requirements. So, I hope this is clear.

(Refer Slide Time: 10:35)





Now, you can go edge pair means paths of length 3 paths of length 4, and so on. But then when do we stop? If so I could say cover all the paths, consider all the paths in a graph. But then take a case where the graph has a loop. Loop means a strongly connected component. So, suppose a graph has a loop, it means that the underlying code there is a loop in the graph.

So, if I consider all possible paths, structurally, what will it mean? It will mean that I go through the loop once maybe I go through the loop twice, I go through the loop 3 times, 4 times, and so on, it will become infeasible to cover all possible paths. So, let me just give you an example. Maybe we will keep here. So, complete path coverage on graphs with loops.

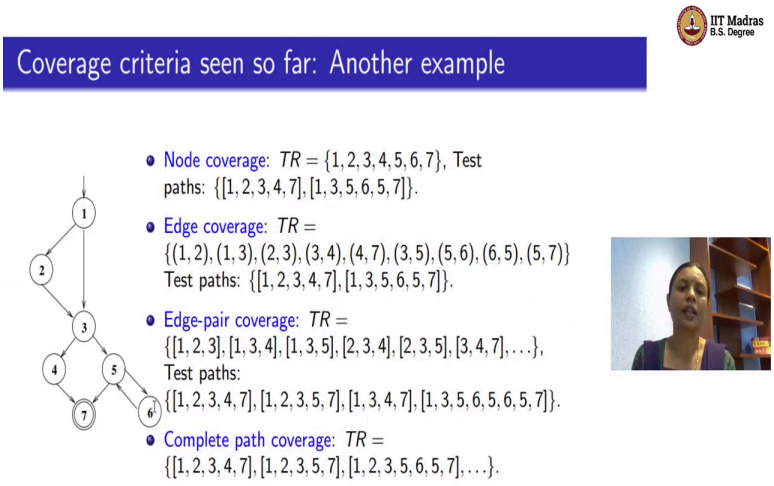
So, we take an example. So, let us say this is the initial node, you have this, this is the final node. Sorry about the sloppy writing. So, let name them as 1 2 3 4 5 6. So, where is the loop here? This is the loop 2 3 4 and 5, this is a loop. So, this loop means what? There exist infinitely many paths. So, I could do 1, 22, 23, 25, 24, 22, again, 23, 25, 24, 22, again say let us say 3 to 5 to 6.

So, here if you see, this loop is visited once here 2 3 4 5, 2 3 4 5. So, like this second repeat, so this behavior can repeat. If these repeats, then it allows many possible paths. Is it clear? So, this is why we say complete path coverage for graphs with loops is basically an infeasible criterion I do not know there are infinitely many paths.

How do I write the test requirement? How do I write test cases, it is not useful at all. So, what people do is they say, as a tester, you specify which are all the path of interest that matter to you. So, suppose there was a loop like this, you might want to specify and say that to cover this loop, maybe once, cover this loop, maybe 5 times for some reason.

So, people specify path or cover a path in this graph, it skips the loop, like for example, go from 1 to 3 to 5 to 6, do not go through the loop. So, people can do what we call specified path coverage, where the test requirement is specify a set of paths, and then achieve test cases to write those paths. And this is completely dictated by the user tester based on their individual needs, team needs and so on. So, this is the thing about complete path coverage and specified path coverage.

(Refer Slide Time: 14:48)



So, here I have given another example just to revisit the criteria that we saw so far. So, on the left you find a graph that has seven vertices, 1 is the initial vertex 7 is the final vertex. And these are all the paths. So, there is roughly a decision statement happening at 1, which lets you take the branch to 2 or 2, 3, there is a loop happening here 5 6, 5 6, 5 6 and then everything edge at 7.

So, what is node coverage, test requirements? Basically, the set of all nodes 1 2 3, and so on up to 7, what would be a test path, pick any set of test paths that will visit each node at least once. So, for example, remember, all test paths in software testing have to start from an initial vertex and always end at a final vertex.

Remember RIPR criteria that we saw, I should be able to give test cases as inputs. And I should let the output be visible always. So, test paths will always begin at an initial vertex always end at a final vertex. So, that will be 1 2 3 4 7. That is one test path that visits the nodes 1 2 3 4 and 7, which are the nodes that are left out, 5 and 6 are left out.

So, pick another test path for node coverage, so you could do 1 3 5 6 5 7. So, between these two test paths, your node coverage is completely met. Instead, you could also do 1 2 3 5 6 5 7. That is also fine there is no requirement about the length of the test path being long or short. Any test path, which begins at an initial node ends at a final node, and visits all the nodes will be a good

set of test paths that will meet the test requirements of node coverage. What is edge coverage? Test requirements, list all the edges.

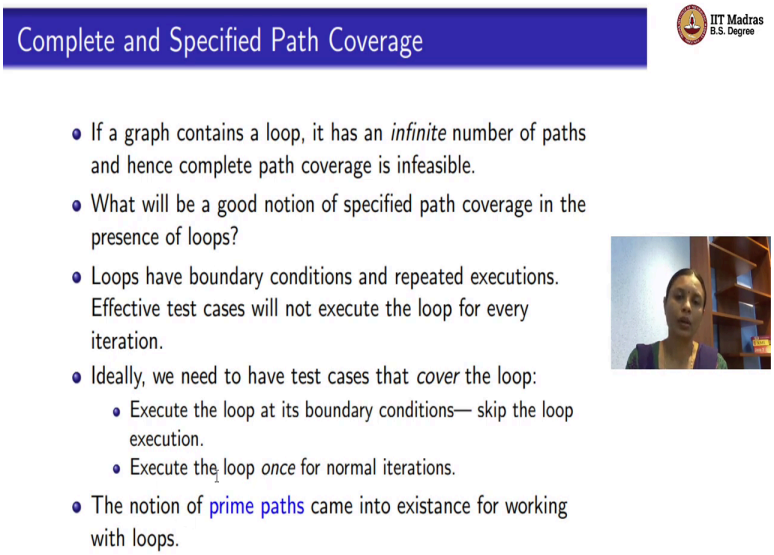
So, if there is an edge from 1 to 2, edge from 1 to 3, edge from 2 to 3, and so on, that is what we have listed here. In this place, edge coverage this full set, what will be a test path, any set of test paths that will visit all these nodes. So, what I have listed here, that need not be the only set anything else that visits all the edges is fine.

So, I have listed here 1 2 3 4 7 same as node coverage 1 3 5 6 5 7, edge pair coverage, list all paths of length 2 it is subsumes paths of length 1 which are edges and paths of length 0, which are nodes, so I am not repeating them. Here, this list with this dot-dot-dot here that you see on the right means there are some more to write, there was no space in the slide that will come too cluttered, so I left it.

So basically, the test requirements for edge pair coverage lists all pairs of edges of length 2, and it includes edge coverage, which is paths of length 1 and node coverage, which is paths of length 0. So, the test paths for edge coverage, I have enumerated them here, you can check it out, write in your notebook and check it out between these four paths.

Every pair of edges consecutive pair of edges will be covered. Complete path coverage please note this graph has a loop whereas the loop it is from 5 to 6 here on this graph. So, complete path coverage the test requirement is going to be an infinite set so, here also have put dot, dot, dot, but this dot, dot, dot represents an infinite set. So, there is going to be infinitely many paths as a test requirement because there are infinitely many paths to achieve complete path coverage is infeasible as a test requirement. So, we have not written test paths to satisfy complete path coverage.

(Refer Slide Time: 18:30)



So, this is what I explained, I will just keep the slide on and quickly revise it. So, we said that if a graph contains a loop, it has infinite number of paths. So, complete path coverage will be infeasible. So, people usually resort to specified path coverage. But then, how good is the tester to specify which is the right set of paths do we need can we have a structural criteria again, that will cater to graphs with loops.

Because I am going to have procedures and functions with loops in my code so, the control flow graphs corresponding to them are going to have loops in their structure, is there a nice way out that I can typically use to test for loops, apart from specified path coverage. So, that is what we are going to see.

So, typically, when you have a loop, and when you want to do white box testing for a loop, as a tester, you will always be worried about can I give a few test cases that will execute the loop once for normal operations. Let us say a loop is meant to execute 10 times I will give a set of test cases that will go enter the loop and maybe execute it maybe 5 times or 6 times.

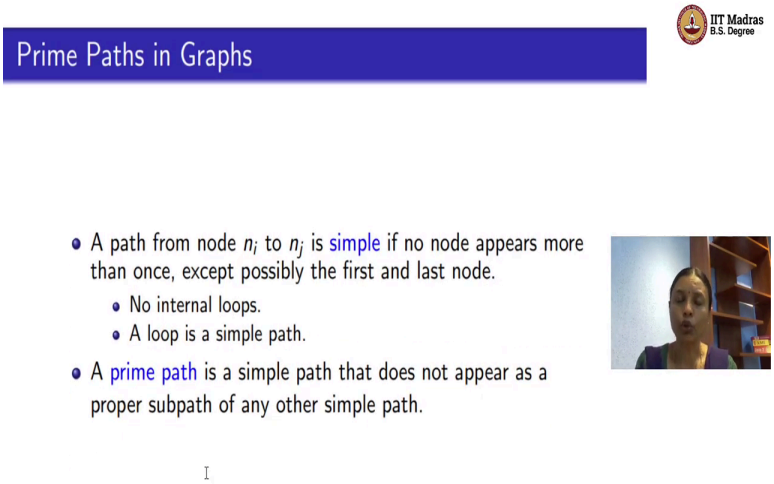
Typically, you never give test cases that will execute the loop once, twice, thrice and so on, all possible iterations, you would not do that. But, once around the normal operations of a loop and the other thing that one is always interested as a tester is to execute the loop at boundary

conditions. That is you skip the loop and see what happens. Suppose the loop was from i equal to 0 to i equal to 10.

Let us say a while loop or a, for loop, you give i equal to 0 and see what happens you give i is equal to 11 and see if the execution skips the loop or not. So, typically, we like to have test cases that cover the loop. As it said here, covering the loop means what we would typically like to have test cases that execute the loop at its boundary conditions, maybe skip, also, and execute the loop maybe once for normal number of iterations.

In the literature of testing, when it comes to graph-based testing, there was a lot of open mean, this, issue was open, how do I efficiently test and then came this proposal through a paper where somebody proposed this notion of what they call a prime path for testing loops. So, the rest of the lecture, we are going to understand what prime paths are about.

(Refer Slide Time: 21:00)

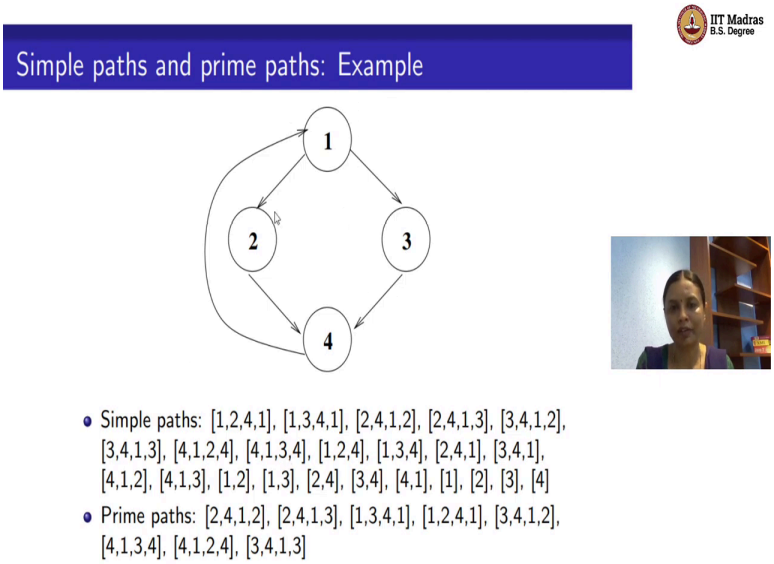


So, to understand the notion of a prime path, you have to first understand the notion of a simple path. So, take a graph, take two nodes, one ni and one n and j, we say a path from node ni to node nj is simple. If no node appears more than once, except possibly the first and the last node that is a path a simple.

If it begins at some node, if it ends a, some node, that is, every path is going to do that. And along the way, no other node repeats. That is, it does not have no internal loops. Is this clear?

That is what we mean by a simple path. A prime path is a simple path that does not appear as a proper sub path of any other simple paths, that is trying paths are maximal simple paths. Is it clear? What is simple path? Is a simple path is a path where a no internal node appears more than once. And a prime path is a maximal simple path. That is, it is a simple path it does not have simple paths inside it so, that is what, is a prime path. So, when we see examples, now it will become clear.

(Refer Slide Time: 22:18)



So, let us take this graph here that you see above in the slide. It has four nodes, 5 edges, and it has a loop. So, there is a loop 1 3 4 1 or 1 2 4 1. It has two loops. So, let us enumerate all the paths in the graph, I have enumerated it in decreasing order of the lengths. So, it might help to read from the sent (())(22:41) from the sent(())(22:42).

So, these four correspond to all paths of length 0, the single vertices, and then I have listed all paths of length 1, the edges of this graph, and then this path contains all paths of length 2, which are the vertices in the graph, I mean, pairs of edges in the graph, and the remaining contains all paths of length 3. This graph has four vertices.

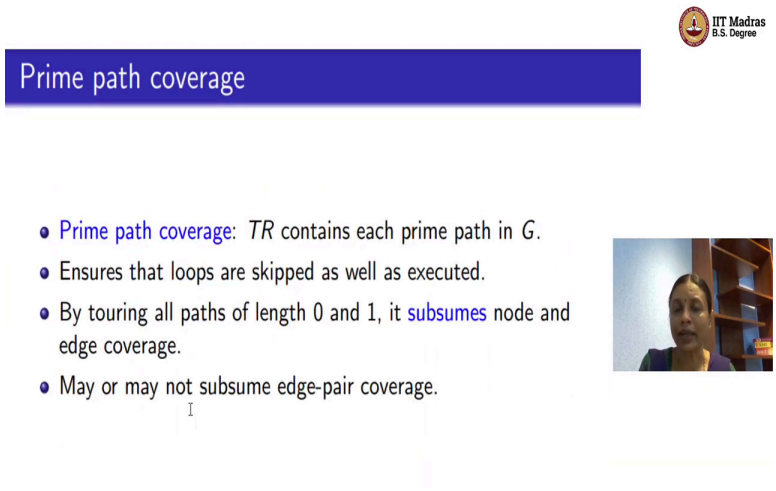
So, it is not too difficult to figure out that if I need a simple path without vertices repeating, then the maximum length a simple path can have is 3. Because the minute a simple path has length 4, which means what it has 5 vertices and 4 edges, but the graph itself has only 4 vertices. So, one

vertex has to repeat, then the path will cease to become a simple path. So, these are all the simple paths of this graph enumerated in order of the decreasing lengths out of these which are prime paths.

So, if you see, all simple paths that are maximal in length are prime paths, like for example, let me take these two simple paths, let me take 1 2 4 here, and 4 1 2 4, so, 4 1 2 4 and 1 2 4. If you see 1 2 4, as a path is completely contained inside 4 1 2 4 as a path. So, when it comes to prime paths, I let go 1 2 4, because it is not maximal.

Instead, I consider 4 1 2 4. So, in this case, it just so happens that all the prime paths are paths of length 3, which are maximal simple paths, but we will soon see other examples where maximal simple paths need not be of maximal lengths**.** So, I hope this definition of water prime path is clear to all of you.

(Refer Slide Time: 24:44)

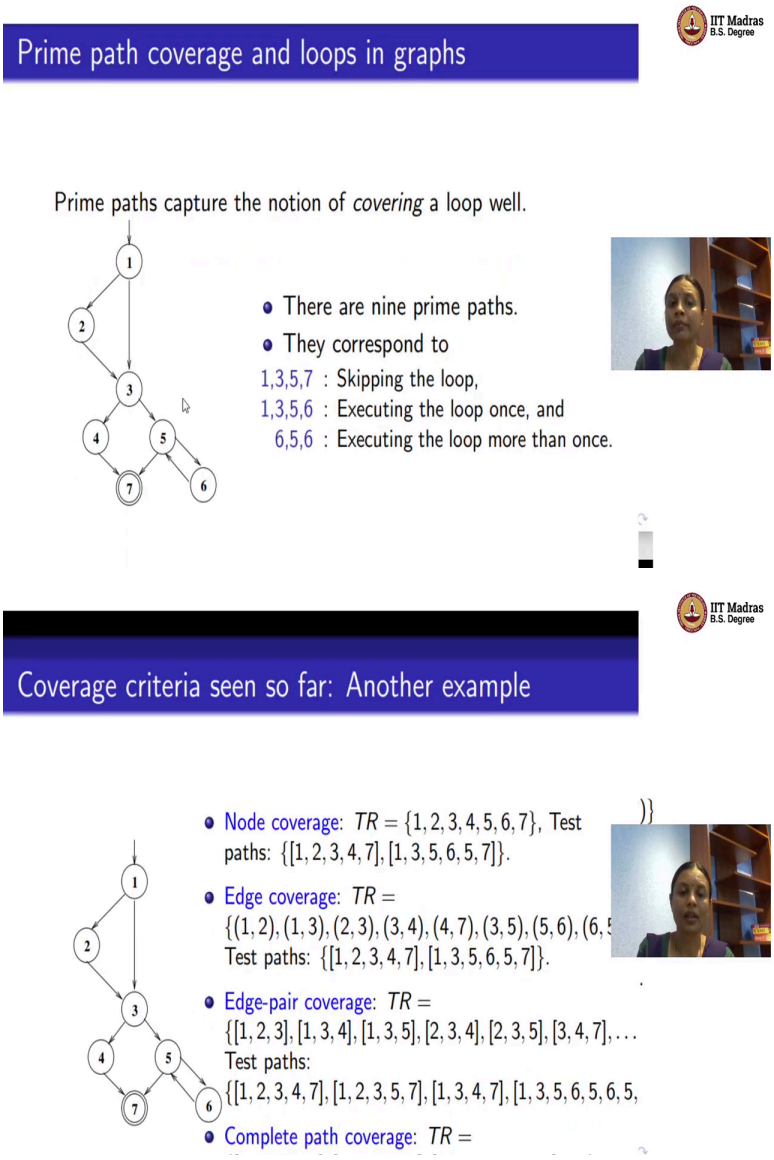


So, I would like to do prime path coverage as the, my next test requirement. The nice thing is when I do prime path coverage as my next test requirement, I just say take a graph enumerate all the prime paths in the graph, we will learn algorithms to enumerate prime paths. And that is your test requirement.

So, the nice thing that happens is when I have my test requirement as enumerate and right test paths that will meet all prime paths, it magically ensures that if there are loops in the graph, each

loop skipped as well as executed. So, it also happens that prime paths subsume node and edge coverage also because they visit paths of length, 0 and paths of length 1.

(Refer Slide Time: 25:33)



I will skip the slide and come back after the example so, will take the same graph that we had in the earlier slide one of the earlier slides. So, this graph that you see on the left has 7 vertices. 1 is the initial vertex 7 is the final vertex, it has one loop. And we enumerated the prime paths for this graph. Let me just go back. I mean, we did not enumerate, but we saw this example.

This is same example here. So, the prime paths for this graph, there will be 9 of them, we will enumerate them when we do a small exercise that will list the prime paths out of these 9 prime paths that you will get in this graph, it will so happen that these three will be a path of those 9 prime paths.

So, 1 3 5 7 is one prime path that if you see 1 3 5 7, carefully studied by putting it on the graph 1 3 5 7, where is the loop in the graph loop is between 5 and 6. So, this is a prime path that keeps the loop. So, it corresponds to testing the boundary conditions of the loop. The next prime path is 1 3 5 and 6.

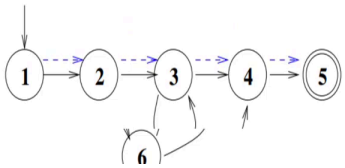
So, this prime paths as a prime path will correspond to entering the loop once when you enter the loop, by entering from 5 to 6, I am forced to execute the loop once. So, that is this prime path. 6 5 6 will also be listed as one of the prime paths that corresponds to executing the loop more than once. So, we will see the same graph and enumerate all the prime paths in detail.

(Refer Slide Time: 27:21)

IIT Madras B.S. Degree

Implementing test requirements for prime path coverage

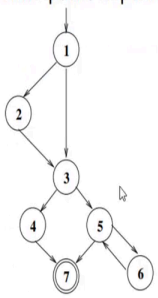
- Prime paths, by definition, do not have internal loops.
- In many cases, it might be impossible to meet the test requirement of prime path coverage without internal loops.
- That is, test paths that meet prime path coverage TR need have internal loops to make prime path coverage feasible.



IIT Madras B.S. Degree

Prime path coverage and loops in graphs

Prime paths capture the notion of *covering* a loop well.

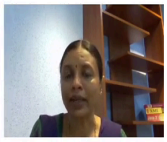


- There are nine prime paths.
- They correspond to

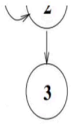
1,3,5,7 : Skipping the loop,
1,3,5,6 : Executing the loop once, and
6,5,6 : Executing the loop more than once.

Prime path coverage

- Prime path coverage: *TR* contains each prime path in *G*.
- Ensures that loops are skipped as well as executed.
- By touring all paths of length 0 and 1, it subsumes node and edge coverage.
- May or may not subsume edge-pair coverage.



Prime path coverage vs. edge-pair coverage



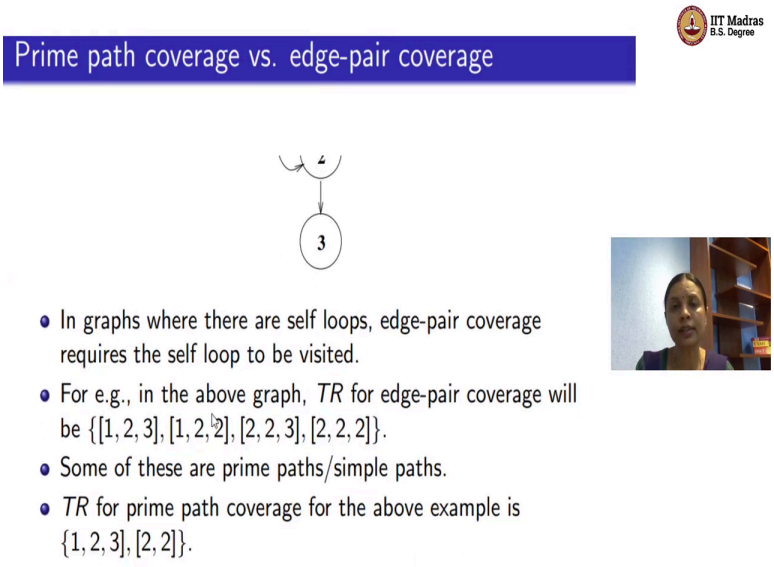
- In graphs where there are self loops, edge-pair coverage requires the self loop to be visited.
- For e.g., in the above graph, *TR* for edge-pair coverage will be $\{[1,2,3],[1,2,2],[2,2,3],[2,2,2]\}$.
- Some of these are prime paths/simple paths.
- *TR* for prime path coverage for the above example is $\{1,2,3],[2,2]\}$.

So, prime paths, by definition do not have internal loops, so they are maximal simple paths. So, it will so happen like as we saw in this example, that when I enumerate all the prime paths for a graph, I will end up getting paths that will skip the loop that will execute the loop once by entering the loop and I can reuse paths like this 6 5 6 to execute the loop as many times as I want.

So, this is one path about prime path and it ensures that all loops are skipped as well as executed. The next thing is by touring all paths of length 0 and one prime path coverage subsumes both node and edge coverage. The other thing to node is that it may or may not subsume edge pair

coverage that is dependent on the structure of the graph. So, here is an example of why prime paths may not subsume edge pairs.

(Refer Slide Time: 28:16)

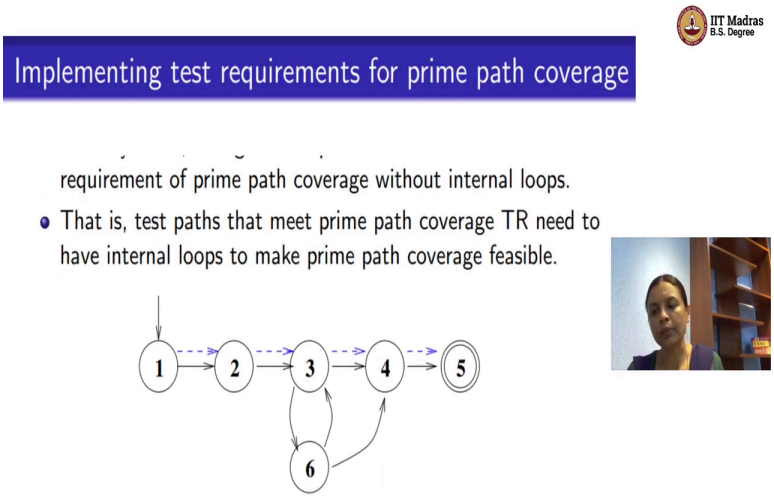


So, you take this graph on top given here 1 2 3, there is a self loop at node 2. So, in graphs whether a self loops, edge pair coverage requires the self loop to be visited for example, the test requirements for edge pair coverage for this graph will be what 1 2 3, 1 2 and then this edge 2, then 2 and the edge 2 that is this loop 2 2 edge 2 2 and 2 3 and then I should also do 2 2 and 2 that is visit that edge twice.

If you see in these this part 2 to 2, the last one is clearly not a prime path this is not a prime path, because there is a repetition of the internal vertices. The first is the only prime path second is not a prime path third is not a prime path fourth is not a prime path also. So, there are edge pairs, paths that correspond to edge pairs that are not necessarily prime paths.

And for this specific example, so, in general, I cannot say that if I do prime path coverage, I will end up covering all the pairs of edges. For this example, the test requirements for prime path happens to be this 1 2 3 and then 2 2, which is visit the loop. Obviously, this set for, each pair coverage, is not contained in this set for prime path coverage. So, in general, prime path coverage may not subsume edge pair coverage.

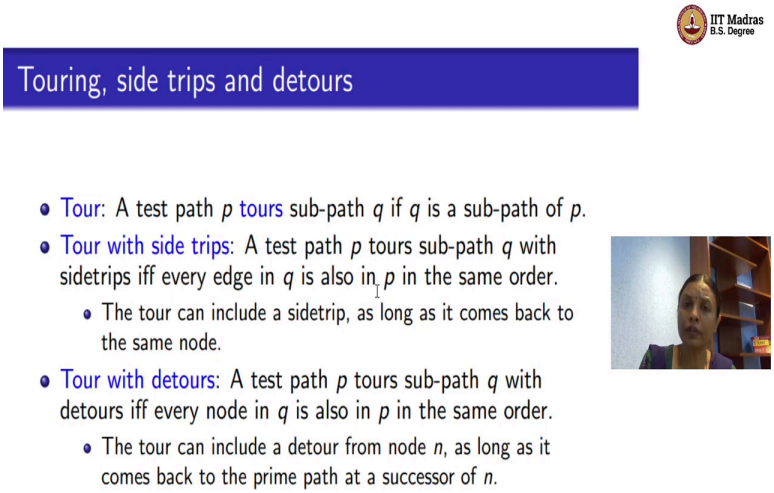
(Refer Slide Time: 29:58)

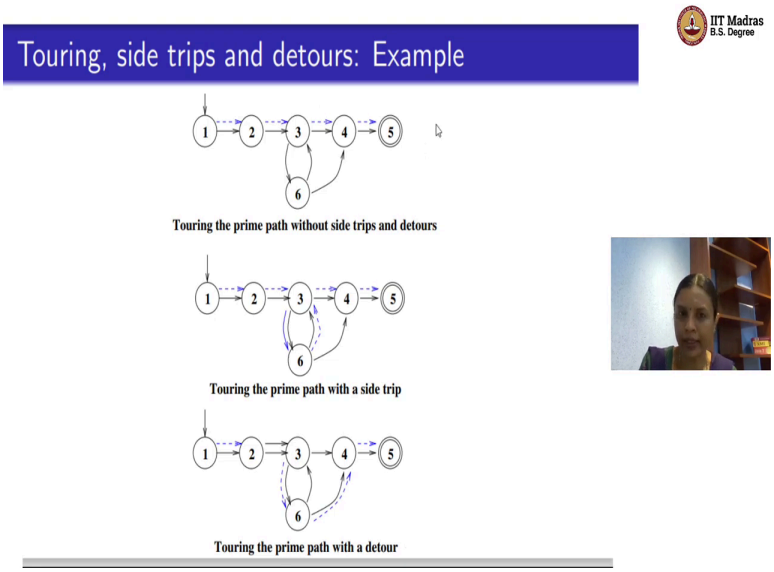


So, as I told you, there is one more settle point that we need to know, prime paths, by definition do not have any loops internally, because they are maximal simple paths. But sometimes, to cover a prime path, you are forced to visit a loop you are forced to visit a loop. And prime paths do not have loops, it just so happens that the method for which you are generating the control flow graph happens to have loops, and enumerated prime paths.

And these prime paths actually like for example, if you see in this example, here in the graph below, look at this path that has colored this blue dashed edges, so 1 to 2, 2 to 3, 3 to 4, 4 to 5, it happens to be a prime path that skips this loop 3 6, but it is so it might so happen that in the real code for which this corresponds to as a control flow graph, it might be impossible to execute this path without visiting the loop 3 6. So, what we tried to do is we introduced two simple notions in testing, called touring.

(Refer Slide Time: 31:07)





With side trips and touring with detours, what is a tour? So, we say a test path p towards a sub-path q, if q happens to be a sub path of p, you can consider two kinds of tours, tours with side trips, and tours with detours in a tour with side trip, what will happen is, the edges that come in the sub path q will also come in the sub path p in exactly the same order.

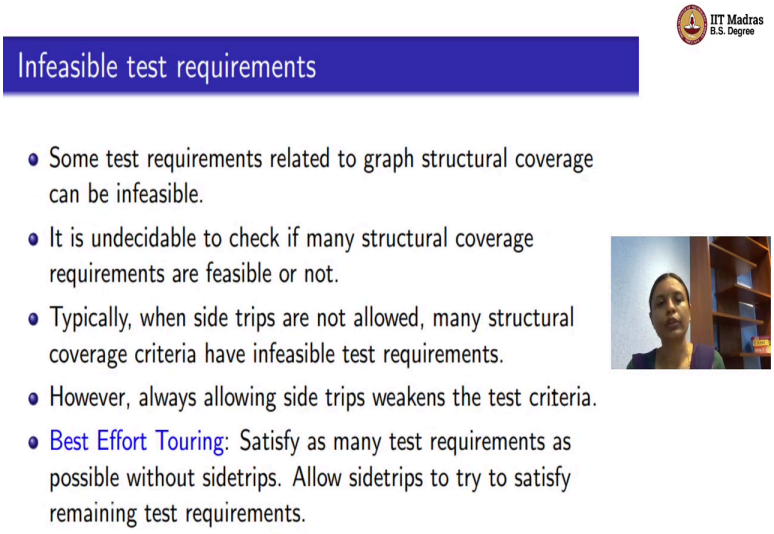
So, that is a tour with a side trip. I will show you an example very soon. In a tour with the detours, the vertices or the nodes that come in q will come in exact in p also in exactly the same order. So, that is the difference between side trip, and detours. So, we will take this example that we had here.

On the graph on top graph with 6 nodes vertex 1 is the initial 1, vertex 5 is the final 1, my goal is to achieve this path colored in blue. And I am allowed to achieve the test requirement of this prime path colored in blue without side trips, which will be directly this path 1 2 3 4 5. But for some reason, if I am forced to take this loop from 3 to 6, I can do a detour with a side trip or a side a side trip.

So, I can do like this 1 2 3, 3 2 6, 6 back to 3. So, I do visit the loop 6 to 3, but I call it as a side trip by calling it as a side trip. I am basically telling myself that it is not a part of the main prime path it is just a side trip. And then I go back 3 to 4, 4 to 5, or I could even do this 1 to 2, 2 to 3, 3 to 6, 6 to 4, 4 to 5. In this case, I also tour the prime path but with a detour.

So, if you see here the edges in a side trip come in the same order as the edges in the original test requirement. The vertices in the detour come in the same order as the vertices in the original requirements. That is what this definitely is. So, a tour a path p towards a sub path q, if q is a sub path of p tours with side trips, test path p towards sub path q with side trips, if every edge of q is also in p in the same order for detours, every node of q is also in p in the same node. Is it clear?

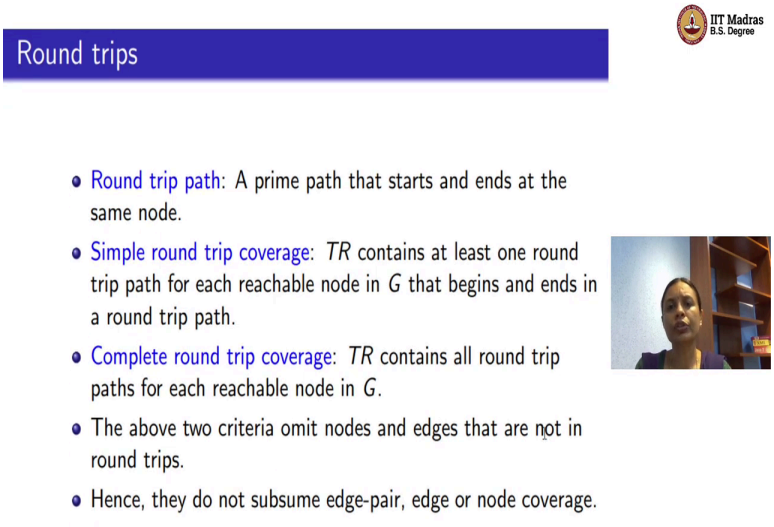
(Refer Slide Time: 33:48)



As I told this is basically what I have covered. Actually, we saw complete path coverage was infeasible as a test requirement. Sometimes things can be infeasible. And I do not worry if it is infeasible, I do not have algorithms to check for infeasibility. It is a undecidable problem. But I

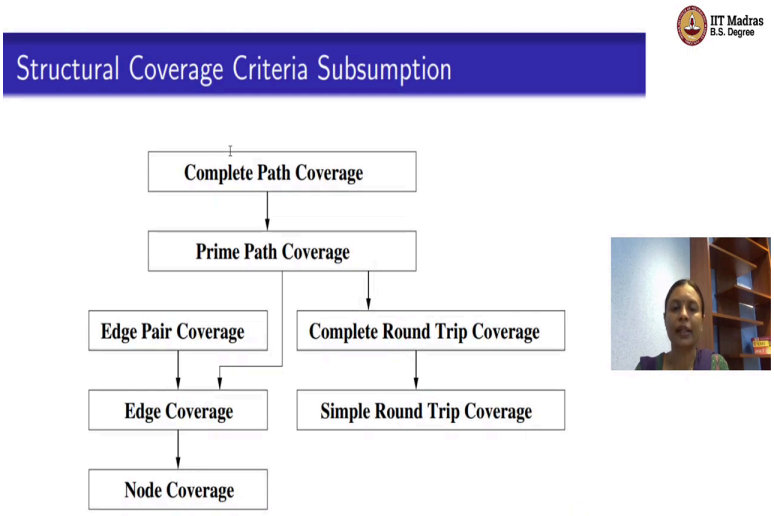
try to achieve feasibility by considering side trips and detours whenever it is possible. So, when I try to achieve beat in feasibility by considering side trips, and detours, we call it as the best effort touring. That is all that is all this is.

(Refer Slide Time: 34:22)



So, what are round trips? The prime path that starts and ends and the same node is called a round trip. Specifically, sometimes we might do this and coverage criteria for round trip says you do simple round-trip coverage or you do complete round trip coverage, simple round-trip coverage says at least one round trip for each reachable node should be their complete round trip says all the round-trip paths should be covered. So, round trips as a separate thing on their own very rarely used in testing, I have provided them here for the sake of completeness.

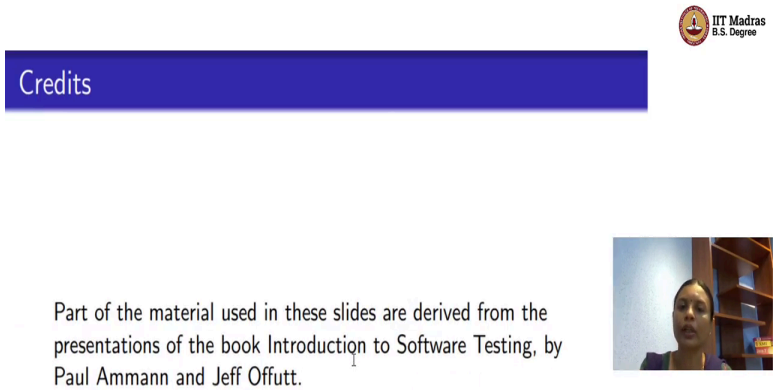
(Refer Slide Time: 35:00)



So, in summary, these are all the coverage criteria that we saw. Let me start from this lower end here. Node coverage most elementary coverage criteria, visit each node exactly once at least once then edge coverage visit each edge at least once edge pair coverage visit paths of length up to 2 at least once, prime path coverage very useful for covering loops, skip the loop execute the loop once execute the loop more than once.

We will see in the next lecture how to get algorithms that will enumerate prime paths and in the next week one of the early lectures I will actually tell you how prime paths are useful to test loops. And then we have complete path coverage only for theoretical reasons. In graphs with loops we saw complete path coverage is infeasible. Sometimes we will do round trips, which are prime paths that begin and end at the same node so you can have complete round trip coverage and simple round-trip coverage so, this was a summary of structural graph coverage area and if you would like to know more.

(Refer Slide Time: 36:04)



You can look at the textbook. That we are going to follow for this part of the course which is this text book, by Ammann and Jeff Offutt.