

Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 06
Structural Graph Coverage Criteria

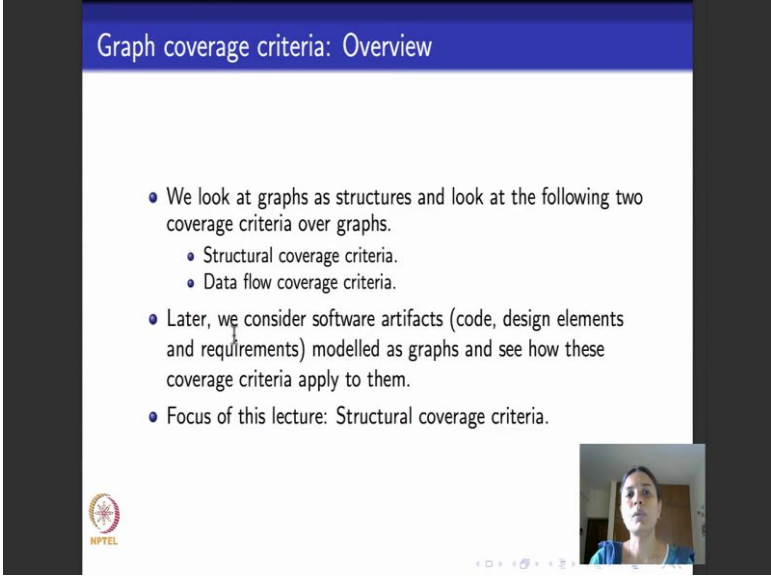
Hello everyone. Now we are in the second week. What we looking at this week is basically you look at graphs and to look at various algorithms that we can use for test case design using graphs. In the last module I gave you a brief background of graphs as it was necessary for this course. So, to recap what we saw in the last module: we saw what graphs where and in the basic concepts like degree of vertex, what is the notion of paths, and what are trips, what is visiting a node and few are the details. So, the plan is to be able to use graphs to design test cases. So, we take software artifacts like code, requirement, design, model them as graphs and see how to design test cases, how to define coverage criteria and then to design test cases using graphs.

So, in the first two modules that we will see as a part of graph based testing what I will do is we look at coverage criteria purely based on graphs. I will not really show you too many examples of how software artifacts are modeled as graphs. Instead we directly deal with graphs as data structure it is and define coverage criteria. In today's lecture we will define coverage criteria that are based on the structure of the graphs. That are based on nodes, vertices, edges, paths and so on.

The next lecture I will look at some algorithms and deal with coverage criteria that are again based on the structure of the graph. Moving on what we will do is when annotate the graph the vertices and the edges of the graph with statements and other entities. And we will define coverage criteria based on data that deals it with in the graphs. After we do these three kinds of coverage criteria, look at how they are related to each other, and how they can be use to design test cases based on graphs we will consider a later module where we will take various kinds of software artifacts; we will begin with code, then to go on to design, then move on to requirements.

Model each of them as graphs and then see how the coverage criteria that we will be learning throughout these modules can be used to actually design test cases for covering the software artifacts.

(Refer Slide Time: 02:29)



Graph coverage criteria: Overview

- We look at graphs as structures and look at the following two coverage criteria over graphs.
 - Structural coverage criteria.
 - Data flow coverage criteria.
- Later, we consider software artifacts (code, design elements and requirements) modelled as graphs and see how these coverage criteria apply to them.
- Focus of this lecture: Structural coverage criteria.

NPTEL

A small video inset in the bottom right corner shows a woman with dark hair, wearing a blue top, speaking.

So, what are we be doing at. So, first I will today's module I will deal with structural coverage criteria, in the next module also we look at structural coverage criteria but we will focus on algorithms, and then we will look at what is called data flow coverage criteria which consider graphs with data, variables and they values and then define coverage based on them. As I told you post this, we will take it has several software artifacts one at a time module them as a graphs and see how to use these coverage criteria to design test cases. So, we begin with coverage criteria in this course.

(Refer Slide Time: 03:03)



Structural coverage criteria over graphs

- Node/vertex coverage.
- Edge coverage.
- Edge pair coverage.
- Path coverage
 - Complete path coverage.
 - Prime path coverage.
 - Complete round trip coverage.
 - Simple round trip coverage.

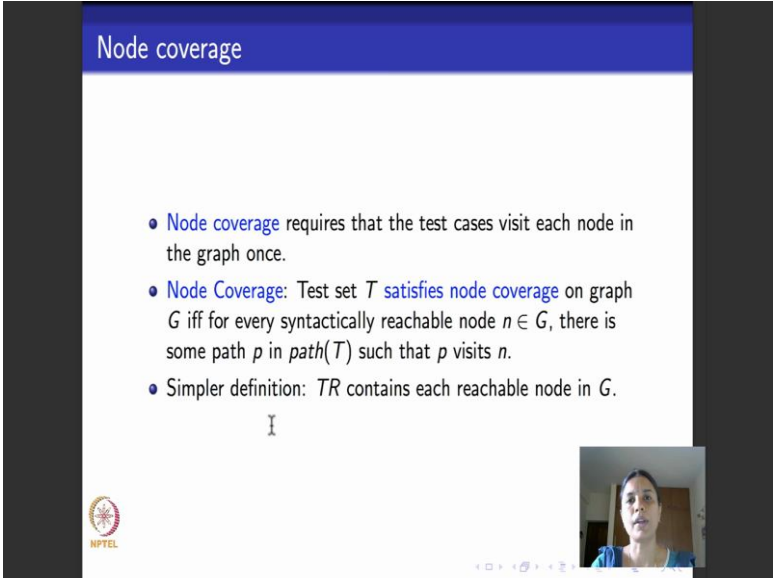
NPTEL

A small video inset in the bottom right corner shows the same woman from the previous slide, continuing her presentation.

So, what are the various coverage criteria that we are going to see? Here is the listing. So, we will begin with what is called node coverage or vertex coverage. We will define a test requirement which says you have to cover/visit every node. And then we will define test coverage requirements for edge coverage where we say we visit every edge. Then we look at edge pair coverage which insists on visiting every consecutive pairs of edges which are paths of lengths two. Then we look at path coverage in the graph. In path coverage these are the various things that we look at. We look at complete path coverage which may not be feasible all the time.

And we will move on and look at very popular coverage criteria called prime path coverage. And then to make prime path coverage feasible we look at these round trip coverage criteria. So, we will see each of these one at a time.

(Refer Slide Time: 03:59)



Node coverage

- **Node coverage** requires that the test cases visit each node in the graph once.
- **Node Coverage:** Test set T satisfies node coverage on graph G iff for every syntactically reachable node $n \in G$, there is some path p in $path(T)$ such that p visits n .
- **Simpler definition:** TR contains each reachable node in G .

NPTEL

So, we begin with node coverage, what is node coverage? It simply insists that you have a set of test cases that visit every node in your graph. So, it could be the case that the graph corresponding to a particular software artifact can be connected or disconnected. If it is disconnected then the graph has several disjoint components. What do we mean by disjoint components? There are no edges between vertices of these components.

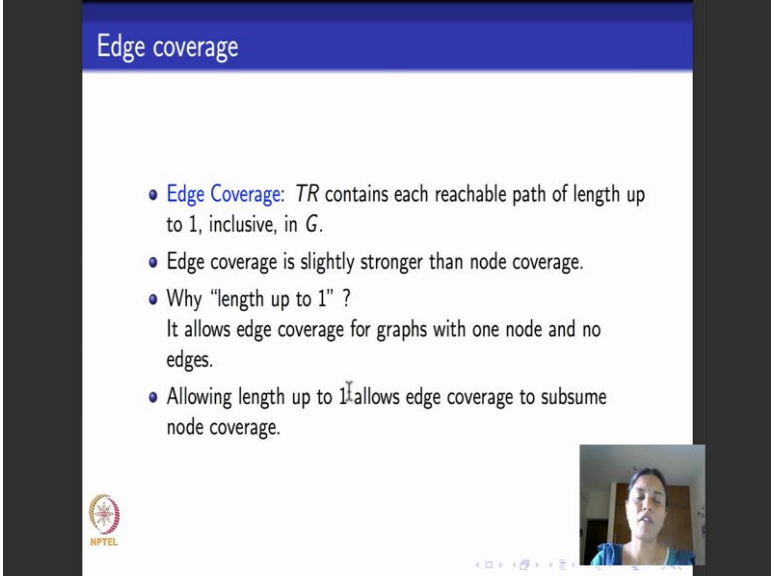
So, in which case if I see node coverage needs to visit every node, you might ask how do I go and visit other nodes then the graphs. So, what we insist is the node coverage visits every reachable node; every reachable node means every node that can be reached from

the designated initial node. So, what we mean is that per component per reachable component that is connected, you visit every node.

So, what is node coverage? Node coverage says, the test requirement for node coverage abbreviated as TR, says you basically right a set of test cases that will visit each node in the graph. Now how will you write a set of test cases that will visit each node in the graph? What will a test cases look like? How can you go about visiting nodes in a graphs. You have to start from a initial state and you have to walk along paths in the graphs. As you walk along paths you visit various different nodes.

So, set of test cases that will meet the test requirement on node coverage would be a set of test paths that begin at an initial state and visit every reachable node in the graph. I will show you an example after a couple of slides.

(Refer Slide Time: 05:40)



The slide is titled "Edge coverage" in a blue header. It contains four bullet points:

- **Edge Coverage:** TR contains each reachable path of length up to 1, inclusive, in G .
- Edge coverage is slightly stronger than node coverage.
- Why "length up to 1" ?
It allows edge coverage for graphs with one node and no edges.
- Allowing length up to 1 allows edge coverage to subsume node coverage.

In the bottom right corner, there is a small video inset showing a person speaking. The NPTEL logo is visible in the bottom left corner of the slide area.

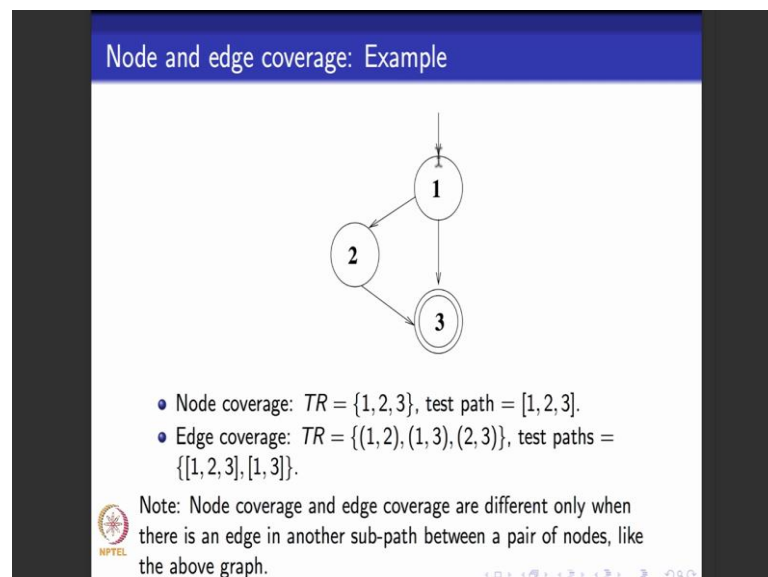
And then the next coverage criteria that we will be looking at; is what is called edge coverage. So, light node coverage edge coverage basically insists that your visit every edge in the graph. So, your test requirement says you visit each requirement path of length up to 1. So, you might ask when I say visit every edge why am I writing it as visit each reachable path of length up to 1? Now look at an edge in a graph what is an edge look like an edge can be thought of as a path of length 1, because which is connects two vertices. But instead of saying visit every path of length 1 which basically means visit every edge, we are slightly modifying the conditions to say visit every path of length up

to 1, which means you visit paths of length is 0 and paths of length 1. What do paths of length 0 correspond too? Paths of length 0 correspond to paths the just contains single vertex.

We want to be able to say that edge coverage subsumes node coverage in the sense that if I have a set of test paths that satisfy edge coverage then those set of test paths will also satisfy node coverage. Because we want to be able to say so, we modify edge coverage criteria definition to include that it visits every path of length up to 1.

So, it visits every single vertex and it visits every single edge. So, by definition edge coverage will subsume node coverage.

(Refer Slide Time: 07:17)



Here is an example to illustrate how edge and node coverage work. So, here is a small three vertex graph: vertex one is the initial vertex. As you can see, its mark to design coming arrow. Vertex three is the final vertex it is marked with its double circle and it is a small graph. If you remember in the previous example we had a new statement and I showed you how to model the control flow graph corresponding to that if statement remember, which did not have a else part just had the then part. This was the graph that correspond to the control flow graph of a that if statement.

I of course, remove the labels in the edges all that stuff, because we are looking at purely structural coverage criteria which defines coverage criteria based on the basic structure

of the graph; does not really look at labels of edges of vertices. So, in this graph what would be node and edge coverage. Node coverage: the test requirement for node coverage says that there are three nodes so you please visit all three reachable nodes, all three of them are reachable here. So, how many paths can visit all three reachable nodes? So, I could take this path I start from 1 and node 2 and then to 3. In this path I have visited all the three nodes 1, 2 and 3. There is another possible path in the graph I could start from 1 and then I go to 3. Suppose I take this as the test path corresponding to the test requirement then I have only partially achieved node coverage, I have visited the nodes 1 and 3 I have not visited the nodes 2.

There is no harm in choosing that, suppose you choose that then you have to add this path also 1, 2 and 3. But instead, I could directly choose this path 1, 2 and 3 as my test path. And in that case I would directly met the test requirement of node coverage which is one path.

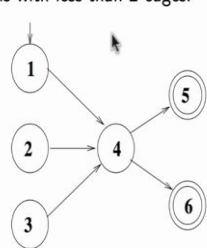
Now, let us look at edge coverage how many edges are there in the graph? There are three edges in the graph: one edge from 1 to 2, one edge from 1 to 3 and another edge from 2 to 3. What is edge coverage mean? The test requirement or TR, for edge coverage says you visit all the three edges. The edge 1, 2 the edge 1, 3 and the edge 2, 3. Here if you see suppose I take this path 1, 2 and 3. How many edges do I visit in this path? I visit 1, 2 and 2, 3. So, I visit two edges so that is this path here. But I have to be able to visit the edge 1, 3 so I have to take this path.

So, to meet the test requirement for edge coverage and this graph I need two test paths; the test path 1, 2, 3 and then the test path 1, 3. This is another small observation. Suppose the graph is a sort of a straight line right, there are no branching and it just to corresponds to some core the corresponds to sequence statements. It just like one linear order or a straight line or a chain. In that case node and edge coverage are same, because when I visit every edge there is only one path in the graph I visited the path and in the process I visited every node also. So, node and edge coverage become different in terms of the test paths that satisfy it only if there is a branching in the graph.

(Refer Slide Time: 10:24)

Covering Multiple Edges: Edge-Pair Coverage

- **Edge-Pair Coverage (EPC):** TR contains each reachable path of length up to 2, inclusive, in G .
- Paths of length up to 2 correspond to pairs of edges.
- Again, the phrase "length up to 2" ensures edge-pair coverage holds for graphs with less than 2 edges.



$TR = \{[1, 4, 5], [2, 4, 5], [3, 4, 5], [1, 4, 6], [2, 4, 6], [3, 4, 6]\}.$

• Test paths are the same as above.

So, we move on. What is the next coverage criteria that were there in my list? Go back to that slide. So, we did node coverage which basically says visit every node. So, you write test path that visit every node edge coverage says visit every edge once, so you write test paths that visit every edge. Now will move on to edge pair coverage and then look at path coverage.

Before we move on to edge pair coverage I would like you to spend a minute thinking about what would node coverage, how would node coverage and edge coverage be useful. Assuming that such a graph is a control flow graph corresponding to a program what can you think of node coverage and edge coverage put together they basically mean you execute every statement in the program.

Because they insist that you visit every node and then they insist that you visit every edge. If you write a set off test path that satisfy this then you are writing a test path that basically execute every statement in the program. So, you are looking for a set of test cases that will exercise for test every statement in the program. Node coverage and edge coverage might be quite difficult to achieve, because for large pieces of program where the control flow graph is fairly large, writing a set of test cases that will achieve execution of every statement, every node or every edge, can be a large set of test cases and sometimes infeasible also.

So, we look at other state of test cases. The next set of structural coverage criteria that I would like to talk about is what is called edge pair coverage. So, as a name says, your test requirement here is to actually consider pairs of edges; not pairs of edges that are far away from each other but pairs of edges that are consecutive to each other. In other words, you consider paths of length up to 2. Paths of length up to 2 include paths of lengths 0 which include nodes, paths of length 1 which are edges and paths of length 2 which are actually sets of edges that occur one after the other. So, if you take this example graph, how many paths of lengths two are there? All these six paths are paths of lengths 2. So, I start from 1 which is an initial vertex and 5 and 6 are final vertices. So, I go from 1 to 4, 4 to 5 that is a path of length 2 pair of edges then I go from 1 to 4, 4 to 6 another pair of edges that are consecutive; 2, 4, 5, another pair of edges 2, 4, 6; 3, 4, 5; 3, 4, 6. So, my test requirement says- you visit all these paths of length up to exactly once. So, here they have written length up to 2, but I have listed here paths of lengths 2. I implicitly assume that it includes paths of length 1 and paths of length 0.

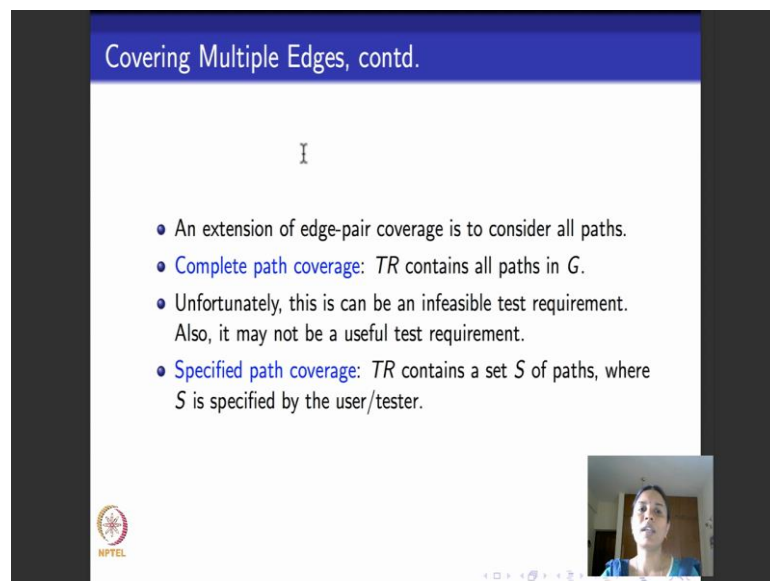
So, how many test requirements can I write? If you look at this structure of the graph it is so happens that I need one path for meeting each of these Trs, I cannot do better. So, test paths or basically all these six things, I cannot write anything shorter. Now why do I need paths of lengths up to 2, the same reason we said when we do edge coverage even though edge is a path of length 1 we changed it as path of length up to 1, because we wanted edge coverage to subsume node coverage. So, for the same reason we want edge pair coverage to subsume node coverage and edge coverage. So we insist that the TR contains all paths of length up to 2.

Suppose you do not you are not very particular about this requirement there is no harm in saying that edge pair coverage means, you visit all paths of length exactly 2. So now, before we move on what you think edge pair coverage would be useful for? One good use of edge pair coverage is to cover all branches in the program. So, if you think of this as the control flow graph representing some piece of code, let's not worry about what that piece of code is, but let us assume that this is the CFG for some code.

What could node four be? Node four could be corresponding to an if statement right, which says it has two branches: if then is true may be you go to 5 if else true may be you go to 6 right due to something here you do something here. So, when I do edge pair coverage what I cover is from 1, 2 and 3, what are the various ways and which I can visit

4, and from 4, what are the various ways and which I can cover the two branches that go out of 4? So, edge pair coverage is useful. Suppose I think about it what did I do I did paths length 0 which was vertices no coverage, then I did paths of length 1 which were edges edge coverage. Now I say paths of length up to 2, I did edge pair coverage you can move on right you can say paths of the length 3, paths of length 4, paths of length 5. Then if we go on like this, what we have is what is called complete path coverage.

(Refer Slide Time: 15:17)



Covering Multiple Edges, contd.

- An extension of edge-pair coverage is to consider all paths.
- **Complete path coverage:** TR contains all paths in G .
- Unfortunately, this can be an infeasible test requirement. Also, it may not be a useful test requirement.
- **Specified path coverage:** TR contains a set S of paths, where S is specified by the user/tester.

NPTEL

You have a test requirement TR which says that you cover all paths in the graph. Now if you think about it what would be the use of this? Will it be useful? What would all paths mean? If I say cover all paths on the graph let us assume a case where the graph has a loop. In the two examples that we saw in the previous two slides, the graph did not have a loop, but let us assume the graph has a loop. How many paths do you think will be there in the graph? They will be infinite number of paths in the graph, because I visit the loop once I get one path, and I visit the loop once again I will get another path, I visit a loop once again I get another path. So, I can go round the loop again and again and again and get more and more paths, longer and longer paths.

So, there will be infinite number of paths. And given these infinite numbers of paths how do I achieve complete path coverage? I will have to be able to go on writing test paths. So, for this reason complete path coverage is believe to be an infeasible test requirement. Infeasible in the sense that I will never be able to stop testing- if I want to achieve

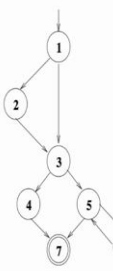
complete path coverage intuitively. Also if you think about it not only is it infeasible, what is a use of it? It may not be very useful for us. What is the point you going on repeatedly executing a loop once, twice, thrice, four times and so on? It is not interesting, infeasible and not very useful.

So, what we say is that we will list complete path coverage as a test requirement, but it is not practically usable. What is practically usable is what is specify path coverage. What does specified path coverage say? It says that the use as a test engineer will give you the set of paths to cover. The TR will be a set of paths that is specified by a test engineer or a user which says you cover these paths. So, paths could be some special things, we do not know what they are, but the test is basically gives a set of paths and says you this is your test requirement. Please write a set of test cases that will cover this path.

So, we see specify path coverage, your test requirements contained a set of paths where that is set is specified by the user.

(Refer Slide Time: 17:44)

Coverage criteria seen so far: Another example



- **Node coverage:** $TR = \{1, 2, 3, 4, 5, 6, 7\}$, Test paths: $\{[1, 2, 3, 4, 7], [1, 3, 5, 6, 5, 7]\}$.
- **Edge coverage:** $TR = \{(1, 2), (1, 3), (2, 3), (3, 4), (4, 7), (3, 5), (5, 6), (6, 5), (5, 7)\}$
Test paths: $\{[1, 2, 3, 4, 7], [1, 3, 5, 6, 5, 7]\}$.
- **Edge-pair coverage:** $TR = \{[1, 2, 3], [1, 3, 4], [1, 3, 5], [2, 3, 4], [2, 3, 5], [3, 4, 7], \dots\}$
Test paths: $\{[1, 2, 3, 4, 7], [1, 2, 3, 5, 7], [1, 3, 4, 7], [1, 3, 5, 6, 5, 6, 5, 7]\}$.
- **Complete path coverage:** $TR = \{[1, 2, 3, 4, 7], [1, 2, 3, 5, 7], [1, 2, 3, 5, 6, 5, 7], \dots\}$.

So, will go and now move on and see what is one specific, useful specified path coverage. Before that I will give you another example just to recap all the coverage criteria that we have seen so far. So, what are the folk structural coverage criteria that we saw so far? We saw node coverage, edge coverage, edge pair coverage and complete path coverage. So, let us take a small graph. Here is a graph on the left hand side, it has about 7 nodes, the node 1 is the initial node, and there is one final node which is node 7.

If you see this could constitute a reasonably interesting control flow graph. So, if you try to map it to assume that it is a control flow graph corresponding to some code there is branching at statement one. So, maybe there was an if statement here. There is another branching at statement 3. There is a another branching statement 5. One branching ends in the final state 7, one branching goes into a loop between 5 and 6. So, maybe there was a while statement here, this means skipping the loop and this means executing the loop.

So, what will node coverage on such a graph be? Node coverage on such a graph--- the test requirements says there are 7 nodes, please visit all 7 nodes. How many test path will visit all the 7 nodes? There could be one test path like this I do 1, 2, 3 4 7. This is the path, this test path from the initial node 1 to the final node 7.

So, in this test path I have visited how many nodes? I have visited 5 of my 7 nodes I visited nodes 1, 2, 3 4 and 7. What are the node 7 left behind; 5 and 6. So, I have to write another test path which includes that. So, again because it is a test path I have to begin from an initial state and end in a final state. So, I begin at node 1, then I go to 3, I do not want to go to 4 because I have already considered that in my test path; from 3 I go to 5; and then I do not want to go to 7 because if I do that then I will miss visiting node 6, so from 5 I visit node 6. And remember it is a test path so I have to be able to end in a final state. So, I come back to 5 and then I do 7, so that is the test path.

So, just to summarize node coverage test requirements says you visit all the 7 nodes which is the set. And what are the test paths that will visit all the 7 nodes there are two test paths one which goes like this 1, 2, 3, 4, 7 which leads of nodes 5 and 6. So, I put another test path which says 1, 3, 5, 6, 5, 7. So, these two test paths put together satisfy this test requirement of node coverage. Similarly on this graph how do I do edge coverage? How many edges are there? So many edges are there right; (1, 2); (1, 3); (2, 3); (3, 4); (4, 7) and so on.

So, test requirement says you visit all this edges which is the set of all edges in the graph. What are the two test paths a very similar to node coverage, because I told you right edge coverage also meant to subsume node coverage. So, if I do this test path 1, 2, 3 4 7 I visited all the edges that occur on this set. Now, I write another test path that lead covers a missing edges which is this (1, 3); (3, 5); (5, 6); (6, 5) and so on. So, between these I have done edge coverage the next test requirement is edge pair coverage. Edge pair

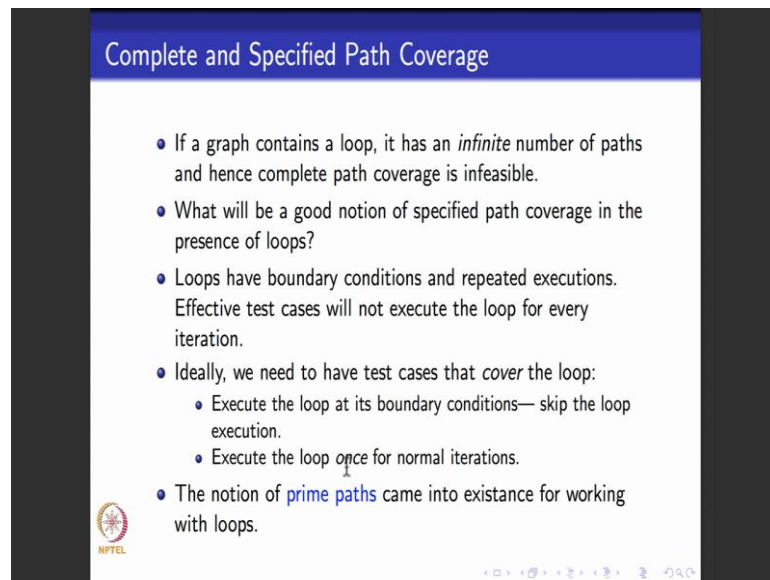
coverage lists all paths of length 2 which is 1, 2, 3; 1, 3, 4; 1, 3, 5 and so on that put this dot dot dot because I ran out of space to list all the edge pairs, but you can finish this its only the finite sets slightly larger than edge set that is my TR.

So, please write path then visit all this path of lengths 2 that would means it will visit all these consecutive pairs of edges. So, test paths for these would be if you see 1, 2, 3, 4, 7 it covers edge pairs that occurs a longest path and then I do 1, 2, 3, 4. I forgot this pair so 2 to a 3, 3, 5, so I do a 3, 5 7 which covers all test paths along these lines and then I do 1, 3, 4, 7 which includes this edge pair. Now I have to include 1, 3, 5, 6 5, 7 which completes all the edge pairs that I had.

So, with this four test paths I can achieve the test requirement for edge pair coverage. Now complete path coverage for this graph- please remember this graph has a loop here from 5 to 6. So, complete path coverage for this graph test requirement will not stop, it will go on listing paths, because I have this path it skips the loop, I have this path which also skips the loop for then I have take this path which enters the loop. Once I enter the loop I have a path that looks like this which visits the loop once. Then I can do the same thing again 1, 3, 5, 6, 5, 6, 5, 6, 7 and then I can do 1, 3, 5, 6, 5, 6, 5, 6, 5, 6, 7 then I can go on writing.

So, test requirement for complete path coverage for this graph with a loop will be an infinite set. So obviously, I am not going to be able to make it feasible or write a test path, set of test paths that will execute complete path coverage. Its only for completeness we really do not consider it is a useful coverage criteria by any means.

(Refer Slide Time: 23:02)



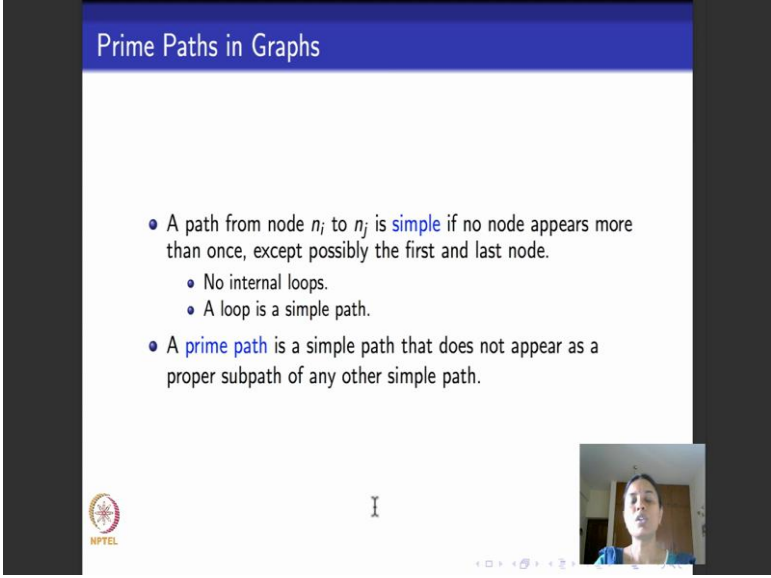
The slide is titled "Complete and Specified Path Coverage" in a blue header. It contains a bulleted list of points. The first point states that if a graph has a loop, it has an infinite number of paths, making complete path coverage infeasible. The second point asks for a good notion of specified path coverage in the presence of loops. The third point notes that loops have boundary conditions and repeated executions, and that effective test cases should not execute the loop for every iteration. The fourth point states that ideally, test cases should cover the loop, with sub-points: "Execute the loop at its boundary conditions— skip the loop execution." and "Execute the loop *once* for normal iterations." The fifth point mentions that the notion of "prime paths" came into existence for working with loops. An NPTEL logo is in the bottom left corner, and navigation icons are in the bottom right.

- If a graph contains a loop, it has an *infinite* number of paths and hence complete path coverage is infeasible.
- What will be a good notion of specified path coverage in the presence of loops?
- Loops have boundary conditions and repeated executions. Effective test cases will not execute the loop for every iteration.
- Ideally, we need to have test cases that *cover* the loop:
 - Execute the loop at its boundary conditions— skip the loop execution.
 - Execute the loop *once* for normal iterations.
- The notion of *prime paths* came into existence for working with loops.

Now we will see how to overcome this problem about complete path coverage. I have this graph with loops; the only way I have touched upon this loop is to be able to do complete path coverage. But then that is not very useful because it gives rights to an infinite number of paths. So, is there a mid way solution? So, the question that we want to ask is what will be a good notion of specified path coverage in the presence of loops. Now let us look at loops. Assuming that this kind of control flow comes from a loop, when we want to test a loop what do we ideally want to test? We say that loops have boundary conditions and then they have normal operations. So, when I test a loop maybe I want to test around its boundary condition. And let us say the loop is meant to execute 100 times, it is a far loop for I is equal to 1 to 100. So, I do not want to really execute 100 normal executions of the loop.

So, I want to be able to test the loop for its normal execution maybe once. And then I want to be able to test the loop around its boundary conditions. What happens when the loop begins, what happens when the loop ends? So, when we say we need a set off test cases that cover a loop we are looking for the following kind of test cases. When we execute the loop at its boundary conditions which will involves skipping the loop also and then we execute the loop maybe once or few number of times for its normal operations.

(Refer Slide Time: 24:36)



Prime Paths in Graphs

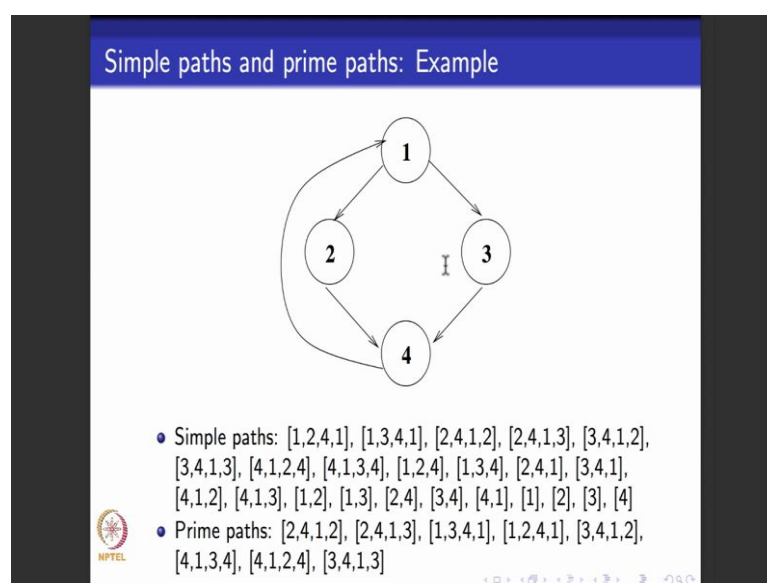
- A path from node n_i to n_j is **simple** if no node appears more than once, except possibly the first and last node.
 - No internal loops.
 - A loop is a simple path.
- A **prime path** is a simple path that does not appear as a proper subpath of any other simple path.

NPTEL

So, the notion of prime paths and prime path coverage help you to achieve this kind of execution for loops. So, what are prime paths? Before we look at prime paths I need to tell you what a simple path is. So, what is the simple path? A path from a particular node n_i to another node n_j is said to be simple, if no node appears more than once except possibly the first and the last node. So, simple path could be cycles, they could begin and end in the same node n_i and n_j could be the same vertex, but in between the path nothing appears more than once; which means there are no internal loops in the graph and every loop is believed to be a simple path.

Now, moving on ,what is a prime path? A prime path is a simple path that does not appear as a sub path of any other simple path. Just to repeat; what is a prime path? It is a simple path; and what is a second condition, second condition says that it is a simple path that does not come as a sub path of any other path.

(Refer Slide Time: 25:36)



So, I will show you an example to make this definition clear. Let us look at this graph, it has four nodes: 1, 2, 3 4. How many simple paths are there? If you go and see this listing what I have done is I have listed all the simple paths, I have listed them in a particular order in fact maybe will begin that this end. This 1, 2, 3 4 if you see right at the end of the listed simple paths they are simple paths of length 1.

Later we will move ahead and listed simple paths of lengths 2 which are basically; sorry 1, 2, 3 4 as simple paths of length 0 they just contains single vertices. Then I have gone ahead and listed simple paths of length 1 which as just edges. Then here I have listed simple paths of length 2 which are edge pairs like way, and the then I have listed simple paths of length 3.

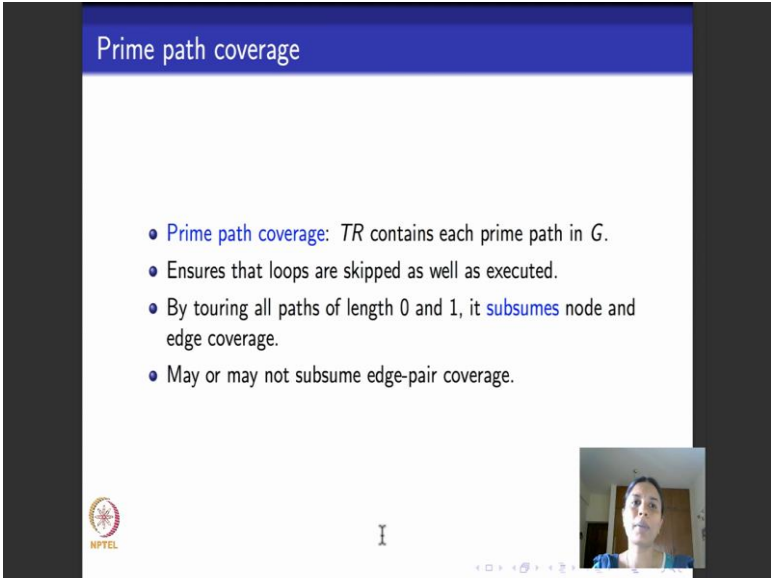
The other thing to note that this is the graph with four vertices. So, simple path means no vertex should be repeated. So, if I have a condition then what is the maximal lengths simple path that I can have with four vertices? The maximal length simple path that I can have with four vertices should be of length 3, because the moment I increase one more edge I am repeating one vertex.

So, the maximum length that I can get without repeating a vertex with four vertices is of length 3. Now if you see, if you look at all these simple paths, there so many of them, I say out of all these simple paths only these a prime paths. Why do I say these are prime paths? Go back to that definition of prime- path prime path should be simple, prime path

should not be a sub path of any other path. So, if you take a path that looks like this let us take 4, 1, 2 this path; 4, 1, 2. If I take a path like that this 4, 1, 2 path comes as a sub path of this path here 2, 4, 1, 2 it also comes as a sub path of this path here 3, 4, 1, 2. So, 4, 1, 2 does not qualified to be a prime path.

Similarly, if I take something like 2, 4; 2, 4 occurs a sub path here 1, 2, 4, it occurs a sub path here 2, 4, 1 it occurs as a sub path here, here, here, several places. So, path like 2, 4 does not qualified to be a prime path. If I go on looking like this, I basically eliminate all paths of length 0, 1, 2 and 3. And only paths of length four in this case happened to be prime paths, because none of these paths of length four occur as sub paths of each other. So, this graph has several simple paths, but it has only about eight prime paths.

(Refer Slide Time: 28:09)



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.

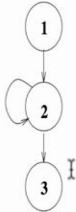
Now, what is prime path coverage? Prime path coverage very simple; the test requirement for prime path coverage says- please cover all the prime paths in G . Now, to be able to meet this test requirement you need to write a set of test paths that first identify the prime paths in C and then cover them. So, in the next lecture I will tell you how to do this, what are the algorithms that will do this, but for now we will go ahead and see a little bit more about prime paths coverage without worrying about how to do it.

So, the first thing that I want to tell you is that prime path coverage actually meets this loop coverage criteria that I told you about; this one test a loop at its boundary and test a loop for its normal operations. So, we will understand how that happens. Prime path


coverage ensures that loops are skipped and executed. And towards all paths of length 0 and 1 we saw that for that example at least. So, by default it subsumes node and edge coverage, because if you see here it includes node coverage and it includes edge coverage.

(Refer Slide Time: 29:23)

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 $\{1, 2, 3\}, [2, 2]\}$.



This node says that it may or may not subsume edge pair coverage. Why is that so? I will show you an example: considered a graph that looks like this, it has three vertices and then it has got a self loop here. So, edge pair coverage for this graph requires that the self loop needs to be visited. So, edge pair coverage for this graph we will say you visit this edge 1, 2, 2; you visit this path of lengths 2 which is 1 to 2 and 2 to itself. But if you look at this path 1 to 2; if you see the vertex 2 repeats here. And that violates the condition of it being a prime path. Remember prime path a simple paths no intermediate vertices as supposed to repeat then non supposed to have loops.

So, except for this problem prime path coverage does cover edge pair coverage, but not for graphs like this; if a graph looks like this right prime path coverage does cover edge pair coverage. But if a graph has a self loop then prime path coverage does not cover edge pair coverage.

(Refer Slide Time: 30:19)

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.

NPTEL

So, now we will move on and understand how prime paths cover the notation of a loop. So, we will go back to this example graph that we looked at a few slides earlier. In this example there is this loop here between nodes 5 and 6. And I am not listing the prime paths for this graph, but you can take it on faith that this graph has nine prime paths. In the next module I will tell you how to list all those nine prime paths.

So, then if we see this is one prime path; 1, 3, 5, 7 because it is a simple path and it does not occur as a sub path of any other path. So, this prime path for this particular example corresponds to skipping this loop, because it completely avoids this loop and if you see here is another prime path 1, 3, 5, 6. This prime path gets into the loop, it corresponds to executing the loop once.

And then this is another prime path for this example 6, 5, 6. Why is this prime path? Remember in prime paths which have simple path the beginning and ending nodes are allowed to repeat, only the intermediate nodes are not allowed to repeat. So, 6, 5, 6 is a prime path. And, what is the main job then 6, 5, 6 is doing for this graph? It is executing the loop more than once. So, if we see these three prime paths 1, 3, 5, 7 skips the loop 1, 3, 5, 6 enters the loop and then 6, 5, 6 helps you to execute the loop for its normal operations. So, it is intuitively in this sense that prime paths exactly capture loop coverage the way we want them to do in test case design.

(Refer Slide Time: 32:04)

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 to have internal loops to make prime path coverage feasible.

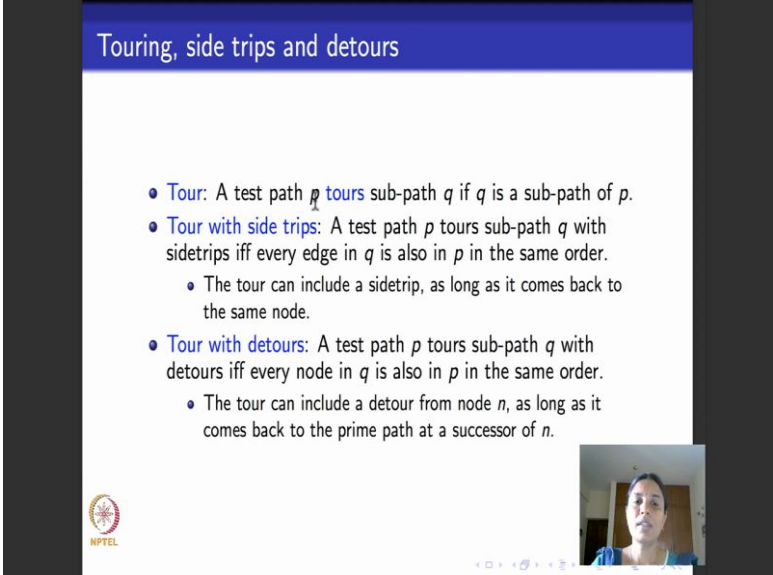
NPTEL

So, one important thing is that, I told you right, in the next module I tell you how to come up with an algorithm that will help us to design test cases that will need the test requirement of prime path coverage. But before we do that I will tell you we could have some problems. Like for example, you take a graph like this. Prime path coverage for this graph, this is a prime path 1, 2, 3, 4, 5, this is a prime path. It might so happened this graph if this graph corresponds to the control flow graphs is some piece of code here is a loop. It might so happen that this code from which this control flow graph is derived, the loop present at statement number 3 and 6 is such that you can never write you have to execute the loop at least once.

Like for example, if I take a C program. There are two kinds of loops: there is a do-while and a while-do loop right. While-do loop first checks the condition and then executes the loop, but if I have a do-while loop I have to execute the loop at least once before I move on. So, it might be an infeasible test requirement for that kind of a code when I say that you achieve this prime path without executing the loop. Why should I not execute the loop? Because if I execute the loop then the vertex three which is an intermediate vertex in this path will occur more than once. So, it will not be a prime path. So, 1, 2, 3, 6, 3, 4, 5 is not a prime path because 3 occurs more than once; 1, 2, 3, 4, 5 is a prime path. But to be able to write a test path and execute it in the code to achieve 1, 2, 3, 4, 5 the code might be such that I might have to go through this loop once. Like I told you write the code might have a do while statement.

So, how do we get over this? We get over this by using a notion of a side trip and a detour. So, we will see what they are.

(Refer Slide Time: 34:09)

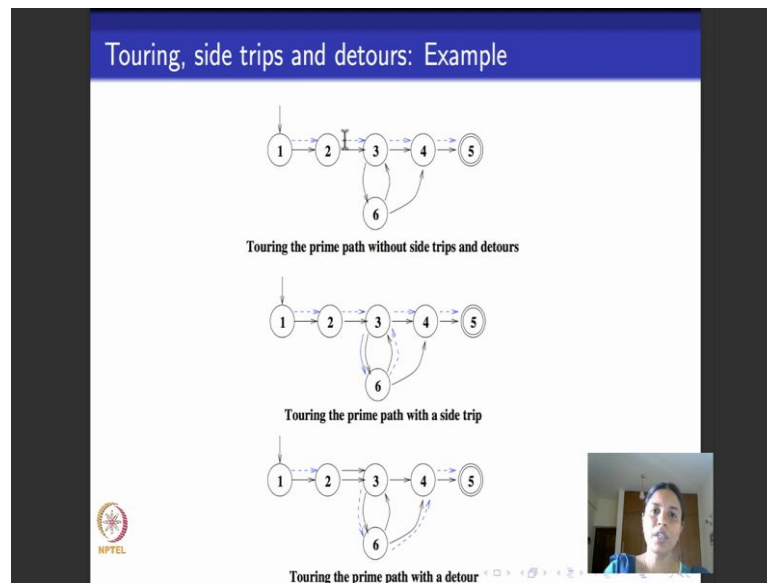


The slide is titled "Touring, side trips and detours" in a blue header. It contains three bullet points defining different types of tours. In the bottom right corner, there is a small inset video of a woman speaking. The NPTEL logo is in the bottom left corner.

- **Tour:** A test path p **tours** sub-path q if q is a sub-path of p .
- **Tour with side trips:** A test path p tours sub-path q with sidetrips iff every edge in q is also in p in the same order.
 - The tour can include a sidetrip, as long as it comes back to the same node.
- **Tour with detours:** A test path p tours sub-path q with detours iff every node in q is also in p in the same order.
 - The tour can include a detour from node n , as long as it comes back to the prime path at a successor of n .

So, what is a tour? Tour I introduced you in the last module when we looked at graphs tour is a test path that tours is a sub path if q is a sub path of the overall test path. So, what is a tour with a side trip? A tour with a side trip is the following--- test path p towards the sub path q with side trips, if every edge of q is also in p in the same order. It is like taking a side trip as a part of a main holiday. And what is a tour with the detour? A tour with a detour is the test path p towards a sub path q with the detour if edges do not come in the same order, but vertices come in the same order.

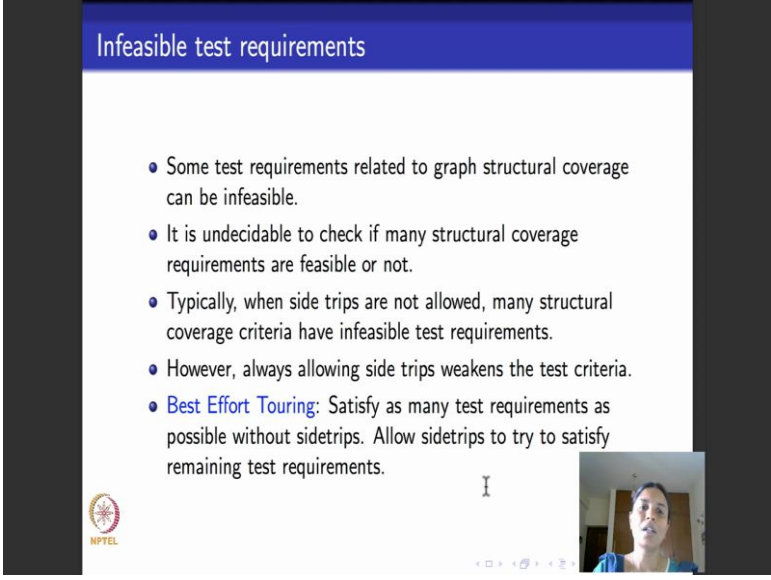
(Refer Slide Time: 34:52)



So, I will show you an example. So, here is my test path 1, 2, 3, 4, 5. I take the same test path to meet the prime path coverage of 1, 2, 3, 4, 5 with the side trip. So, what do I do in the side trip? Side trip says that it is a same path 1, 2, 3, 4, 5, but I do 1, 2, 3 take a side trip around 6 come back to 3 and do 4, 5. So, I retain the vertices that come in the main path. See, side trips says that you retain vertices that come in the same path in the same order; retain the vertices 1, 2, 3, 6, 3, 4, 5, but in the main path which is this blue dotted line the vertices occur in the same module.

Now what is a detour? Detour says you retain edges in the same order, may or may not retained vertices in the same order. So, a detour looks like this 1, 2, 3, 6, 4, 5. It has left this edge it does not retained edges in the same order, but it retains this path. So, why do I need this? I need this because to be able to achieve this prime path coverage I might have to go through the loop. And if I go through the loop the prime paths ceases to be at prime path. So, I do a small work around then say I actually cover this prime path, but with the help of a side trip. Sometimes I might say I cover this prime path, but with the help of a detour. They have just extra additions that we add so as to not to alter the notion of prime paths being simple paths, but I still want to be able to achieve these test requirements.

(Refer Slide Time: 36:30)



The slide is titled "Infeasible test requirements" in a blue header. It contains a list of five bullet points. The first four are in blue, and the fifth is in black. A small inset video of a woman speaking is located in the bottom right corner of the slide area. The NPTEL logo is in the bottom left corner.

- Some test requirements related to graph structural coverage can be infeasible.
- It is undecidable to check if many structural coverage requirements are feasible or not.
- Typically, when side trips are not allowed, many structural coverage criteria have infeasible test requirements.
- However, always allowing side trips weakens the test criteria.
- **Best Effort Touring:** Satisfy as many test requirements as possible without sidetrips. Allow sidetrips to try to satisfy remaining test requirements.

So, this is what I was telling you about. Some test requirements related to graph coverage criteria may be infeasible. In fact, it is undecidable to check whether even test requirement is feasible or not, we won't really go onto its details. But typically when I allow side trips, then I might be able to achieve test requirements with reference to a particular code.

So, what is called a best effort touring is that you tried to satisfy as many test requirements is possible without side trips or detours, and then if you still have test requirements that are unachievable, consider using side trips and detours to be able to satisfy them.

(Refer Slide Time: 37:09)

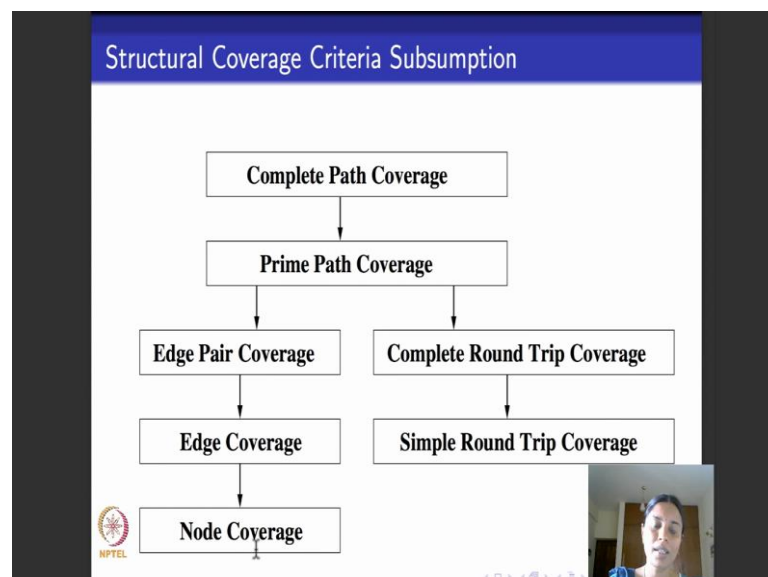
Round trips

- **Round trip path:** A prime path that starts and ends at the same node.
- **Simple round trip coverage:** TR contains at least one round trip path for each reachable node in G that begins and ends in a round trip path.
- **Complete round trip coverage:** TR contains all round trip paths for each reachable node in G .
- The above two criteria omit nodes and edges that are not in round trips.
- Hence, they do not subsume edge-pair, edge or node coverage.

NPTEL

So, what is a round trip? A round trip is a prime path that starts and ends in the same node, we have seen these, these suggest specific kinds of prime path. What is a simple round trip coverage? Test requirement contains one round trip path for each reachable node that begins and ends in the same vertex. What is complete round trip coverage? Test requirement contains all the round trip paths. So, they subsume edge and node; I mean they do not subsume edge pair edge or node coverage.

(Refer Slide Time: 37:42)



Now to summarize what are the various coverage criteria we saw; we saw node coverage, visit every node; edge coverage, visit every edge; edge pair coverage visit every path of length at most two, visit prime paths means compute on the prime paths right test paths that exactly visit every prime paths. We will see how to do this in the next module. Then, complete path coverage which means visit every path in the graph which is, I believe, a useless test requirement. And then we had these two which talk about round trips. They are basically prime paths, but begin and end with the same node.

And why have I put them in the structure? This structure captures how each of them subsume the other. We discussed this right, edge coverage subsumes node coverage. Edge pair coverage subsumes both edge coverage and node coverage. Prime path coverage, except for graphs with self loops, subsume edge pair coverage, edge coverage and node coverage. Complete path coverage subsumes everything, but is infeasible and useless.

So, what we will see in the next module is how do we look at algorithms that given each of these structural coverage test requirements, how do I come up with the test cases? Edge, edge pair and node coverage a easy to do, but prime path coverage is a nice algorithm. So, we will spend most of the next module looking at these algorithms.

Thank you.