Assignment HW7

**Cover Page**

Prepared for:

Dr. Mehra Borazjany

Trung Hieu Tran

Prepared by:

Alex Lundin

Daniel Neal

SE-4367.0U1-Testing

**Assignment Choice:**

N/A for this assignment

10 July, 2018

**Proof of Working Software**

GitHub link:

<https://github.com/AlexLundinEducational/SE-4367-Testing>

Branch Summary:

master – managed by Alex, only fully completed pulls allowed to make TA’s life easy. Master only contains assignment material once they reach completed status.

working – flexible branch for team, ideally, this material should build without causing technical debt during the project.

Commit for grading:

HW7\_Team\_4 Complete, Ready for Merge to master and Ready for Grading

Phase 1 Development

Phase 2 Development

Phase 3 Development

Graphs: created with draw.io plugin for OneDrive

<https://cometmail-my.sharepoint.com/:f:/g/personal/aml140830_utdallas_edu/EtAfX0sIEZlJnp-ekn87Jc8B6lxRT7ic4WXA5p8q70NnCw?e=GNShuO>

1.(30 points) The following set of edges defines a graph:

•E = { (1, 2), (1, 3), (1, 4), (2, 7), (3, 2), (3, 4), (3, 5), (3, 6), (4, 7), (5, 2), (5, 7), (6, 4), (6, 7), (7, 8), (7, 9) }

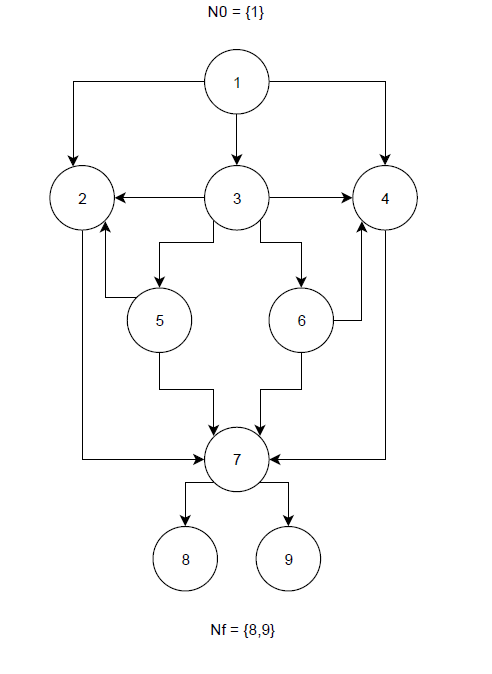
•N0 = { 1 }

•Nf = { 8, 9 }

Use this graph to answer the following questions:

a.(5points)

Draw the graph. Be sure to annotate the initial and final nodes correctly.



b.(2points)

Is the sequence [1, 3, 6, 4, 7, 9] a Test Path?

**Yes, it starts on an initial node and ends on a final node.**

c.(2points)

Is the sequence [1, 3, 2, 7] a Test Path?

**No, it does not end on a final node.**

d.(2points)

Is the sequence [1, 2, 4, 7, 9] a Test Path?

**No, this is not a reachable path. 2 does not connect to 4.**

e.(3points)

What is the reach set for node 4?

**Reach (4): [4,7,8,9]**

f.(3points)

What is the length of the shortest test path?

**4**

g.(3points)

Does the Test Path [1, 3, 6, 7, 9] tour subpath [6, 4, 7, 9]?

**No.**

h.(5points)

Give the test requirements for edge-pair coverage.

**(1,2,7) (1,3,2) (1,3,4) (1,3,5) (1,3,6) (1,4,7)**

**(2,7,9)! (2,7,8)!**

**(3,2,7) (3,5,2) (3,5,7) (3,6,4) (3,6,7) (3,4,7)**

**(4,7,9)! (4,7,8)!**

**(5,2,7) (5,7,8)! (5,7,9)!**

**(6,4,7) (6,7,8)! (6,7,9)!**

i.(5points)

Give the test requirements for prime path coverage

**len3: (1,2,7,8)! (1,2,7,9)! (1,4,7,8)! (1,4,7,9)!**

**len4: (1,3,6,7,8)! (1,3,6,7,9)! (1,3,5,7,8)! (1,3,5,7,9)! (1,3,2,7,8)! (1,3,2,7,9)! (1,3,4,7,8)! (1,3,4,7,9)!**

**len5: (1,3,5,2,7,8)! (1,3,5,2,7,9)! (1,3,6,4,7,8)! (1,3,6,4,7,9)!**

2.(60points) (Ch7-Exercises #7 of Section 7.3)

Use the method printPrimes() for questions a–f below.

1. /\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. \* Finds and prints n prime integers

3. \* Jeff Offutt, Spring 2003

4. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

5. private static void printPrimes (int n)

6. {

7. int curPrime; // Value currently considered for primeness

8. int numPrimes; // Number of primes found so far.

9. boolean isPrime; // Is curPrime prime?

10. int [] primes = new int [MAXPRIMES]; // The list of prime numbers.

11.

12. // Initialize 2 into the list of primes.

13. primes [0] = 2;

14. numPrimes = 1;

15. curPrime = 2;

16. while (numPrimes < n)

17. {

18. curPrime++; // next number to consider ...

19. isPrime = true;

20. for (int i = 0; i <= numPrimes-1; i++)

21. { // for each previous prime.

22. if (isDivisible (primes[i], curPrime))

23. { // Found a divisor, curPrime is not prime.

24. isPrime = false;

25. break; // out of loop through primes.

26. }

27. }

28. if (isPrime)

29. { // save it!

30. primes[numPrimes] = curPrime;

31. numPrimes++;

32. }

33. } // End while

34.

35. // Print all the primes out.

36. for (int i = 0; i <= numPrimes-1; i++)

37. {

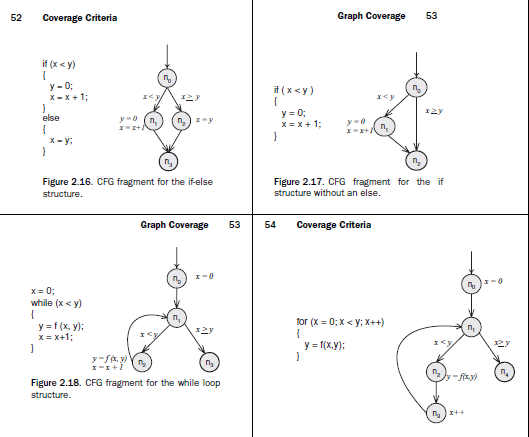
38. System.out.println ("Prime: " + primes[i]);

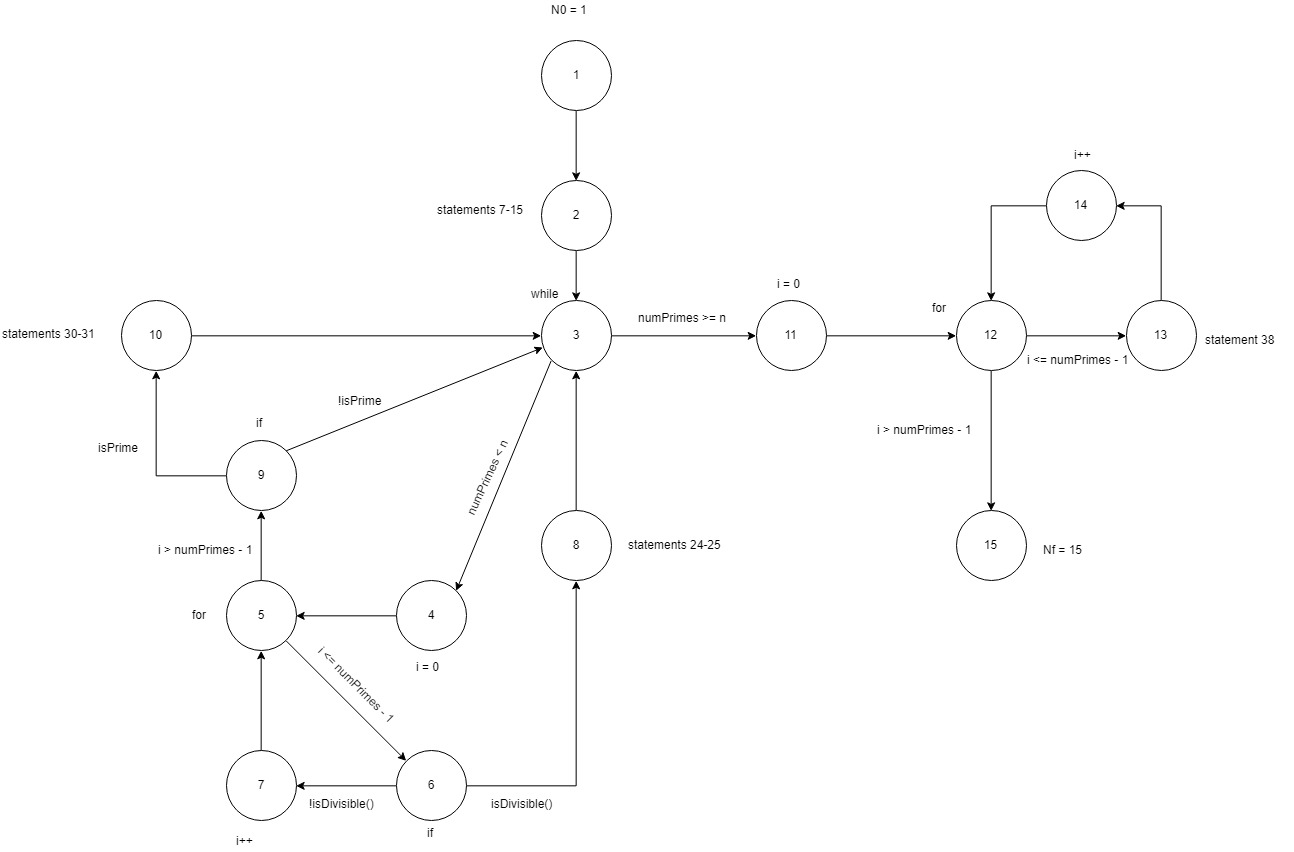
39. }

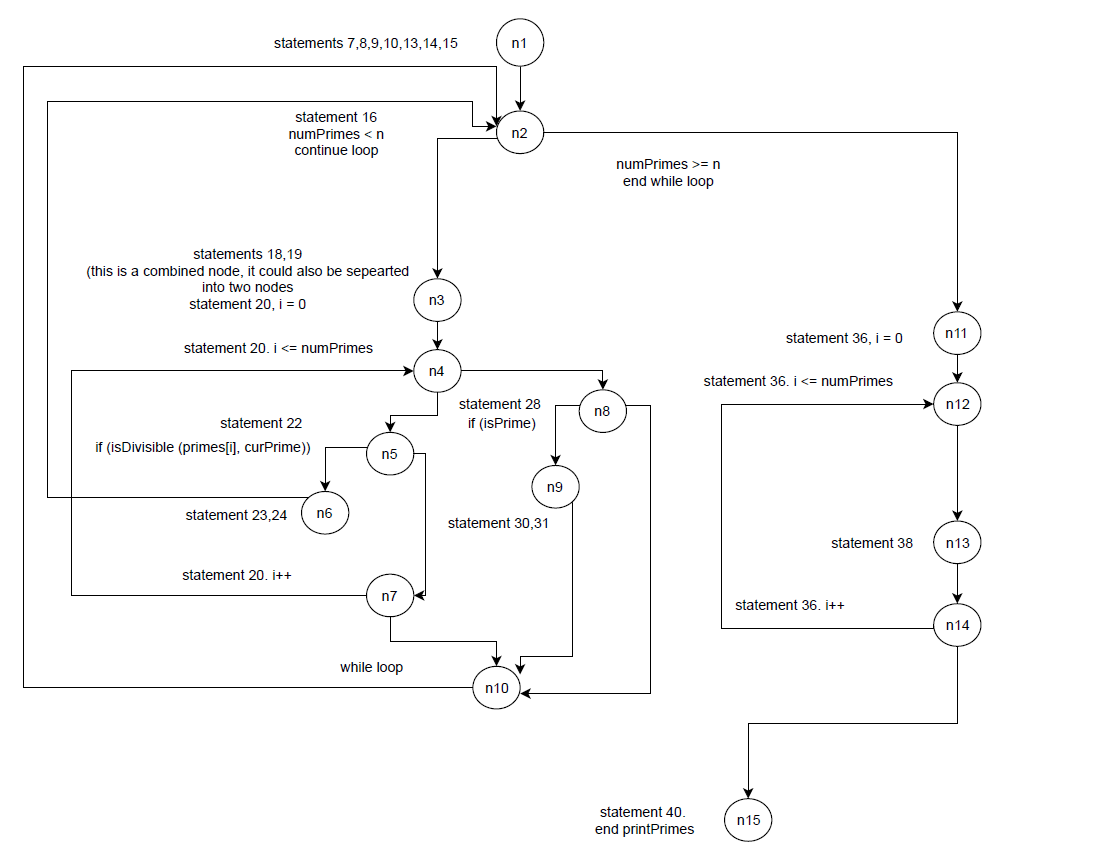
40. } // end printPrimes

a.(10 points)

Draw the control flow graph for the printPrimes() method.







b.(10 points)

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.

**This question seems based on distinguishing which nodes t1 and t2 will cover. Since t2 specifically reaches the if condition, it will find any fault with the break statement logic. T1 is less likely to find the break statement faults, because there is no guarantee that we will enter the nested loop body, at n4**.

**The designed fault is accidentally setting isPrime = true; before breaking.**

c.(10points)

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.

**T3 = (n = 6)**

**This test case specifics we must test the edge that connects the beginning of the while to the for statement, there are two for statements, so I’m assuming this means the for statement n4, on the left side of the graph.**

**This test case also specifies we must reach node 6, which breaks out of the while loop, ensuring we don’t execute the body of the while loop, which is required in the question.**

Definitions for d e and f

**Node coverage = statement coverage, block coverage, each reachable node (s15-Ch07, Slide 12)**

**Edge coverage = “up to length 1”, branch coverage (s15-Ch07, Slide 13)**

**Simple path = path from ni to nj where no node appears more than once, except possibly, first and last(s15-Ch07, Slide 19)**

**Prime Path = A simple path that does not appear as a proper subpath of any other simple path (s15-Ch07, Slide 19)**

**^ Slides s15-Ch07**

**Edge Coverage:**

**Node Coverage:**

**Simple paths:**

**Prime Paths:**

d.(10points)

List the test requirements for Node Coverage, Edge Coverage, and Prime Path Coverage.

**TR(node): { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15! }**

**TR(edge): { (1,2), (2,3), (3,4), (3,11), (4,5), (5,6), (5,9), (6,8), (6,7), (7,5), (8,3), (9,10), (9,3), (10,3), (11,12), (12,13), (12,15)!, (13,14), (14,12) }**

**TR(prime): (47 total prime paths)**

**1. [6,7,5,9,10,3,11,12,13,14]**

**2. [4,5,9,10,3,11,12,13,14]**

**3. [4,5,6,8,3,11,12,13,14]**

**4. [6,7,5,9,10,3,11,12,15]!**

**5. [7,5,6,8,3,11,12,13,14]**

**6. [6,7,5,9,3,11,12,13,14]**

**7. [4,5,9,3,11,12,13,14]**

**8. [4,5,9,10,3,11,12,15]!**

**9. [4,5,6,8,3,11,12,15]!**

**10. [7,5,6,8,3,11,12,15]!**

**11. [6,7,5,9,3,11,12,15]!**

**12. [4,5,9,3,11,12,15]!**

**13. [1,2,3,4,5,6,7]**

**14. [1,2,3,4,5,6,8]**

**15. [1,2,3,4,5,9,10]**

**16. [1,2,3,11,12,13,14]**

**17. [9,10,3,4,5,6,8]**

**18. [9,10,3,4,5,6,7]**

**19. [6,8,3,4,5,9,10]**

**20. [6,7,5,9,10,3,4]**

**21. [5,6,8,3,4,5]\***

**22. [5,9,10,3,4,5]\***

**23. [4,5,9,10,3,4]\***

**24. [6,8,3,4,5,6]\***

**25. [1,2,3,11,12,15]!**

**26. [3,4,5,9,10,3]\***

**27. [4,5,6,8,3,4]\***

**28. [3,4,5,6,8,3]\***

**29. [9,3,4,5,6,8]**

**30. [9,10,3,4,5,9]\***

**31. [9,3,4,5,6,7]**

**32. [10,3,4,5,9,10]\***

**33. [7,5,6,8,3,4]**

**34. [6,7,5,9,3,4]**

**35. [8,3,4,5,6,7]**

**36. [8,3,4,5,6,8]\***

**37. [4,5,9,3,4]\***

**38. [5,9,3,4,5]\***

**39. [3,4,5,9,3]\***

**40. [9,3,4,5,9]\***

**41. [5,6,7,5]\***

**42. [6,7,5,6]\***

**43. [13,14,12,15]!**

**44. [14,12,13,14]\***

**45. [13,14,12,13]\***

**46. [12,13,14,12]\***

**47. [7,5,6,7]\***

e.(10 points)

List test paths that achieve Node Coverage but not Edge Coverage on the graph.

**Test paths:**

**[1,2,3,11,12,13,14,12,15]**

**[1,2,3,4,5,9,10,3,11,12,15]**

**[1,2,3,4,5,6,8,3,11,12,15]**

**[1,2,3,4,5,6,7,5,9,3,11,12,15]**

f. (10 points)

List test paths that achieve Edge Coverage but not Prime Path Coverage on the graph.

**Test paths:**

**[1,2,3,4,5,6,7,5,9,10,3,11,12,15]**

**[1,2,3,4,5,6,8,3,11,12,13,14,12,15]**

**[1,2,3,4,5,9,3,11,12,15]**

Remember to describe who did what

(10 points)