Software Testing

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

Lecture - 20
Assignment 4: Graph coverage criteria: Data flow coverage over source code, design and specifications

Hello everyone. Welcome to week 5: the goal is to finally leave graph coverage criteria and move on to other data structures and models. But before we do that the first video for week 5 would be, telling you how to solve the assignment that was uploaded for week 4 because there was a programming question and you were asked to answer some questions about def use paths on that program. So, I would like to spend this first video for week 5 trying to see whether you have solved it and to help you, if you have not been able to solve and you could also use this video to be able to crosscheck and see if your answers were right or wrong.

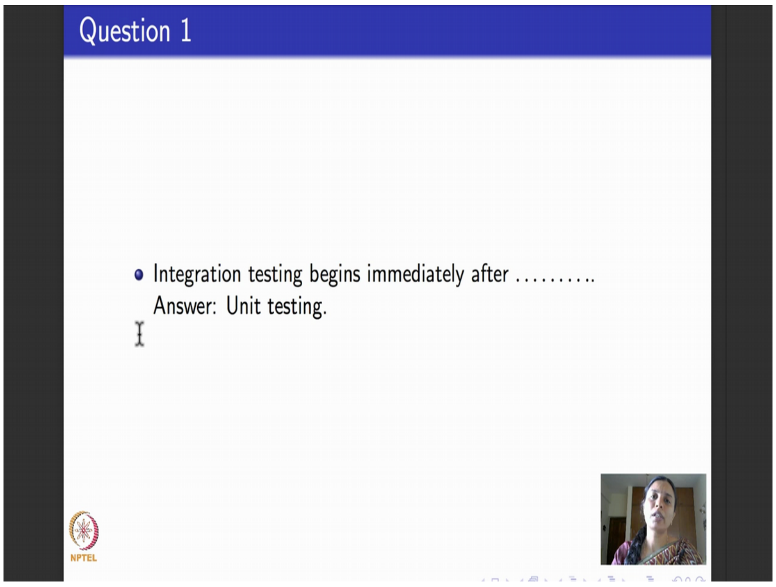
(Refer Slide Time: 00:55)

Assignment for week 2

This assignment is on design integration testing and graph coverage criteria as applied to design, sequencing constraints and finite state machines. It also includes problems on data flow coverage criteria over source code.

What was the assignment? The assignment for week 4 was; on design integration testing graph coverage criteria as it was applied to design, sequencing constraints and we also saw one problem on data flow coverage criteria.

(Refer Slide Time: 01:08)



Question 1

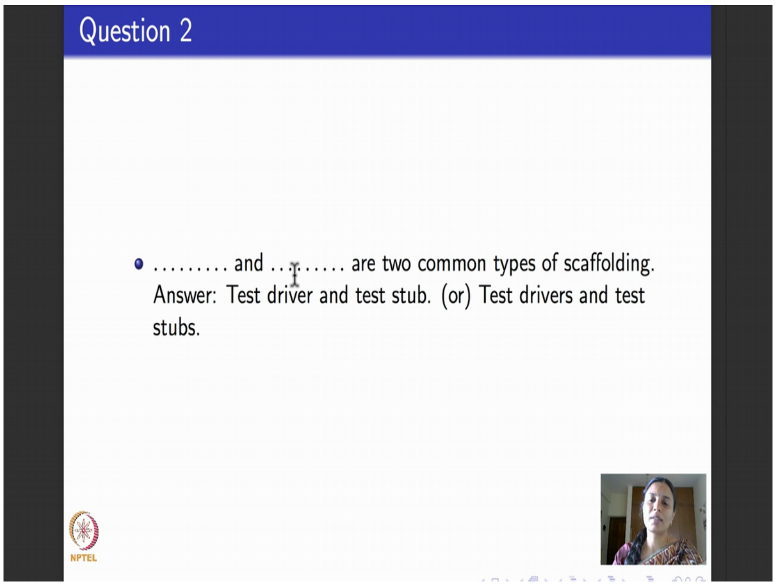
- Integration testing begins immediately after ....

  Answer: Unit testing.

So, what I will do is; I will walk you through one question after the other in the assignment and tell you what the correct answer is and what it means, what does the correct answer mean. So, the first question in that assignment was the fill in the blanks question, it asked which is the phase or stage at which integration testing sits?

That means which is the phase immediately after which integration testing comes, if you remember the v model or the waterfall model that includes testing. Integration testing comes immediately after unit testing, when the individual unit tested modules are ready. I am ready to put them together and do design integration testing, so the answer to the first question would be unit testing.

(Refer Slide Time: 01:51)



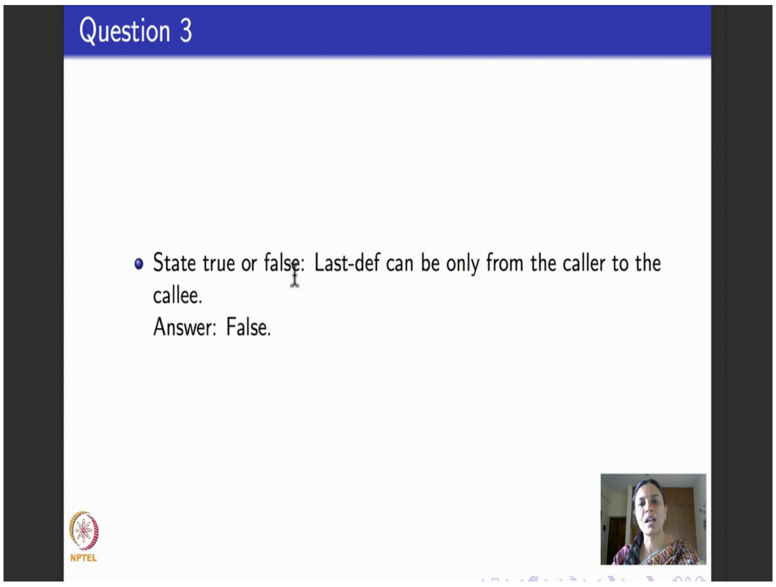
Question 2

- .... and .... are two common types of scaffolding.
  Answer: Test driver and test stub. (or) Test drivers and test stubs.

Second question asks when I do integration testing; I do what is called scaffolding. If you remember the modules that we saw when we looked at classical design integration testing, it says, which are the two most common types of scaffolding? Two most common types of scaffolding are test drivers and test stubs. If you remember, when we do top down integration or bottom up integration, I may or may not have all the modules ready.

So, I write a test stub when I do not have a module ready, but I want it to behave as if a module is ready. So, it is like a dummy module and when do I do a test driver, when I want the dummy module to be able to call the lower level modules, when I do bottom up testing that is when I use test drivers. So, the correct answer to second question will be test stubs and test drivers.

(Refer Slide Time: 02:53)



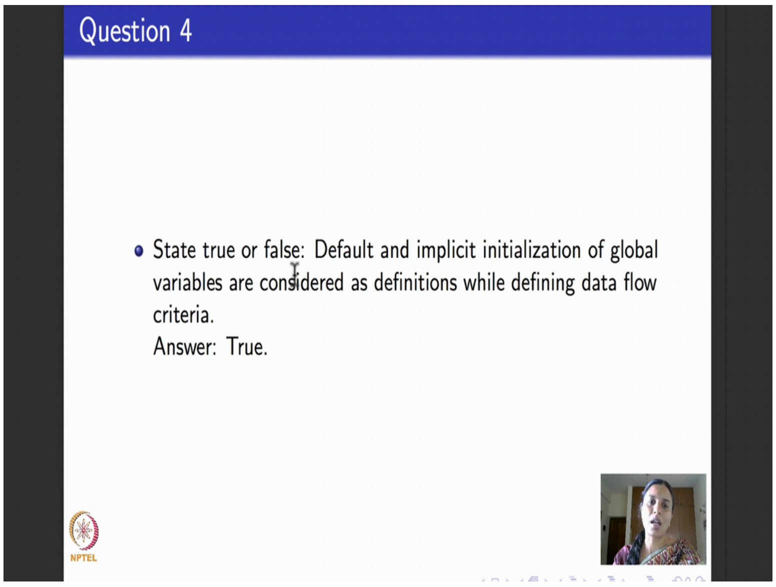
Question 3

- State true or false: Last-def can be only from the caller to the callee.
  Answer: False.

The combinations that are given here basically let you answer them in any order; you should have got full marks if you answered them in any order or in singular or plural words. The third question asks; gives you a statement which is Last-def can only be from the caller back to the callee; from the caller to the callee, sorry not back to the callee. So, is it true or false; so it is obvious that it is false because even in the examples that we saw, if you just remember we saw quadratic root example.

Then we saw small toy examples or just the caller, callee definitions, Last-defs can also be from the callee back to the caller. So, they can be in two different ways, they can be from the caller method to the callee method and they can be given returned back values that are returned back from the callee method back to the caller method.

(Refer Slide Time: 03:44)



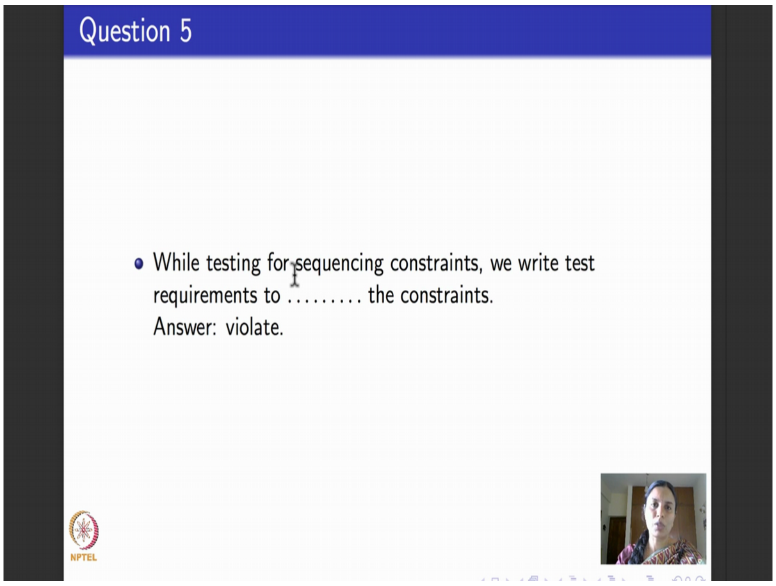
Question 4

- State true or false: Default and implicit initialization of global variables are considered as definitions while defining data flow criteria.
  Answer: True.

So, this statement is false; they can be both ways, it cannot be just one way. The fourth question asks again another true or false statement, it asks whether default and implicit initializations of global variables. Specifically, when it comes to programs like C or Java; the programs that are written in C or Java are they to be considered as definitions or not. Because they are implicit initializations, they are not explicitly given by the program; it does not mean that the values are not there.

So, they are initialized, the values are very much present; so they have to be considered as definitions even though they are not explicitly present a statements in the program. So, the answer to this question is true. Default and implicit initializations of global variables should be considered as definitions while working with data flow criteria.

(Refer Slide Time: 04:33)

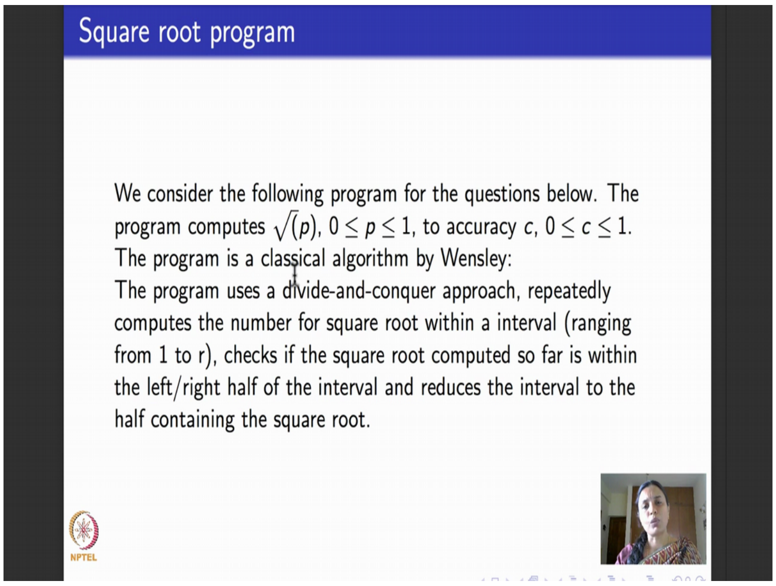


Question 5

- While testing for sequencing constraints, we write test requirements to .... the constraints.
  Answer: violate.

The fifth question was the fill in the blank question; it asks you while you are testing for sequencing constraints, how are do we write typically the t r or the test requirements for. If you remember we looked at this file a d t example and then that file a d t example had a few methods that it was exposing; open a file, read a file, write a file, close a file. So, typically we write test requirements to violate the sequencing constraint so that we can flag it as an error; so, the answer to this fill in the blank question is; violate.

(Refer Slide Time: 05:12)



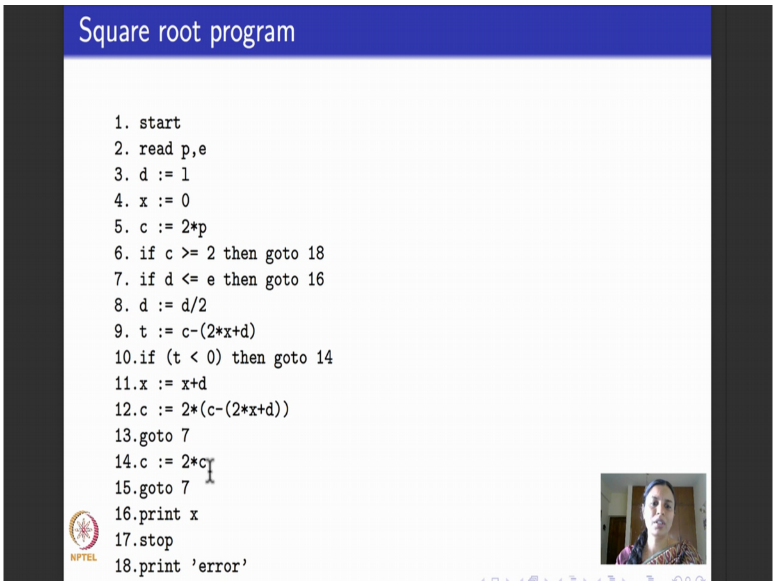
Square root program

We consider the following program for the questions below. The program computes $\sqrt(p)$, $0 \leq p \leq 1$, to accuracy c, $0 \leq c \leq 1$.
The program is a classical algorithm by Wensley:
The program uses a divide-and-conquer approach, repeatedly computes the number for square root within a interval (ranging from 1 to r), checks if the square root computed so far is within the left/right half of the interval and reduces the interval to the half containing the square root.

From question 6 onwards; in the fourth assignment, you were given a program that was computing the square root. The program was given without any loops, but it had a loop, in fact it had more than one loop in the form of go to statements. I have taken this program from the classical papers by Sandra Rapps and Elaine Weyuker, 1982 paper; which I gave you as a reference for data flow testing. This program is from that paper, there were a few questions about that program; specifically related to dataflow coverage criteria.

So, what is the program doing; the program was trying to compute square root of p, where p was this number up to some kind of allowed accuracy and it is a classical algorithm by this Wensley. You can Google for it and check, it uses what is called a divide and conquer approach; it tries to look at first look at the interval 0 to p and try to see where the square root of p could be. Will it be in the first half or will it be in the second half and based on what it says, it breaks the interval in two halves, repeats this procedure again. Breaks the next interval into two halves; repeats this procedure again and it narrows down the interval where square root of p comes and when it finally, terminates an outputs; it outputs a value that is almost the square root of p; modulo a small difference in accuracy.

(Refer Slide Time: 06:34)



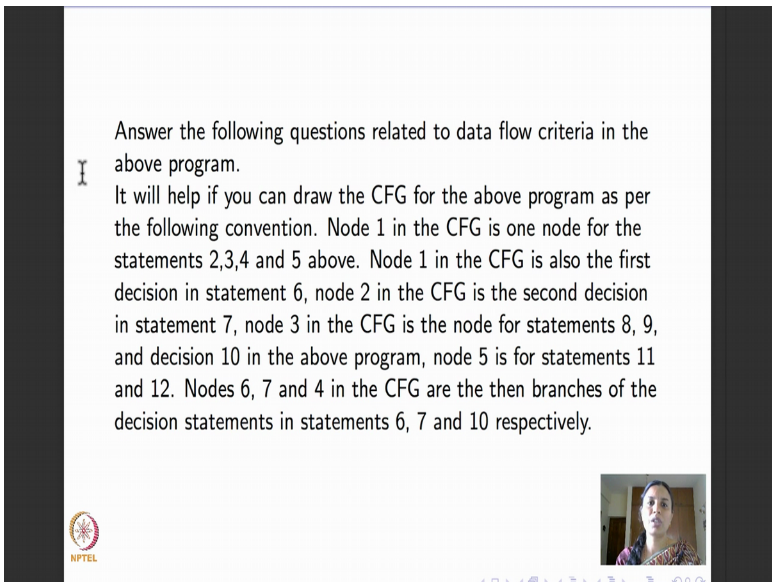
Square root program

```
1. start
2. read p,e
3. d := 1
4. x := 0
5. c := 2*p
6. if c >= 2 then goto 18
7. if d <= e then goto 16
8. d := d/2
9. t := c-(2*x+d)
10.if (t < 0) then goto 14
11.x := x+d
12.c := 2*(c-(2*x+d))
13.goto 7
14.c := 2*c
15.goto 7
16.print x
17.stop
18.print 'error'
```

So, here is the code of the program written specifically in long way that helps you to learn also how to draw control flow graphs. As I told you, the code the program has while loops not as while loops or for loops, it has them as go to statements. So, here I begin; I read the numbers, I do a set of initializations and then I say if this number is greater or equal to 2; then you go to 18, 18 means I cannot find square root; print error and let say if the number is less than equal to e; go to 16; 16 says I have found the square root printed.

So, then where is the loop; the loop is around here, so there is a if statement here; it computes d and t. If t is less than 0, it says go to 14; 14 does 2 star c and then it says go back to 7. So, you do this then you could either go to 16 or you continue 8, 9, 10, 14 and go back to 7, so there is a loop here.

(Refer Slide Time: 07:35)



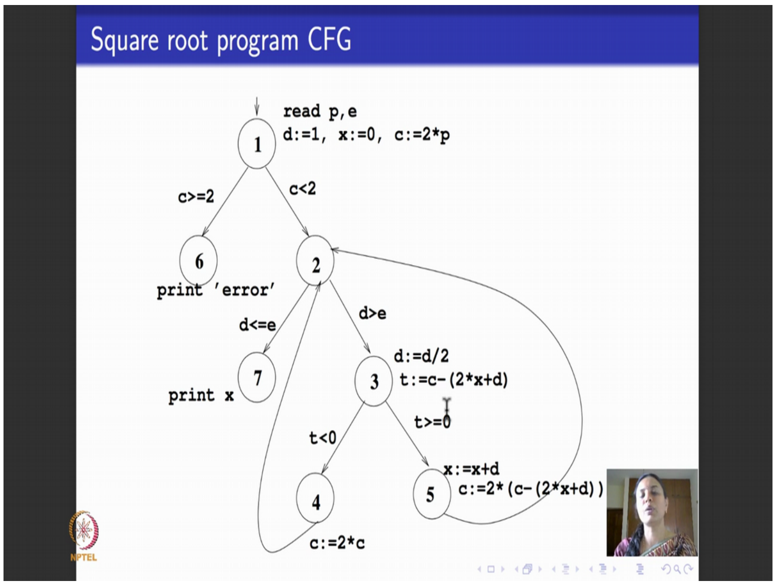
Answer the following questions related to data flow criteria in the above program.
It will help if you can draw the CFG for the above program as per the following convention. Node 1 in the CFG is one node for the statements 2,3,4 and 5 above. Node 1 in the CFG is also the first decision in statement 6, node 2 in the CFG is the second decision in statement 7, node 3 in the CFG is the node for statements 8, 9, and decision 10 in the above program, node 5 is for statements 11 and 12. Nodes 6, 7 and 4 in the CFG are the then branches of the decision statements in statements 6, 7 and 10 respectively.

So, what was the question about? The question was about, first it asked you to draw a CFG; there were no marks allotted to drawing of the CFG. But the questions that were to follow were based on the CFG, especially four of those questions; the first one was not; the later questions were based on those of drawing the CFG and because this program has a lot of statements and the questions were based on parts in the CFG, I had also given you some guidelines on how to label the vertices of your CFG; so, these are the guidelines.

So, what it says is the first node in your CFG; call it node 1, should be one node that represents this bunch of statements that are in one after the other 2, 3, 4 and 5. So, we will go back to the program statement 2; read p e, 3 this statement d is equal to 1, 4 x is equal to 0, 5 c is equal to 2 star p, all this at the beginning of the program; will be node 1 in the CFG; that is what was given here and node 1 also includes this first decision statement at line number 6.

Then there is a branching into node 2 and some another node and in node 2; the second decision statement comes which means this decision statement is false; second decision statement comes and then d by 2 t and so on; that is node 3 which is statements 8 and 9; node 3 in the CFG and so on.

(Refer Slide Time: 09:07)


Square root program CFG

So, here is how this CFG will look like; if you had made an attempt to draw it using pen and paper. So, node 1 represented with the initial node of the program; where the program computation begins. It represents these four statements read p e, d is equal to 1, x assigned to 0, c is 2 star which are these four statements and it also represents the decision. If the, if statement turns out to be true; that is if c is greater than or equal to 2, go to 18; 18 says print error. So in the CFG, you go to some node call it 6; as per the convention that I had given you to do in the question, which is labeled with the statement print error.

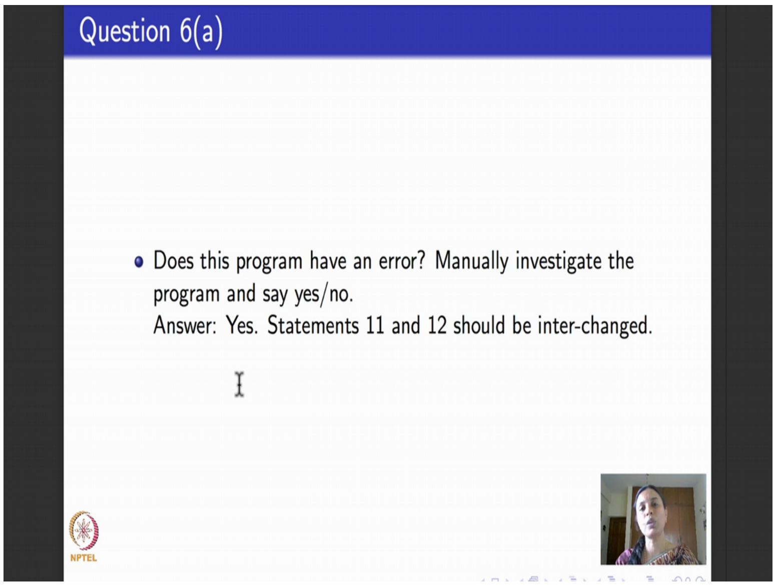
Otherwise you go to statement node 2 in the CFG, which is the next if statement; d is less than or equal to e. So, what you do here if d is less than or equal to e, then you print x go back to the code; if d is less than or equal to e, go to 16; 16 says print x. Otherwise d is greater than e, you come here d is equal to d by 2; statements 8, 9, 10. So, d is equal to d by 2 statements 8; t is equal to c minus 2 into x plus d, statement 9 and the decision at 10 is also modeled here.

Decision could be positive or negative, it checks if t is less than 0; if t is less than 0; it asks you to go to 14; where you do c is equal to 2 star c, if t is less than 0; then you do c is equal to 2 star c and then what you do; you go back to 7; which is node 2 first. If t is greater or equal to 0; then you do two more statements and you go back to 7. So, these are the loops that I was telling you; that the program has, divides the number by 2, checks the range of the square root, takes the first half of the second half based on what the where the square root it keeps doing it. It reaches the value that is approximately the same as that of the square root, but these loops that you find in the CFG are not directly given as while statements or for statements in the program, they are given in terms of goto’s.

So, I hope you all could get a CFG that looks somewhat like this, even if it was not drawn like this; your CFG should have had 7 nodes and they should have been 3 decision points. The point 1 here, the point 2 here, point 3 here; corresponding to 3 if statements that were in the code; one here, one here and one here and these goto’s tell you to loop back and that is what we have done here; the goto’s loop back correctly.

Please do not get confused with the numbers 1, 2, 3, 4 that you find in the nodes here with the statement numbers in the program. These are different, they do not mean the same, so when I say goto 7 here, I mean go to statement number 7 in the program; statement number 7 in the program is actually node number 2 in the CFG. So, please do not mix up these two, so once you have drawn the CFG; you should be able to answer most of the questions that were asked.

(Refer Slide Time: 12:18)



Question 6(a)

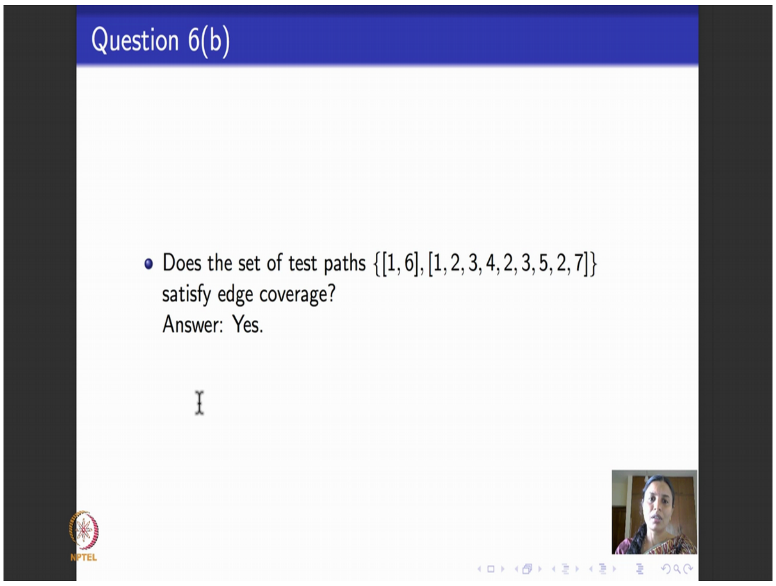
- Does this program have an error? Manually investigate the program and say yes/no.
  Answer: Yes. Statements 11 and 12 should be inter-changed.

So now we look at one question after the other and see; what were the questions that were asked? So, the first question in fact, was not dependent on the CFG; it just asked you to look at the program, to study the program and find out on your own without resorting to any graph based testing or any other testing; does this program have an error. We typically give these kinds of questions in a testing course, to help you to appreciate that the goal of a tester is to be able to find an error.

So if you had done this, if you just studied the program and understood what it was doing with reference to divide and conquer algorithm; you will realize that the program did have an error. The error in the program was the order in which statements 11 and 12 were given to you; got interchanged. It should first compute this value, only then can it increment or reset the interval value of x, but the program was first incrementing x and then computing c, it will miss computing certain values of the square.

So, you take a small enough number and let the program go through it; you will realize that the statements 11 and 12 should be interchanged, so that is the error in the program this program does have an error. So, the rest of the questions are aimed at helping you to understand data flow coverage and specifically can we use any of the data flow coverage criteria that we saw to find this error, the error that says statement 11 and 12 were indeed interchanged.

(Refer Slide Time: 13:46)

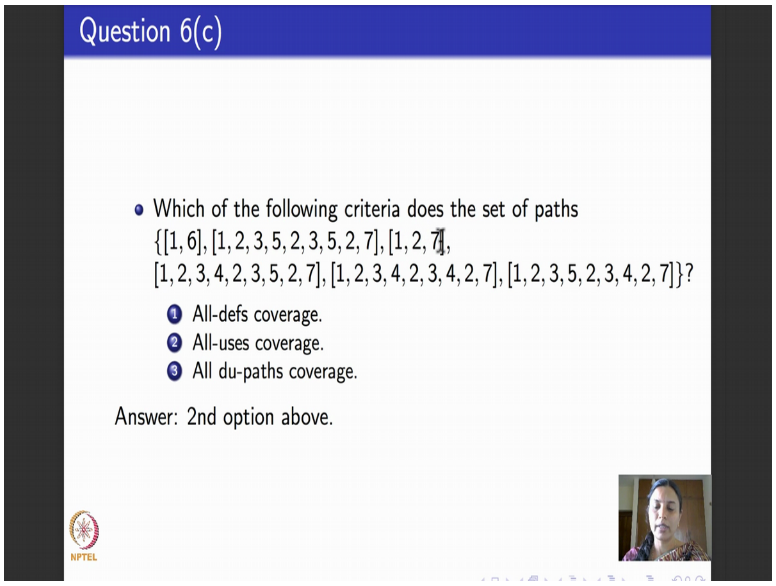


Question 6(b)

- Does the set of test paths $\{[1,6],[1,2,3,4,2,3,5,2,7]\}$ satisfy edge coverage?
  Answer: Yes.

So, the second question was does the set of test paths 1, 6; 1, 2, 3, 4, 2, 3, 5, 2, 7 satisfy edge coverage. So, please remember these test paths 1, 6 and then 1, 2, 3, 4, 2, 3, 5, 2, 7; we will go back to the CFG 1, 6 and then 1, 2, 3, 4, 2, 3, 5 and then 7, does this satisfy edge coverage? Yes, it does because 1, 6 covers this edge, then 1, 2, 3; we have covered these three edges, 4; this edge is also done 2, 3 once again, but we have already covered it; does not matter; 5, 3, 5; then 5, 2 and 2, 7. So, the answer to that question is yes; it does satisfy edge coverage; in fact it covers the edge 2, 3 twice, but that does not matter; it visits every edge exactly once.

(Refer Slide Time: 14:49)



Question 6(c)

- Which of the following criteria does the set of paths $\{[1,6],[1,2,3,5,2,3,5,2,7],[1,2,7],$ $[1,2,3,4,2,3,5,2,7],[1,2,3,4,2,3,4,2,7],[1,2,3,5,2,3,4,2,7]\}$?
  1. All-defs coverage.
  2. All-uses coverage.
  3. All du-paths coverage.

Answer: 2nd option above.

The second question asked gives you a set of test paths lot of them in fact and it asks you; list three coverage criteria and asks you, which amongst these three coverage criteria are satisfied by this test path? So, if you see the test paths before looking at these test paths; let us go and look at the control flow graph. As I told you; control flow graph has how many branches? There is one branch here, there is one branch here and one branch here corresponding to the three if statements.

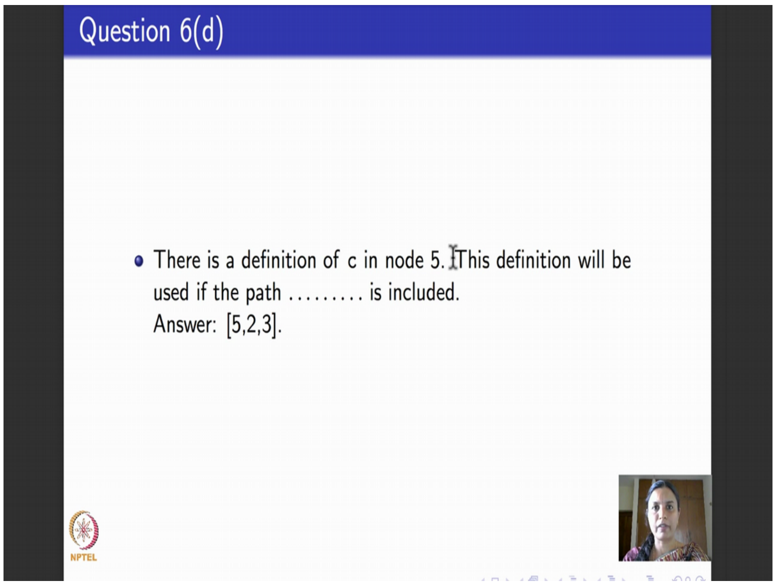
The third if statement had goto’s, so it was like having a loopback. So, if I want to do edge coverage or any kind of coverage I should cover this 1, 6 separately because it is a standalone path and then I can do all these. So, these paths that are given here do 1, 6 separately and then they had this one; which we saw in the last question about edge coverage and then it had 1, 2, 7.

So, let us go back and see what 1, 2, 7 is 1, 2, 7; so 1, 2, 3 took this branch 1, 2, 7 took this branch fine. So, we seem to be doing fine and then what else is there; then it is the same thing as this path, the four paths; they are very similar to the second path. What is the difference? The only difference is that 1, 2, 3, 5 here 1, 2, 3, 4. So, let us see what does that do; 1, 2, 3, 4, 2, 3, so it does cover this part of the second decision statement and similarly this covers the other half of the second decision statement through 4 and this covers the other half of the second decision statement through 5.

So, the question that was asked was; does it fixed coverage criteria does it use All-defs, All-uses or All du-paths. The correct answer is that it does all uses coverage, you could have also said All-defs coverage, but please remember All-uses coverage subsumes all defs coverage. So, we are looking for the stronger option here; the stronger correct answer is All-uses coverage. So, if you had written All-defs coverage; then marks would not have been given unless you had written All-uses coverage as your answer.

It does not cover All du-paths; can you tell me which is the du path that it is going to leave out. It will be good to look at this control flow graph and help yourself by understanding which is the du-path that is left out and why do these set of test paths not satisfy all du-paths coverage. Please do that as a small little self check for yourself, so the correct answer to this question would be All-uses coverage.

(Refer Slide Time: 17:38)



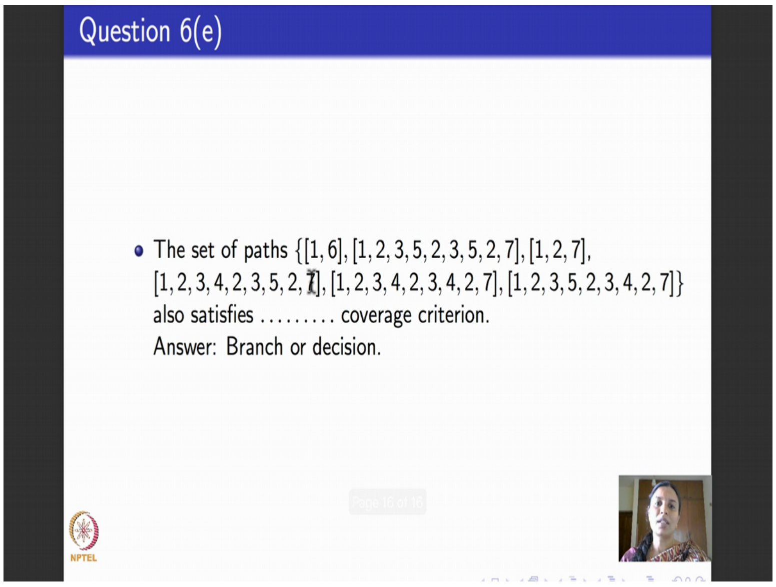
Question 6(d)

- There is a definition of c in node 5. This definition will be used if the path .... is included.
  Answer: [5,2,3].

The next question asks; tells you there is a definition of c in node 5; so, we will go back to the control flow graph and see is there a definition of c in node 5? Yes, because c is done given this assignment statement, so c is defined. Then, it asks you to find out where is c used? How will c be used? The only way to use c would be to go out of 5 and the only way to go out of 5 would be to go back to 2. So, once I go back to 2; I have a choice, I could go to 7 or I could go to 3, but my goal is to be able to see this definition of c where was it used; that was the specific question about.

So, from 5 I go back to 2 and suppose I take the path 2, 7; clearly this path does not result in the use of c, because the use of this edge at d and e and the use of this vertex 7 is x. So, this does not help for that and suppose I take 3; if you see c is used here to define t. A correct answer to that question would be; the definition of c will result in a use if the path 5, 2, 3 is included. You have to be able to do 5, 2 and then go to 3; is that clear?

(Refer Slide Time: 18:51)



Question 6(e)

- The set of paths {[1, 6], [1, 2, 3, 5, 2, 3, 5, 2, 7], [1, 2, 7], [1, 2, 3, 4, 2, 3, 5, 2, 7], [1, 2, 3, 4, 2, 3, 4, 2, 7], [1, 2, 3, 5, 2, 3, 4, 2, 7]} also satisfies .... coverage criterion.
  Answer: Branch or decision.

So, the next question again gives you a whole set of paths and it asks you which is the coverage criteria that it satisfies. A simple answer to this question would be; branch coverage, you could have also written decision coverage that would have also been correct. Why does it cover branch coverage? Because if you see this set of paths nicely take the three branches corresponding to three if statements and these set of paths precisely take those branches. This path takes the first branch 1, 6; then these are the paths out of the second branch 1, 6 and 1, 2 and in 1, 2; you could do 1, 2, 3 or 1, 2, 7 which are the two options and 3; you could do 1, 2, 3, 4 or 1, 2, 3, 5.

So, we will go back to the CFG for a minute, so I do 1, 6 or 1, 2 then at 2; I have a choice 1, 2, 7 or 1, 2, 3 for the two branches and then at 3; I have a choice 1, 2, 3, 4 or 1, 2, 3, 5. So, all possible branches are listed here and the set of paths that this coverage criteria satisfy; would be branch coverage or decision coverage. You should have got full marks, if you had written either of these options.

So, I hope this assignment helped you to understand; look at simple programs, draw their control flow graphs, annotate it with defs and usage and compute data flow and structural coverage criteria over them. What we will do from next lecture onwards; would be to look at logical predicates, coverage criteria over logical predicates and read all these exercises all over again; take source code. See how logic applies to source code, take specifications, and see how logic applies to specification and so on.

Thank you.