

Software Testing, Quality Assurance & Maintenance (ECE453/CS447/SE465): Assignment 1 Solutions

Patrick Lam

I've excluded solutions for questions 2, 3 and 7. I'll try to make them available later, but they won't help you with midterm preparation anyway. The CFG in this version has been slightly corrected; thanks to Daqiang Mo for pointing out an error.

Question 1 (5 points)

(Section 1.3, Q1): *Suppose that coverage criterion C_1 subsumes coverage criterion C_2 . Also suppose that test set T_1 satisfies C_1 on program P and test set T_2 satisfies C_2 , also on P .*

- (5 points) *If P contains a fault, and T_2 reveals the fault, T_1 does not necessarily also reveal the fault. Explain.*

Recall that when a test suite satisfies a coverage criterion, it must address each test requirement generated by the coverage criterion. Let's say that criterion C_2 contains test requirements A, B and C. Test set T_2 satisfies C_2 and also "accidentally" meets test requirement Z. Criterion C_1 must contain test requirements A, B, C, and perhaps also D, but nothing requires C_1 to contain Z. If Z contains the fault, then T_1 might or might not reveal the fault; both C_1 and C_2 are silent about Z.

Question 4 (5 points)

(Section 2.3, Q4) *Consider the pattern matching example in Figure 2.21. In particular, consider the final table of tests in Section 2.3. Consider the*

variable `iSub`. Number the (unique) test cases, starting at 1, from the top of the `iSub` part of the table. For example, $(ab, c, -1)$, which appears twice in the `iSub` portion of the table, should be labelled test t_4 .

- Give a minimal test set that satisfies all defs coverage.
- Give a minimal test set that satisfies all uses coverage.
- Give a minimal test set that satisfies all du-paths coverage.

Note that Table 1 by itself is insufficient for answering this question; you need to combine Table 1 and Table 2. Recall that the textbook puts uses for conditionals on both branches; I like to put the use at the branch point, so either convention is OK, as long as you're consistent. (A couple of you only saw the uses of `iSub` on the nodes. I actually made the same mistake at first; I really don't like the way that the book puts uses on edges for conditionals. It's not intuitive at all to me. Nevertheless, you need to consider, one way or another, the uses at the conditionals, whether you put them on the edges or on the nodes.)

We enumerate the defs and the uses. There are two defs of `iSub`: one at node 2 and one at node 10. There are 2 uses of `iSub` on nodes, at node 5 and at node 10, and six uses on edges: $(3, 4)$, $(3, 11)$, $(4, 5)$, $(4, 10)$, $(7, 8)$, $(7, 9)$.

- We need to cover the def at node 2 and the def at node 10. Note that all executions except t_5 in our table containing 10 subsequently hit the edge $(3, 11)$, covering the def at 10. Similarly, all executions containing 2 also contain either edge $(3, 11)$ or $(3, 4)$, each of which cover the def containing 2. Hence any test case except t_5 makes up a test set which will cover all defs.
- Now we need to cover all du-pairs. Consider the following list of du-pairs and test cases which satisfy them:

$\langle 2, 5 \rangle$	t_1, t_2, t_3	$\langle 10, 5 \rangle$	t_6, t_7, t_8
$\langle 2, 10 \rangle$	$t_1, t_2, t_3, t_4, t_6, t_7, t_8$	$\langle 10, 10 \rangle$	t_4, t_6, t_7, t_8
$\langle 2, (3, 4) \rangle$	$t_1, t_2, t_3, t_4, t_6, t_7, t_8$	$\langle 10, (3, 4) \rangle$	t_4, t_6, t_7, t_8
$\langle 2, (3, 11) \rangle$	t_5	$\langle 10, (3, 11) \rangle$	$t_1, t_2, t_3, t_4, t_6, t_7, t_8$
$\langle 2, (4, 5) \rangle$	t_1, t_2, t_3	$\langle 10, (4, 5) \rangle$	t_6, t_7, t_8
$\langle 2, (4, 10) \rangle$	t_4, t_6, t_7, t_8	$\langle 10, (4, 10) \rangle$	t_4
$\langle 2, (7, 8) \rangle$	t_3	$\langle 10, (7, 8) \rangle$	t_8
$\langle 2, (7, 9) \rangle$	t_1	$\langle 10, (7, 9) \rangle$	t_6

We see that each test in $\{t_1, t_3, t_4, t_5, t_6, t_8\}$ appears alone and is therefore mandatory in any test set that is to achieve all-uses coverage. Furthermore, this test set covers all du-pairs and therefore achieves all-uses coverage.

- Finally, we need to cover all du-paths. For this example, we know that all-du paths coverage is feasible, since the text claims that it is feasible for all variables in the program, so it is clearly feasible for just `iSub`. Since all-du-paths subsumes all-uses, we know that we need at least the tests in the previous question. We can just check to see if we need t_2 and t_7 to achieve all-du-paths.

We see that t_2 appears in $\langle 2, 5 \rangle, \langle 2, 10 \rangle, \langle 2, (3, 4) \rangle, \langle 2, (4, 5) \rangle, \langle 10, (3, 11) \rangle$. To show that t_2 is mandatory, we look at Table 2.5 and observe that only t_2 covers the du-path $[2, 3, 4, 5, 6, 10]$. (Of course, if you are unlucky and try something other than $\langle 2, 10 \rangle$ first, then you get more work.)

We next check t_7 , which appears in the du-pairs $\langle 2, 10 \rangle, \langle 2, (3, 4) \rangle, \langle 2, (4, 10) \rangle, \langle 10, 5 \rangle, \langle 10, 10 \rangle, \langle 10, (3, 4) \rangle, \langle 10, (3, 11) \rangle, \langle 10, (4, 5) \rangle$. We observe that the path $[10, 3, 4, 5, 6, 10]$ is a du-path for `iSub` which appears only in test case t_7 .

Hence the full set $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$ is the minimal set to cover all-du-paths.

Question 5 (10 points)

(Section 2.3, Q7) Use the method `printPrimes()` for questions a-f below.

- (1) Draw the control flow graph for the `printPrimes()` method.
- (3) Consider test cases $t1$: ($n = 3$) and $t2$: ($n = 5$). Although these tour the same prime paths in `printPrimes()`, they do not necessarily find the same faults. Design a simple fault that $t2$ would be more likely to discover than $t1$ would.
- (3) For `printPrimes()`, find a test case such that the corresponding test path visits the edge that connects the beginning of the while statement to the for statement without going through the body of the while loop.

- (1) Enumerate the test requirements for Node Coverage, Edge Coverage, and Prime Path Coverage for the graph for **printPrimes()**.
- (1) List test paths that achieve Node Coverage but not Edge Coverage.
- (1) List test paths that achieve Edge Coverage but not Prime Path Coverage on the graph.

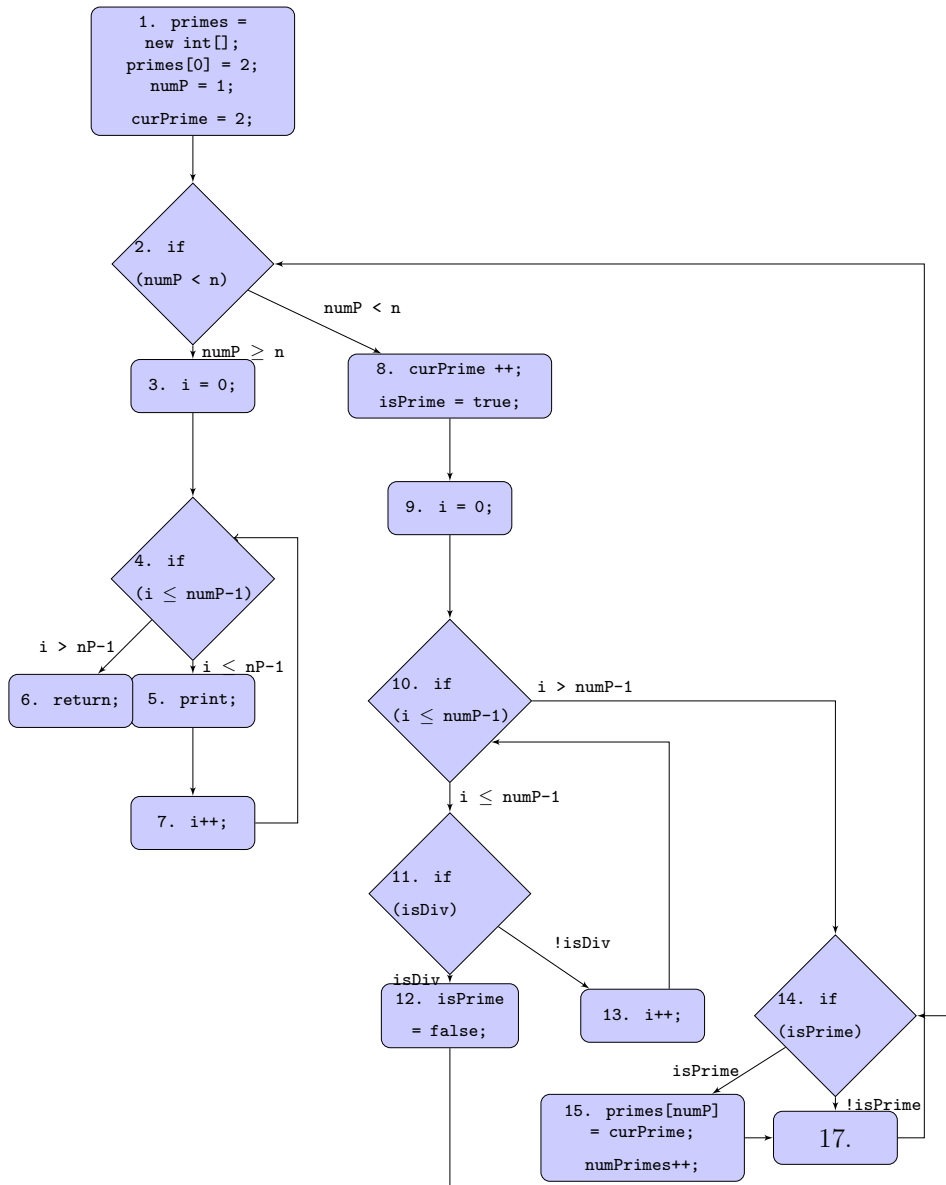


Figure 1: Control-flow graph for `printPrimes()`.

- The case $n = 3$ does not find many faults at all. One example of a fault that $n = 3$ does not find is swapping the order of parameters in

`isDivisible`: writing `if (isDivisible (curPrime, primes[i]))` at line 22 would be hidden for $n = 3$ and visible for $n = 5$. (Many other faults are possible.)

- This question asks about the `for` loop on line 36 (it should have been more clear). The case $n = 1$ never executes the `while` body.
- Node coverage is simple: cover all of the numbered nodes in your CFG. Because I introduced a dummy node at 17, I should cover it; if you don't use one, you don't need to cover it. Edge coverage requires the following edges: (1, 2), (2, 3), (2, 8), (3, 4), (4, 5), (4, 6), (5, 7), (7, 4), (8, 9), (9, 10), (10, 11), (10, 14), (11, 12), (11, 13), (13, 10), (12, 3), (14, 15), (14, 17), (15, 17), (17, 2). You can use the program you wrote for Q2 to calculate the test requirements for prime path coverage!

Question 6 (5 points)

(Section 2.5, Q2) For the following questions a-c, consider the method *FSM* for a (simplified) programmable thermostat. Suppose the variables that define the state and the methods that transition between states are:

```
partOfDay : {Wake, Sleep}
temp      : {Low, High}

// Initially "Wake" at "Low" temperature

// Effects: Advance to next part of day
public void advance();

// Effects: Make current temp higher, if possible
public void up();

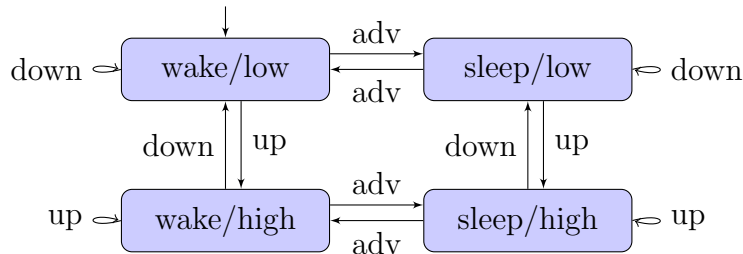
// Effects: Make current temp lower, if possible
public void down();
```

- (1) How many states are there?
- (2) Draw and label the states (with variable values) and transitions (with method names). Notice that all of the methods are total.

- (2) A test case is simply a sequence of method calls. Provide a test set that satisfies Edge Coverage on your graph.

There was some confusion about the meaning of “wake” and “sleep” as parts-of-day. Just treat them as names; it’s not the case, for instance, that the temperature doesn’t change while the part-of-day is “sleep”.

The FSM contains $2 \times 2 = 4$ states, as we’ll see below.



A test case satisfying edge coverage is: `down()`, `up()`, `up()`, `down()`, `adv()`, `down()`, `up()`, `up()`, `adv()`, `adv()`, `down()`, `adv()`.