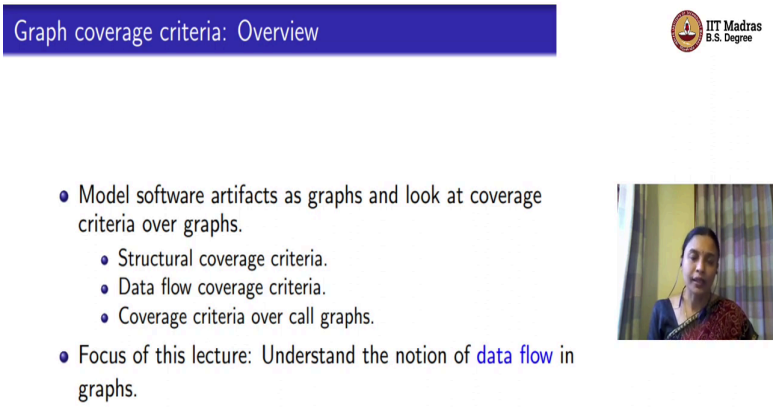
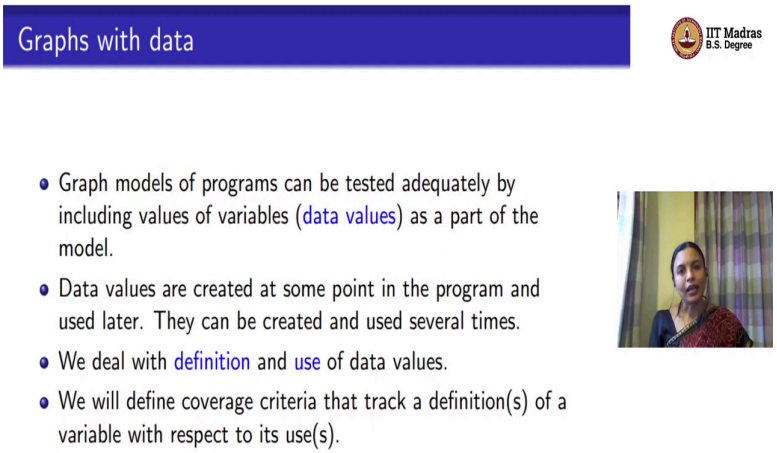
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science Engineering
Indian Institute of Technology, Bangalore
Data Flow in Graphs

Welcome back to week 3, we will continue with graph coverage criteria. I am going to tell you about adding data information to the control flow graphs to get data flow-related information and testing related to data flow in graphs. We will first see what kind of data flow can be represented in graphs.

(Refer Slide Time: 0:38)





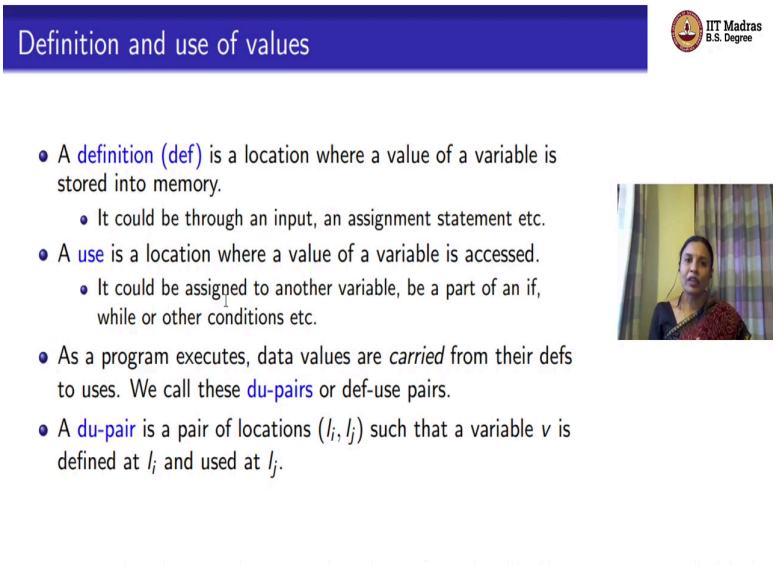
So till now we have seen structuring coverage criteria. You saw node edge, edge pair, prime path coverage, and so on. And then how to write test cases for prime path coverage. Now,

what we are going to do is take the same graph, control flow graph, but add data flow information to that and define in the next lecture some data have low coverage criteria. That is the plan. So what is data flow? How do I add graphs with data?

So you know that control flow graphs, as models or programs, methods, or functions can be tested by on their own. And they can also be tested by adding values of variables to them. So the values of the variables that we add, are called data values, that belong to the part of the model. So we know that every procedure function, every program has some set of variables, those variables get defined somewhere and then later, there are statements in the program that use the value stored in these variables.

So put together, these variables constitute the data and the data flow, we will talk about the places where the variables are defined, and the places where the variables are used. So data that gets defined at some point, will have to get used later in the code, maybe once, maybe more than once. And we work with coverage criteria that will track how data gets defined. And the data once it gets defined, how does it get used?

(Refer Slide Time: 2:20)



So what is the definition of a value or a variable? So let us take a variable, let us not worry about what that variable is for now, it could be an integer type, it could be a string, it could be a Boolean type, it could be a file, it could be an array, but let us say there is a variable. At some point it gets defined in the program. When it gets defined in the program, it could get declared, it could get declared and initialized.

Whatever happens at that statement, a memory location for that variable is created and a value is stored for that variable into the memory location. So, these places are what we call definitions abbreviated as short as depths of a variable. The place or statement where a memory location for a variable is created, and a value for that variable is stored into that memory location. That is what we call it as the definition or def of a variable.

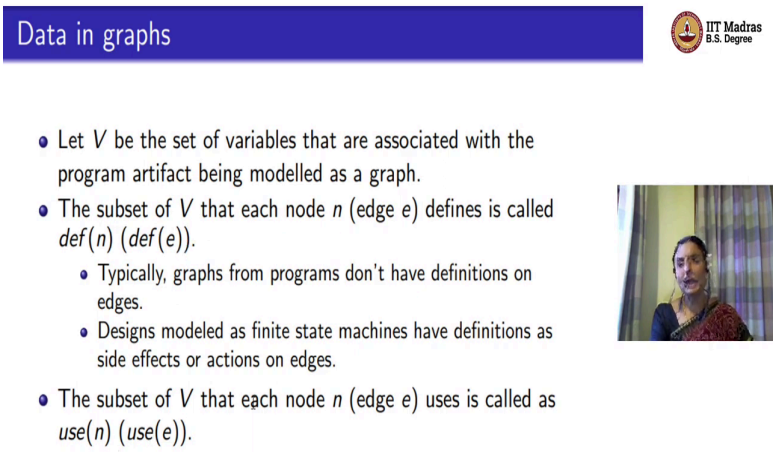
Use of a variable is a location where the value that is stored or the variable in the memory location is accessed, it could be through an assignment statement, where this variable's value is assigned to another variable, this variable is used as part of an expression or it could be the case that this variable is used as a part of a decision statement like if and so on. So as a program or a function executes, variables get defined, and the variables that get defined get carried from that definition to their uses.

So, these data values that get carried from the definition to the uses are what we call definition use pair def use pair or du-pairs, d short form for definition, u short form for use, so def use shortened as du-pair. So what is the du-pair? A du-pair is a pair of locations, Li and lj. li and lj are two different statements in the program such that for a variable v, the variable v is defined at li and used at lj. Is it clear what it is?

A definition of, just to repeat, a definition of a variable is any statement where the value of the variable is stored into memory. It could be created for the first time, it could be reassigned anytime, anything can happen. A use of a variable is any statement or location where the value of the variable is retrieved from memory. It could be a part of an assignment statement, where its value is assigned as a part of an expression to another variable, it could be a part of a decision statement, and so on.

As the program executes, every variable's value is carried from its definition to its use, we call these pairs as du-pair. So for a variable v, a du-pair is a pair of locations such that the variable is defined at location li and used at location lj.

(Refer Slide Time: 5:30)

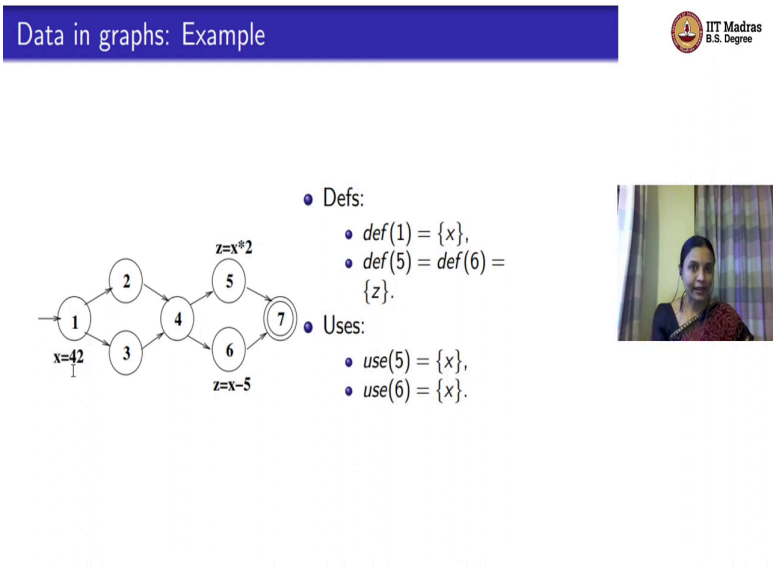


So, some notations that we will use. So let us say we have a particular program and let capital V be the set of all variables that are part of that program. Then, every statement in the program, or every basic block of statements in the program corresponds to one node in the control flow graph. And a subset of variables get defined at that node. So we call it as the def of that node, def of n is the subset of variables that get defined at that node.

Similarly, for every edge, a subset of the set of variables can get used to that edge. Like for example, the edge could represent a branch out of an if statement. And maybe two of those variables are used for decision and the if statement. So, we call that subset as the definition associated with the edge E. Typically, graphs from programs do not have definitions on the edges because edges are decision statements represent transfer of control from one statement to the other.

So, graphs that come from programs control, flow graphs have definitions mainly only on nodes, they have uses on edges as through decision statements, condition checks, and so on. There are other models where there could be definitions on edges like finite state machines, which we will see later. So, the subset of the set of variables that each node n uses is called use of n. Similarly, the subset of the set of variables that each edge uses is, here is a small example.

(Refer Slide Time: 7:05)



So, let us take this small graph, let us assume that it represents some control flow graph, it has 7 nodes. Node 1 is the initial node, node 7 as its marked my double circle, the final node. I have marked some of the nodes with some statements. Node 1 actually happens to be representing the statement exercise 42. Read this diagram like this. Similarly, node 5 happens to be the assignment statement corresponding to z is assigned x into 2.

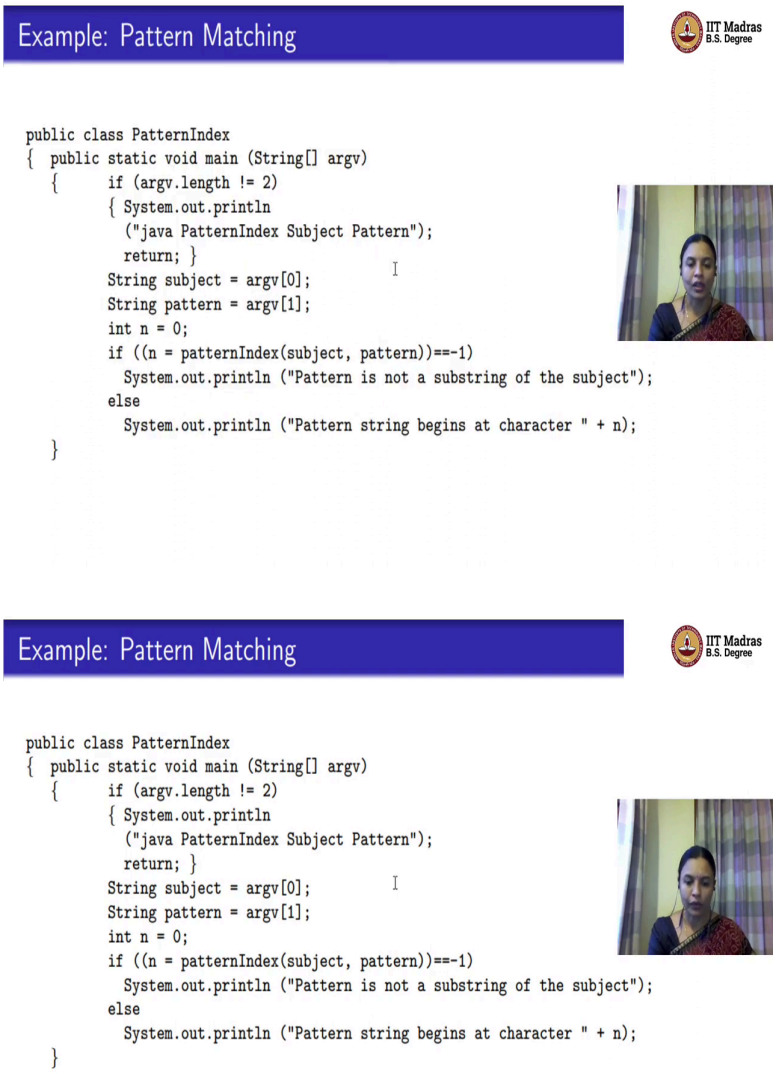
Similarly, node 6 happens to be the assignment statement z is assigned x minus 5. So there could be other things which I have not marked in this graph, like for example, we do not know what node 2, node 3, node 4 and node 7 represent. We do not know but with the information that we know that node 1 represents the statement x is equal to 42, node 5 represents z is equal to 2x, node 6 represents z is equal to x minus y. We are ready to write some information about definitions and uses. Now, what are all the variables that you see in this diagram?

In this graph, there is a variable x, there is a variable z and in the node 1, x is assigned 42. That is, the memory location for x is accessed is updated with the value 42. So I could say definition of 1 if singleton x is drifting. Similarly, in the nodes 5, and in the nodes 6, the variable z gets defined as 2x in 5 as x minus 5 and 6. So definition of 5 is z, definition of 6 is also z. Is that clear?

Similarly, now let us go to uses. To define z in nodes 5 and 6, I use the variable x. In node 5, z is defined as 2x. So use of 5 is also x. Similarly, in node 6, z is defined as x minus 5. So x is

also used at 6. So I take a graph like this, I look at some statement, and I can mark out the definitions and uses. We will see more examples as we move on.

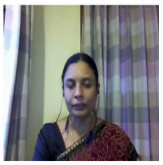
(Refer Slide Time: 9:29)



Pattern Matching Example contd.

IIT Madras B.S. Degree

```
while (isPat == false && iSub + patternLen - 1 < subjectLen)
{  if (subject.charAt(iSub) == pattern.charAt(0))
   {  rtnIndex = iSub; // Starting at zero
      isPat = true;
      for (int iPat = 1; iPat < patternLen; iPat++)
          { if(subject.charAt(iSub+iPat) != pattern.charAt(iPat))
            { rtnIndex = NOTFOUND;
            isPat = false; }
            break; // out of for loop
          }
      }
   iSub ++;
   }
   return (rtnIndex);
```

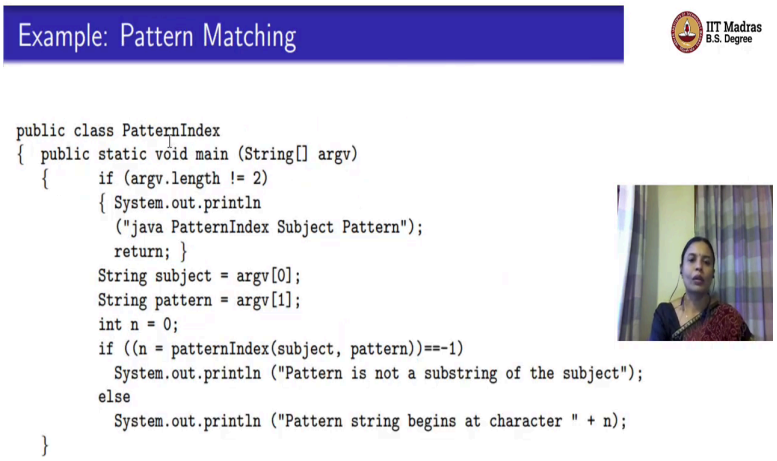


So the next example that we are going to see, remember last time, we saw the statistics program, which computed statistics parameters. This is another example that we will see. This is a pattern matching program that looks for a pattern in a string. You must have used these things. Suppose you use an editor let us say Microsoft Word and you want to search for the occurrence of a particular string, then you type for that string, type Ctrl F, type for that string and then it will highlight all the places in the file where that pattern occurs.

There is an underlying pattern matching algorithm. So a pattern matching algorithm basically takes a subject, a larger string, and a pattern, a smaller string and checks if the pattern occurs in the subject or not. If it occurs in the subject, then it might give you the first occurrence of the pattern in the subject or the last occurrence of the pattern in the subject or all the occurrences of the pattern in the subject.

If it does not occur in the subject, it might tell you that it does not occur in the subject. So this is a Java program that does pattern matching. For ease of readability, I have given the program over three slides. So the slide 1 is the first part of the program, slide 2 is the second part, continue, read it as one continued program in slide 2, which continues and ends in slide 3.

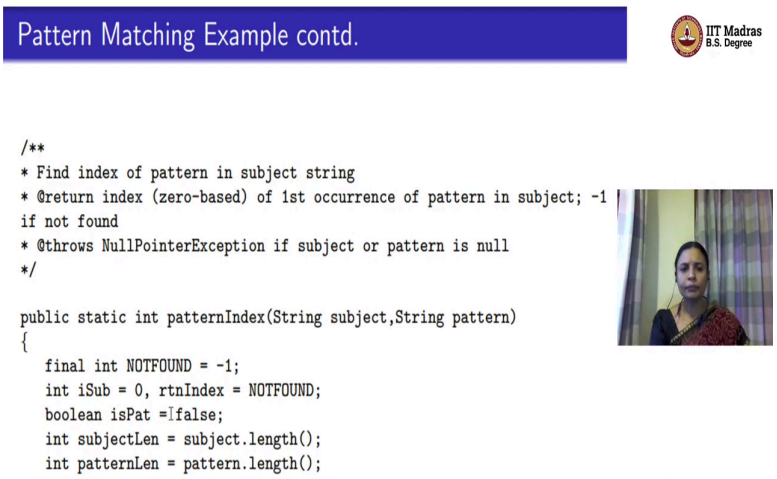
(Refer Slide Time: 11:00)



So let us read and understand what the program says. Let me go back to the beginning of the program. So it says public class patternindex, then it takes a string. And now it asks for input two things, one is it takes input strings on subject as input, and another string for pattern as input. So basic job is to look for whether this string pattern occurs in the string subject or not. So it uses a counter n and it, initializes it to 0, and then starts.

So it uses this patternindex. And if it says if it returns minus 1, then you say the pattern is not present in the subject. Otherwise, it will return n, which is the index, first place where the pattern is present in this app. So let us continue.

(Refer Slide Time: 12:03)

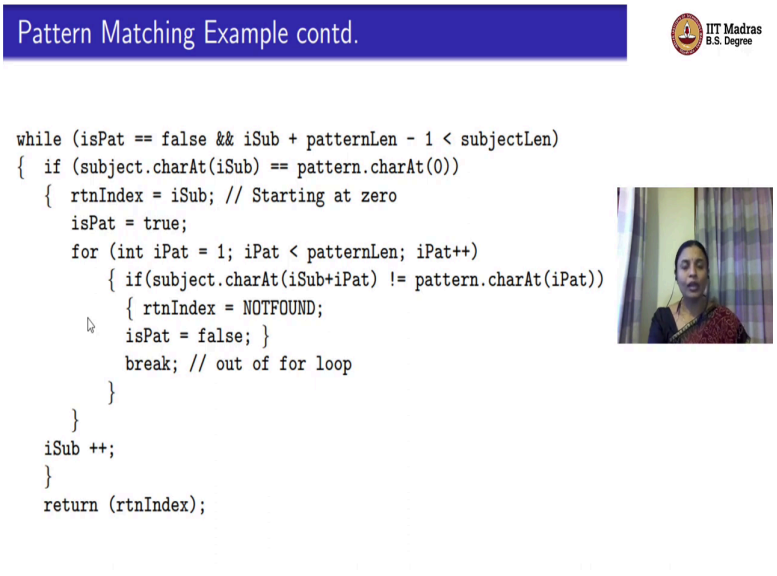


So this is the same program running into the second slide, please remember that. So it finds the index of the pattern in the subject string, and it returns the first occurrence of the pattern in the subject if it is found, otherwise, it returns minus 1.

If there are any issues like both subjects, or pattern, or one of them is null, it will throw a null pointer exception, the code for that is not explicitly given here. So this is the method. The method is called patternindex. It takes two arguments- a subject and a pattern, both of them are strings, and then it starts its work. It initializes a whole set of variables. There is something called not found, then it initializes iSub, an index to vary over subject, then it initializes something called rtnindex, short form for return index.

Then it uses a flag called ispattern. So the Boolean flag which is set to false. Then it computes the length of the subject and the length of the pattern, and then it moves on.

(Refer Slide Time: 13:07)



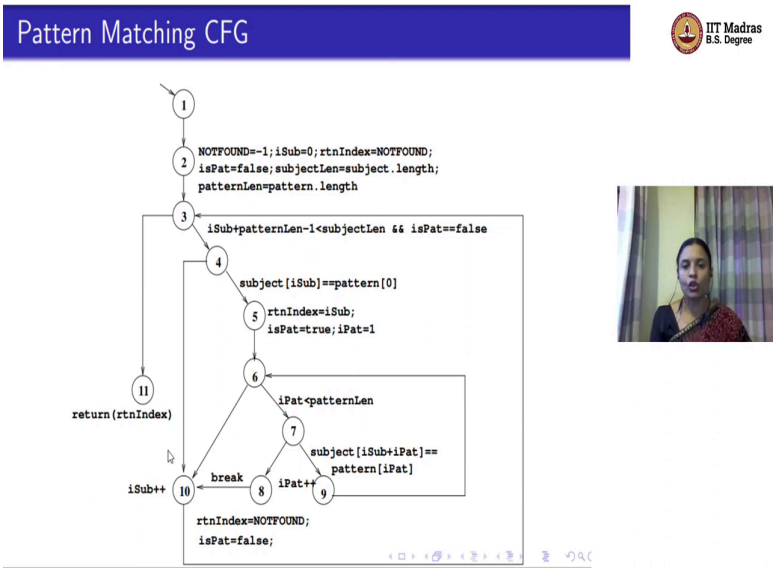
So this is a while loop. So it says while the pattern found is set to false, and the index over the subject is a plus, the length of the pattern minus 1 is less than the length of the subject, that is as I start with the first placeholder in the character of the subject, and I shift one at a time, and the shift, I keep track with reference to this variable iSub. So then it checks.

Look at the character at this particular index, iSub in the subject, if it matches the first character in the pattern that is at index 0, then you say, okay, return index is iSub, I found the pattern may be, so set it to true, and then check if the subsequent characters of the pattern are found in the same order in the subject. So the back check is done by this for loop.

So it says for in the index of a pattern, set it to 1, as long as the index of the pattern is less than length of the pattern, check if the subject character found at this position iSub plus iPat is the same as the pattern character at the index of the pattern. If it is not, then you say sorry, I have not found patternindex, and you break out the formula. If it is, then you increment iSub and keep repeating this, keep repeating. It is a standard pattern matching program, no big surprises.

If you are familiar with how to look for a pattern in a subject, you could do the same. So why did we look at this program? We would like to use this as an example, draw its control flow graph and try to look at some data flow information. So just to reiterate, I have given you a pattern matching example, Java program over three slides, you can also find it in the textbook, the reference for the textbook is given in the last slide.

(Refer Slide Time: 15:12)



Pattern Matching Example contd.

IIT Madras B.S. Degree

```
/**
* Find index of pattern in subject string
* @return index (zero-based) of 1st occurrence of pattern in subject; -1
if not found
* @throws NullPointerException if subject or pattern is null
*/

public static int patternIndex(String subject,String pattern)
{
   final int NOTFOUND = -1;
   int iSub = 0, rtnIndex = NOTFOUND;
   boolean isPat = false;
   int subjectLen = subject.length();
   int patternLen = pattern.length();
```

Now, a small exercise, I urge you all to draw the control flow graph for this one. So if you draw the control flow graph using what we had learned earlier in the course, then your control flow graph will look like this. I have drawn it with these boxes, big boxes, long-long lines for edges just to improve readability. So, the control flow graph for the pattern matching example, is going to be a graph that looks like this, maybe the one you draw will not have the exact same outlay graphs can be laid out differently, but it is roughly going to have 11-12 nodes, there is going to be an initial node.

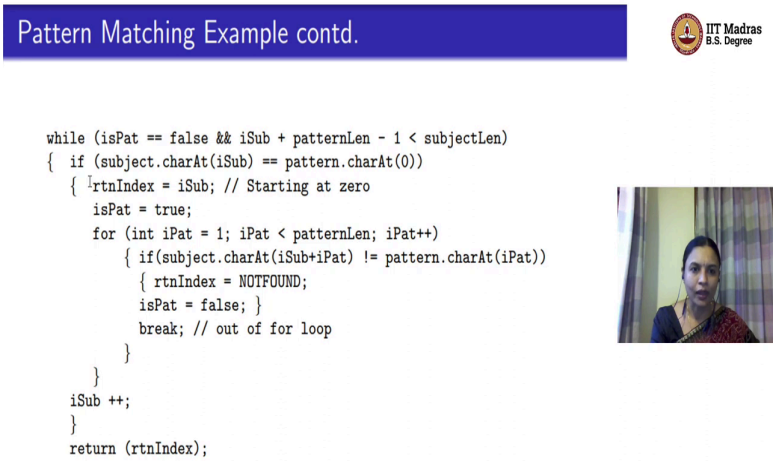
And then this node 2 represents all these statements. Let me just go up a little bit back and forth. Node 2 represents these statements, these initial initialization. So that is node 2 in the graph here. After that, code comes where? Here. It starts with this while statement, there is an if statement, there is a for statement, there is another if inside that, just remember the structure. That is exactly what the CFG is going to capture.

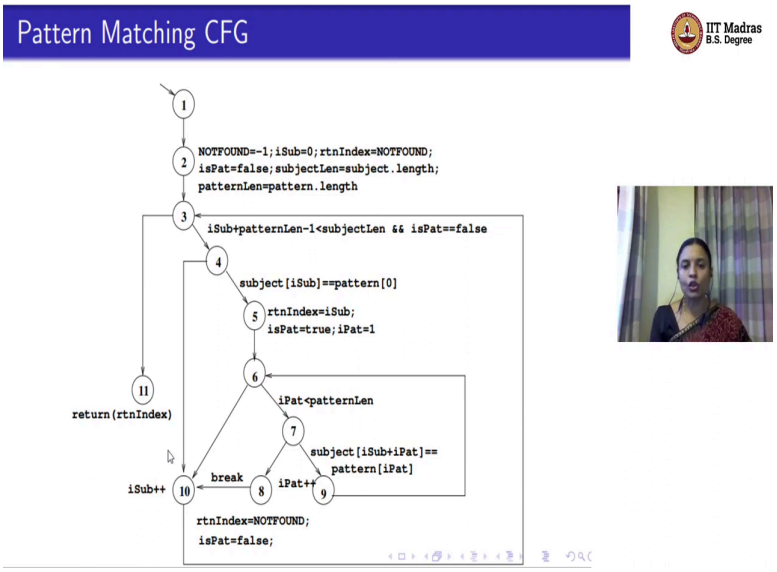
Node 3 represents the while statement, 3 to 11 is the edge where the while fails, 3 to 4 is you enter the while loop, 4 is the if statement, then 679 679 is the for loop, and 7 to 8 to 9 is the if inside the for loop, and various exits are exits out of the respective ifs and the respective fors. So for ease of traceability, what I have done here is I have marked the reset of statements, the label each of the nodes and some of the edges.

For example, node 2 represents a basic block of all the initializations that you find, node 3 represents the while loop. So edge 3 to 4 is marked with this decision statement of the while loop. Similarly, node 4 represents the if statement. So edge 4 to 5 is marked with the decision

predicate of the if statement, same thing for node 4 to 10 also, but it will be the negation of that.

(Refer Slide Time: 17:26)





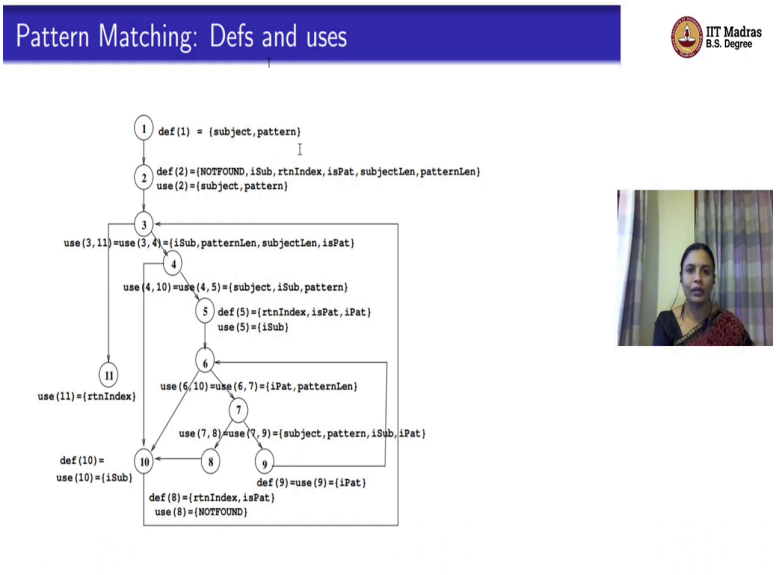
5 represents these small statements inside this if, that you find here, these ones the return index, this is sub ispat is true. These these kinds of statements are these kinds of statements, these two statements are given here, node 5. Node 6, as I told you represents the beginning of the for loop, edge 6 to 7 is labeled with the predicate of the for loop, 7 represents if loop, edge 7 to 8 represents the predicate of the if loop, and so on and so forth.

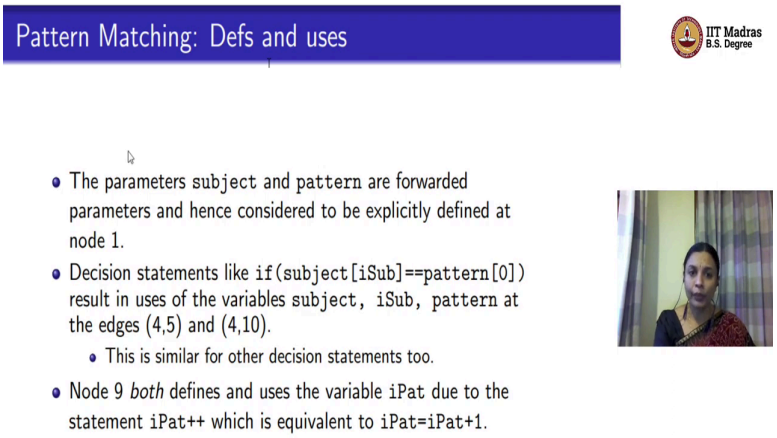
So this is the control flow graph. Why have we now labeled it, labeled some of the nodes and some of the edges with statements from the program? It is from these labels that we are going

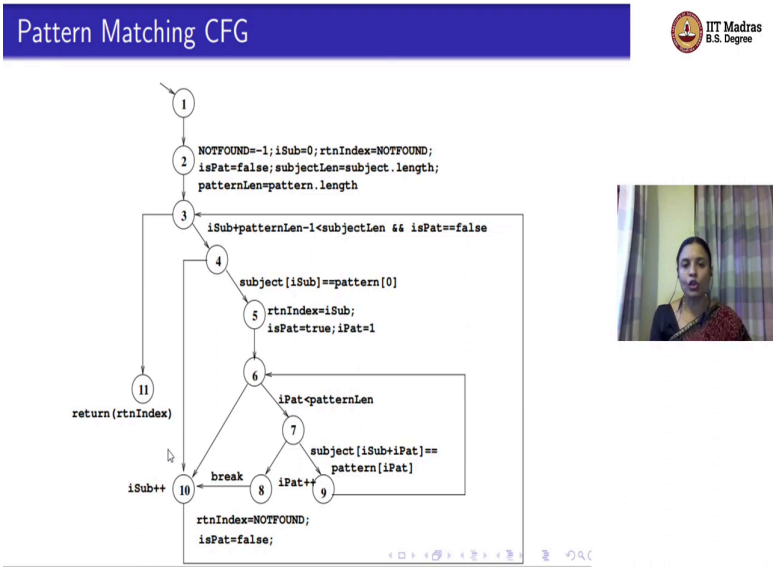
to infer the definitions and uses of the variables. So how are they going to be like? So I have just written the same thing here. So if you see, let us go back here, node 1 was empty, I will come back to it in a minute.

Node 2 had all these, it had a definition for not found as minus 1, iSub as 0, return index as not found. So there is a definition and the use are not found here, iSub is defined, isPat defined, length of the subject is defined, length of the pattern is defined.

(Refer Slide Time: 18:45)







So def of node 2 is so many things- not found, iSub, return index, isPat, length of the subject, length of the pattern. What is being used? Subject and pattern. Where are they used? They are used everywhere. They are used to find the length of the subject, they are used to find the length of the pattern. Being the definition of 1 here, if you see, it was empty, I did not decorate it with any value, but here we have put it as definition of 1 is the array subject and the array pattern. Why so? That is given here in the next sentence.

It so happens that the parameters subject and pattern are forwarded parameters and hence they are considered to be explicitly defined at node 1. So at the cost of going back, let me just explain what I mean by the thing here.

(Refer Slide Time: 19:40)

Pattern Matching Example contd.

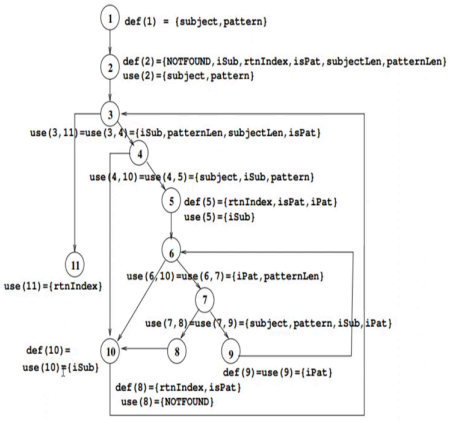
IIT Madras B.S. Degree

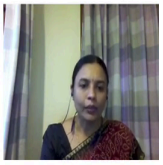
```
/**
* Find index of pattern in subject string
* @return index (zero-based) of 1st occurrence of pattern in subject; -1
if not found
* @throws NullPointerException if subject or pattern is null
*/

public static int patternIndex(String subject,String pattern)
{
   final int NOTFOUND = -1;
   int iSub = 0, rtnIndex = NOTFOUND;
   boolean isPat = false;
   int subjectLen = subject.length();
   int patternLen = pattern.length();
```

Pattern Matching: Defs and uses

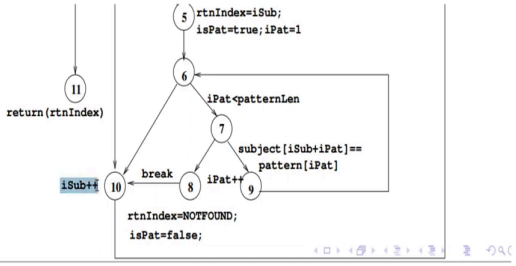
IIT Madras B.S. Degree





Pattern Matching CFG

IIT Madras B.S. Degree



So if you see here to this method, patternindex, what is being passed here in this line, patternindex? Subject is being passed, pattern is being passed. These are two strings that are being passed. That is the first node of this control flow graph, node number 1, and because they are being passed, this node represents the definition at this node is subject and pattern, that is what is written here.

Parameters subject and pattern are forwarded parameters and hence considered to be explicitly defined at node 1. Similarly, other definitions are given here, like for example, at 10 iSub is marked as iSub plus plus, that was the statement.

If you focus on what extend here, iSub is iSub plus plus, this one. So, it means that the variable item which represents the index over the subject is both defined and used at node number 10. Similarly, you can trace back and forth between the other ones.

At node 11, return index is returned, so at node 11, return index is used, use of 11 is return index. Similarly, for use of edges, if you see this edge that marks the beginning of the while loop edge 3 4, this is the predicate that is checked for being true or false. So, the use of that edge, use of 3 4 which is the same as the use of 3 11, the same predicate that is being checked it returns true then you take the edge 3 4, it returns false then you take the edge 310.

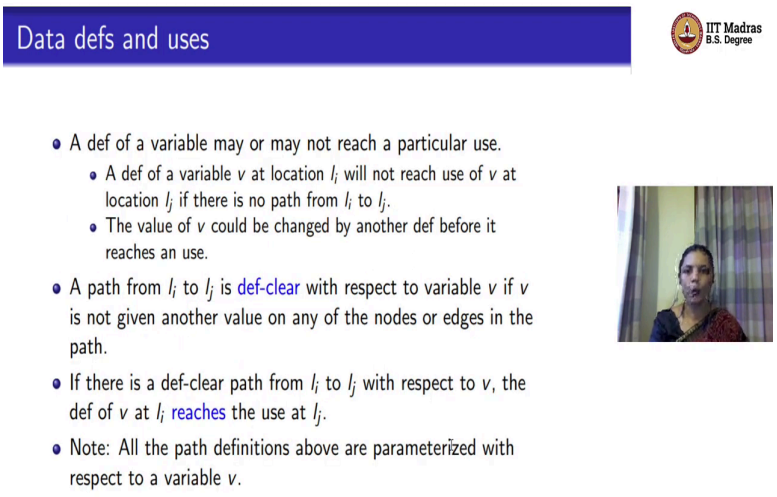
So, use of the edge 3 4 is the same as use of the edge 3 11, which are all these values- iSub, length of the pattern, length of the subject and this Boolean flag. Similarly, for this if statement, what is the predicate? If statement checks whether subject, iSub, is the same as pattern or 0. So, use of this 4 5, use of the edge 4 10 are nothing but the variables subject iSub and pattern.

Similarly, I go on marking definition associated with the nodes, users associated with nodes and users associated with edges. Not all edges will have users associated with them. Typically, whenever edges have predicates there, wherever there are decisions, they will have users associated with it. That is what is marked. Like for example, the edge 1 2 has no use because it just represents a direct transfer of control.

Similarly, the edge 2 3 has no explicit uses marked to it because that also represents direct transfer of control. Wherever they are decision statements like if, while, for, then those edges will have users marked to them. Otherwise, some nodes might have both definitions and or uses marked to them. So, I hope you can understand how to mark the definitions and uses.

So, take your method or a procedure, draw its control flow graph normally, then start studying which are the assignment statements, which are the decision statements and based on the nature of those statements mark out the definitions and uses. So, this graph is what we call the data flow graph, control flow graph augmented with definitions and uses.

(Refer Slide Time: 23:26)



So, I basically explained all these things, so, I will skip through this. Now let us look at what is a data definition what is the data use. A definition of a variable we know is a place where a value of a variable is written into memory. A definition, a variable might be defined, but maybe programmer never had a use for it. Or maybe a variable gets defined, but before it gets used somewhere else, some other assignment statement redefines it.

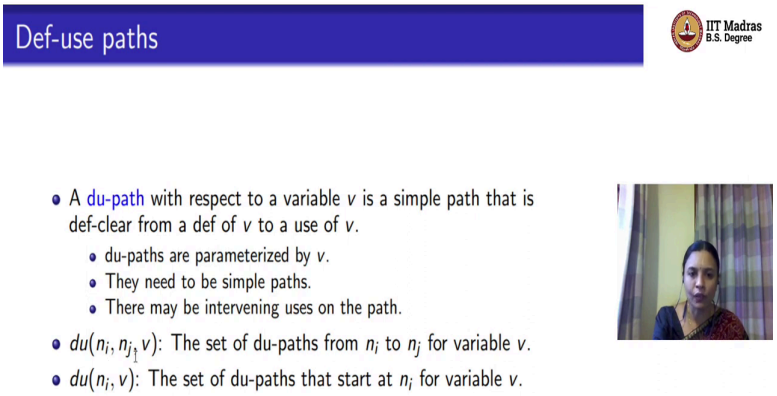
So, lots of things can happen. So, that is what is given here. A definition of a variable may or may not reach a particular use. So, if it is defined at a particular location and at a later location gets used as a part of a decision statement or as a part of another assignment statement, then we say that the definition of a particular variable say v at location i reaches the use of v at location i.

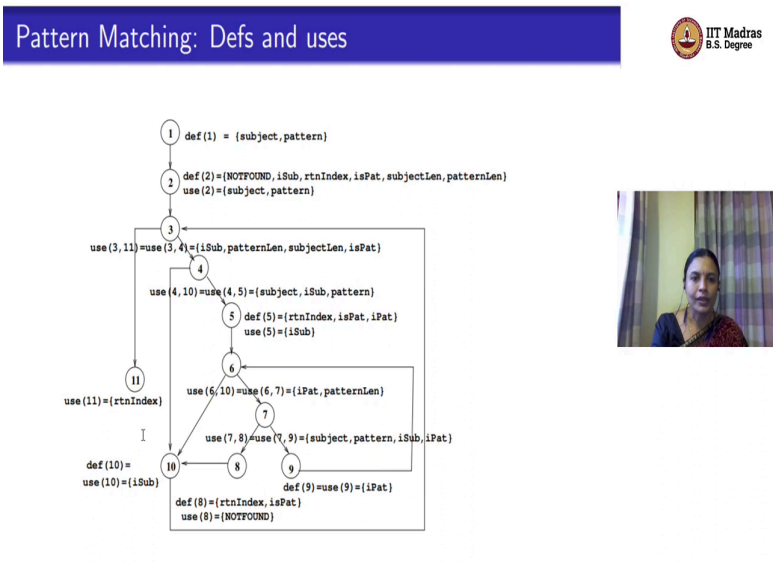
So the definition of a variable v at location i reaches the use of v at i if v is defined at li and used at lj. Otherwise we say definition of the variable v does not reach the use at lj. It does not reach meaning there could be no use or it could have been redefined. We say the path from a location li to a location lj, is definition clear with respect to a variable v, if v is not given another value on any of the nodes or edges in the path.

So, let us say or take a variable v, v is defined at a particular location li, let us say through assigned a particular value and then v is used at a particular location lj later in the program execution in between this location li and location lj, v does not get defined with a new value, does not get assigned a new value, then we say the path from li to lj for the variable v is def clear with reference to v.

If there is a def clear path from li to lj for a variable v, then we say the same thing, the definition of v at node li reaches the use and lj and these are done one variable at a time. So for the next variable, there could be two other locations and so on and so forth.

(Refer Slide Time: 26:09)





So, a du path shortform for def use path with reference to a variable v is a simple path, that is def clear from a definition of v to the use of v. du paths are parameterised by v, du paths have to be simple paths. Do you remember what a simple path is? A simple path is a path where there are no internal vertices that repeats, we saw it in the context of prime path, the same definition here.

And remember, in the very first location, it should have been defined and in the last location, it should have been used, then we say that there is a definition use path for that variable. And that path should be simple. And along the way, there should not be another definition for the value v, there could be many uses for v, but there should not be redefinition for the value v, then we say that such a path is a def use path for the variable v and we use it with this notation, du ni nj v vs. The set of all du paths for the variable v from node ni to the node n.

What is du ni v? It is the set of all paths for the variable v that start at node ni. I just marked the definition in the last case, do not explicitly identify the uses but whenever I say du ni nj v, I mark the definition and the use. That is the difference.

So for the same pattern matching example, let us go back and look at some of the definitions and uses. Just give me a minute to go back. So let us take the node 1, which are the two variables that are defined at node 1? Subject and pattern. So let us focus our attention on the variable, subject. Subject is defined at node 1.

Let us look for uses for the variable, subject. Subject is used right here in the second node itself because we use it for computing subject length, it is marked as use of two subjects. So there is a small, single edge dupath from 1 to 2 for the variable subject. Where else is the subject that is defined at one used? So let us go down- 1, then 2, 3, if you see subject is used in the edge 4 to 5 that is, it is used in the if statement corresponding to the edge 4 to 5.

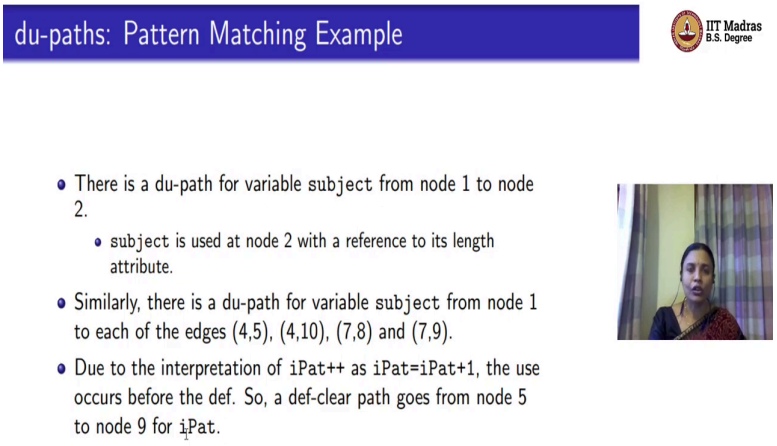
So use of the edge 4 to 5 and use of the edge 4 to 10, both are subjects. So it is used again. So this is a du path, the path from node 1 to the edge 4 5 is a du path for the variable subject. Similarly, the path from node 1 to the edge 4 10 is a du path for the variable subject. Similarly, you can do for other things also.

Like let us take isPat, if you see here isPat, is boolean flag, that is defined at note 5 and it is used at edge 6 7. So there is a du path for isPat from node 5 to the edge 6 7. It is again defined and used at node 9, isPat plus plus. So when it is defined the new set node 9 like here in this example for this iPat, similarly here at node 10, we have definition and use of iSub,

you should carefully look at the code to see whether the definition comes first or the use comes first, this is a standard programming practice.

So suppose I have a variable, let us say iSub only, suppose I write iSub plus plus or plus plus iSub, then based on what, whether the plus plus comes before the variable or after the variable, I can add one very easily. So you have to know whether the definition comes first or the use comes first. For a du path, definition should come first, not the use.

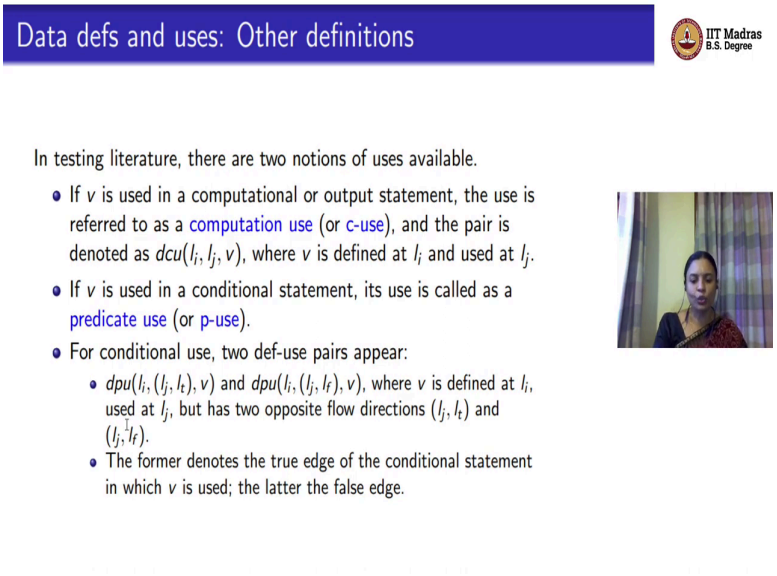
(Refer Slide Time: 30:35)



So this is what we were talking. There is a du path for the variable subject from node 1 to node 2. There is a du path for the variable subject from node 1 to all these edges. Due to interpretation of iPat plus plus, this iPat equal to iPat plus one, it is how that statement is expanded. Use occurs before the definition. So the definition clear path goes from node 5 to node 9 for the variable iPat.

So I hope this was clear what definition is, what use is, the simple things to remember. Just to summarize, definition is a place where the value of a variable is updated in memory, use is a place where a value of a variable is retrieved from memory, a du path is a simple path from a definition of a variable to the use of a variable such that there are no real definitions along the way.

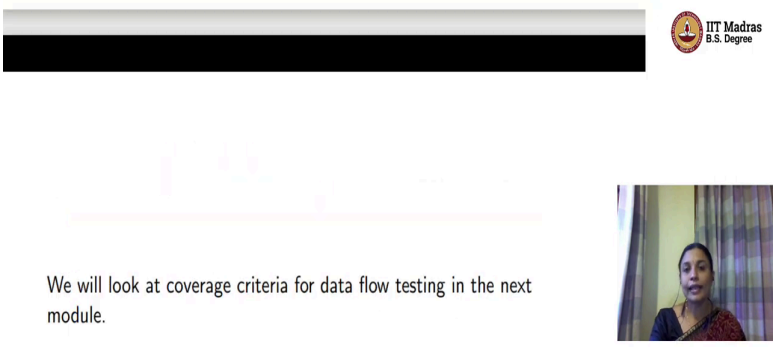
(Refer Slide Time: 31:40)



Sometimes in testing literature, or if you search on the internet, you will find various kinds of uses that are distinguished. Sometimes use is as a part of if condition or a while condition along with an edge, it comes, so we call those kinds of use as predicate use, or P use. The uses that come along with if or while statements and come along with edges.

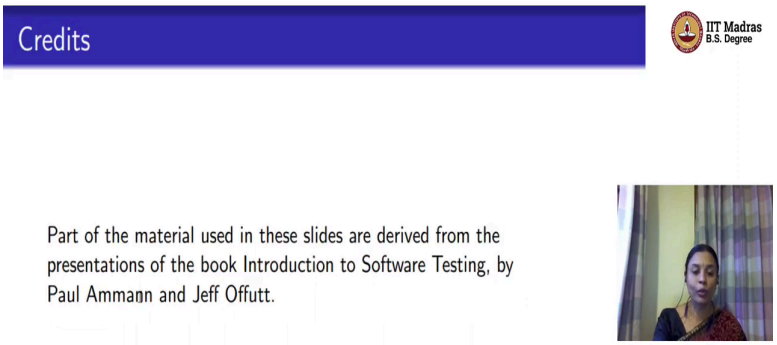
If the use is as a part of a computation, or an output statement, as a part of an assignment statement, or as a part of an output statement, then such a use, this comes along with a node in the control flow graph and it is called a C use or computation use. Sometimes they distinguish but for the purposes of our lectures, we will call both of them as uses only based on whether they come in edges, or they come in node, you could use either of them.

(Refer Slide Time: 32:32)



So in the next lecture, I will tell you on data flow coverage criteria.

(Refer Slide Time: 32:39)



And if you would like to know more about this example, as I told you look at this textbook. So I will stop here for now. We will come back with the data flow coverage criteria.