3. Below are four faulty programs. Each includes a test case that results in failure. Answer the following questions about each program.

```
public int findLast (int[] x, int y)
{
//Effects: If x==null throw NullPointerException
//   else return the index of the last element
//   in x that equals y.
//   If no such element exists, return -1
   for (int i=x.length-1; i > 0; i--)
   {
      if (x[i] == y)
      {
         return i;
      }
   }
   return -1;
}
   // test:  x=[2, 3, 5]; y = 2
   //        Expected = 0
```

```
public static int lastZero (int[] x)
{
//Effects: if x==null throw NullPointerException
//   else return the index of the LAST 0 in x.
//   Return -1 if 0 does not occur in x

   for (int i = 0; i < x.length; i++)
   {
      if (x[i] == 0)
      {
         return i;
      }
   }
   return -1;
}
   // test:  x=[0, 1, 0]
   //        Expected = 2
```

```
public int countPositive (int[] x)
{
//Effects: If x==null throw NullPointerException
//   else return the number of
//        positive elements in x.
   int count = 0;
   for (int i=0; i < x.length; i++)
   {
      if (x[i] >= 0)
      {
         count++;
      }
   }
   return count;
}
   // test:  x=[-4, 2, 0, 2]
   //        Expected = 2
```

```
public static int oddOrPos(int[] x)
{
//Effects: if x==null throw NullPointerException
// else return the number of elements in x that
//      are either odd or positive (or both)
   int count = 0;
   for (int i = 0; i < x.length; i++)
   {
      if (x[i]%2 == 1 || x[i] > 0)
      {
         count++;
      }
   }
   return count;
}
   // test:  x=[-3, -2, 0, 1, 4]
   //        Expected = 3
```

(a) Identify the fault.
(b) If possible, identify a test case that does **not** execute the fault.
(c) If possible, identify a test case that executes the fault, but does **not** result in an error state.
(d) If possible identify a test case that results in an error, but **not** a failure. Hint: Don't forget about the program counter.
(e) For the given test case, identify the first error state. Be sure to describe the complete state.
(f) Fix the fault and verify that the given test now produces the expected output.

`findLast()` Solution (Instructor only):

(a) *The for-loop should include the 0 index:*
`for (int i=x.length-1; i >= 0; i--) {`

(b) *The null value for x will result in a NullPointerException before the loop test is evaluated–hence no execution of the fault.*

Input:	*x = null; y = 3*
Expected Output:	*NullPointerException*
Actual Output:	*NullPointerException*

(c) *For any input where* `y` *appears in the second or later position, there is no error. Also, if* `x` *is empty, there is no error.*

Input:	*x = [2, 3, 5]; y = 3;*
Expected Output:	*1*
Actual Output:	*1*

(d) *For an input where* `y` *is not in* `x`*, the missing path (i.e. an incorrect PC on the final loop that is not taken) is an error, but there is no failure.*

Input:	*x = [2, 3, 5]; y = 7;*
Expected Output:	*-1*
Actual Output:	*-1*

(e) *Note that the key aspect of the error state is that the PC is outside the loop (following the* `false` *evaluation of the* `0>0` *test. In a correct program, the PC should be at the if-test, with index* `i==0`*.*

Input:	*x = [2, 3, 5]; y = 2;*
Expected Output:	*0*
Actual Output:	*-1*
First Error State:	
	x = [2, 3, 5]
	y = 2;
	i = 0 (or undefined);
	PC = just before `return -1;`*;*

(f) *See (a)*

`lastZero()` **Solution:**

(a) *The for-loop should search high to low:*
`for (int i=x.length-1; i >= 0; i--) {`

(b) *All inputs execute the fault - even the null input.*

(c) *If the loop is not unrolled at all, there is no error. Also if the loop is only unrolled once, high-to-low and low-to-high evaluation are the same. Hence there is no error for length 0 or length 1 inputs.*

Input:	*x = [3]*
Expected Output:	*-1*
Actual Output:	*-1*

(d) *There is an error anytime the loop is unrolled more than once, since the values of index* `i` *ascend instead of descend.*

Input:	*x = [1, 0, 3]*
Expected Output:	*1*
Actual Output:	*1*

(e) *The first error state is when index* `i` *has the value* `0` *when it should have a value at the top of the array. Hence, the first error state is encountered immediately after the assignment to* `i` *in the for-statement if there is more than one value in* `x`.

Input:	*x = [0, 1, 0]*
Expected Output:	*2*
Actual Output:	*0*
First Error State:	
	x = [0, 1, 0]
	i = 0
	PC = just after `i= 0;`

(f) *See (a)*

`countPositive()` **Solution (Instructor only):**

(a) *The test in the conditional should be:*
`if (x[i] > 0) {`

(b) `x` *must be either null or empty. All other inputs result in the fault being executed. We give the empty case here.*

Input:	*x = []*
Expected Output:	*0*
Actual Output:	*0*

(c) *Any nonempty* `x` *without a 0 entry works fine.*

Input:	*x = [1, 2, 3]*
Expected Output:	*3*
Actual Output:	*3*

(d) *For this particular program, every input that results in error also results in failure. The reason is that error states are not repairable by subsequent processing. If there is a 0 in* `x`*, all subsequent states (after processing the 0) will be error states no matter what else is in* `x`*.*

(e)

Input:	*x = [-4, 2, 0, 2]*
Expected Output:	*2*
Actual Output:	*3*

First Error State:
x = [-4, 2, 0, 2]
i = 2;
count = 2;
PC = immediately after the count++ statement.

(f) *See (a)*

`oddOrPos()` **Solution:**

(a) *The if-test needs to take account of negative values (positive odd numbers are taken care of by the second test):*
`if (x[i]%2 == -1 || x[i] > 0)`

(b) `x` *must be either null or empty. All other inputs result in the fault being executed. We give the empty case here.*

Input:	*x = []*
Expected Output:	*0*
Actual Output:	*0*

(c) *Any nonempty* `x` *with only non-negative elements works, because the first part of the compound if-test is not necessary unless the value is negative.*

Input:	*x = [1, 2, 3]*
Expected Output:	*2*
Actual Output:	*2*

(d) *For this particular program, every input that results in error also results in failure. The reason is that error states are not repairable by subsequent processing. If there is a negative value in* `x`*, all subsequent states (after processing the negative value) will be error states no matter what else is in* `x`*.*

(e)

Input:	*x = [-3, -2, 0, 1, 4]*
Expected Output:	*3*
Actual Output:	*2*

First Error State:
x = [-3, -2, 0, 1, 4]
i = 0;
count = 0;
PC = at end of `if` *statement, instead of just before* `count++`

(f) *See (a)*

7. Answer questions a-d for the graph defined by the following sets:
   - $N = \{0,\ 1,\ 2\}$
   - $N_0 = \{0\}$
   - $N_f = \{2\}$
   - $E = \{(0,1),\ (0,2),\ (1,0),\ (1,2),\ (2,0)\}$

   Also consider the following (candidate) paths:
   - $p_0 = [0,\ 1,\ 2,\ 0]$
   - $p_1 = [0,\ 2,\ 0,\ 1,\ 2]$
   - $p_2 = [0,\ 1,\ 2,\ 0,\ 1,\ 0,\ 2]$
   - $p_3 = [1,\ 2,\ 0,\ 2]$
   - $p_4 = [0,\ 1,\ 2,\ 1,\ 2]$

   (a) Which of the listed paths are test paths? Explain the problem with any path that is not a test path.

   Solution:
   Answer: Only p_1 and p_2 are test paths. p_0 fails to terminate at a final node. p_3 fails to start at an initial node. p_4 includes an edge that does not exist in the graph.

   (b) List the eight test requirements for Edge-Pair Coverage (only the length two subpaths).

   Solution:
   Answer: The edge pairs are:

   $\{\ [n_0, n_1, n_0],\ [n_0, n_1, n_2],\ [n_0, n_2, n_0],\ [n_1, n_0, n_1],\ [n_1, n_0, n_2],\ [n_1, n_2, n_0],\ [n_2, n_0, n_1],\ [n_2, n_0, n_2]\ \}$

   (c) Does the set of **test** paths (part a) above satisfy Edge-Pair Coverage? If not, identify what is missing.

   Solution:
   Answer: No. Neither p_1 nor p_2 tours either of the following edge-pairs:

   $\{\ [n_1, n_0, n_1],\ [n_2, n_0, n_2]\ \}$

   As discussed in (a), the remaining candidate paths are not test paths.

   (d) Consider the prime path $[n_2,\ n_0,\ n_2]$ and path p_2. Does p_2 tour the prime path directly? With a sidetrip?

   Solution:
   Answer: No, p_2 does not directly tour the prime path. However, p_2 does tour the prime path with the sidetrip $[n_0, n_1, n_0]$.

8. Design and implement a program that will compute all prime paths in a graph, then derive test paths to tour the prime paths. Although the user interface can be arbitrarily complicated, the simplest version will be to accept a graph as input by reading a list of nodes, initial nodes, final nodes, and edges.

   Instructor Solution Only

Exercises, Section 2.2.1

1. Redefine *Edge Coverage* in the standard way (see the discussion for *Node Coverage*).

   Instructor Solution Only

2. Redefine *Complete Path Coverage* in the standard way (see the discussion for *Node Coverage*).

   Instructor Solution Only

3. Subsumption has a significant weakness. Suppose criterion C_{strong} subsumes criterion C_{weak} and that test set T_{strong} satisfies C_{strong} and test set T_{weak} satisfies C_{weak}. It is not necessarily the case that T_{weak} is a subset of T_{strong}. It is also not necessarily the case that T_{strong} reveals a fault if T_{weak} reveals a fault. Explain these facts.

   Instructor Solution Only

4. Answer questions a-d for the graph defined by the following sets:
   - $N = \{1,\ 2,\ 3,\ 4\}$
   - $N_0 = \{1\}$
   - $N_f = \{4\}$
   - $E = \{(1,2),\ (2,3),\ (3,2),\ (2,4)\}$

   (a) Draw the graph.

   Solution:
   See the graph tool at `http://www.introsoftwaretesting.com`

   (b) List test paths that achieve Node Coverage, but not Edge Coverage.

   Solution:
   For this program, this is not possible. All test paths must begin at node 1, visit node 2, and, eventually, end at node 4. Any test path that visits node 3 also visits both edge $(2,3)$ *and edge* $(3,2)$.

   (c) List test paths that achieve Edge Coverage, but not Edge Pair Coverage.

   Solution:
   $T = \{[1,2,3,2,4]\}$
   Note that the edge pair $[3,2,3]$ *is not toured by the single test path given.*

   (d) List test paths that achieve Edge Pair Coverage.

   Solution:
   $T = \{[1,2,3,2,3,2,4]\}$

5. Answer questions a-g for the graph defined by the following sets:
   - $N = \{1,2,3,4,5,6,7\}$
   - $N_0 = \{1\}$
   - $N_f = \{7\}$
   - $E = \{(1,2),(1,7),(2,3),(2,4),(3,2),(4,5),(4,6),(5,6),(6,1)\}$

   Also consider the following (candidate) test paths:

- $t_0 = [1,\ 2,\ 4,\ 5,\ 6,\ 1,\ 7]$
- $t_1 = [1,\ 2,\ 3,\ 2,\ 4,\ 6,\ 1,\ 7]$

(a) Draw the graph.

Solution:
See the graph tool at `http://www.introsoftwaretesting.com`

(b) List the test requirements for Edge-Pair Coverage. (Hint: You should get 12 requirements of length 2).

Instructor Solution Only

(c) Does the given set of test paths satisfy Edge-Pair Coverage? If not, identify what is missing.

Instructor Solution Only

(d) Consider the simple path **[3, 2, 4, 5, 6]** and test path **[1, 2, 3, 2, 4, 6, 1, 2, 4, 5, 6, 1, 7]**. Does the test path tour the simple path directly? With a sidetrip? If so, identify the sidetrip.

Instructor Solution Only

(e) List the test requirements for Node Coverage, Edge Coverage, and Prime Path Coverage on the graph.

Instructor Solution Only

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

Instructor Solution Only

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

Instructor Solution Only

6. Answer questions a-c for the graph in Figure 2.2.

(a) Enumerate the test requirements for Node Coverage, Edge Coverage, and Prime Path Coverage on the graph.

Solution:
$TR_{NC} = \{n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$
$TR_{EC} = \{(n_0, n_3),\ (n_0, n_4),\ (n_1, n_4),\ (n_2, n_5),\ (n_2, n_6),\ (n_3, n_7),\ (n_4, n_7),\ (n_4, n_8),\ (n_5, n_1),\ (n_5, n_9),\ (n_6, n_9),\ (n_8, n_5)\ \}$
$TR_{PPC} = \{\ [n_0, n_3, n_7],\ [n_0, n_4, n_7],\ [n_0, n_4, n_8, n_5, n_1],\ [n_0, n_4, n_8, n_5, n_9],\ [n_1, n_4, n_8, n_5, n_1],\ [n_1, n_4, n_8, n_5, n_9],\ [n_2, n_5, n_1, n_4, n_7],\ [n_2, n_5, n_1, n_4, n_8],\ [n_2, n_5, n_9],\ [n_2, n_6, n_9],\ [n_4, n_8, n_5, n_1, n_4],\ [n_5, n_1, n_4, n_8, n_5],\ [n_8, n_5, n_1, n_4, n_7],\ [n_8, n_5, n_1, n_4, n_8]\ \}$

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

Instructor Solution Only

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

Instructor Solution Only

Exercises, Section 2.2.3

1. Below are four graphs, each of which is defined by the sets of nodes, initial nodes, final nodes, edges, and defs and uses. Each graph also contains a collection of test paths. Answer the following questions about each graph.

Graph I.
$N = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}$
$N_0 = \{0\}$
$N_f = \{7\}$
$E = \{(0,1),\ (1,2),\ (1,7),\ (2,3),\ (2,4),\ (3,2),$
$(4,5),\ (4,6),\ (5,6),\ (6,1)\}$
$def(0) = def(3) = use(5) = use(7) = \{x\}$

Test Paths:
$t1 = [0,\ 1,\ 7]$
$t2 = [0,\ 1,\ 2,\ 4,\ 6,\ 1,\ 7]$
$t3 = [0,\ 1,\ 2,\ 4,\ 5,\ 6,\ 1,\ 7]$
$t4 = [0,\ 1,\ 2,\ 3,\ 2,\ 4,\ 6,\ 1,\ 7]$
$t5 = [0,\ 1,\ 2,\ 3,\ 2,\ 3,\ 2,\ 4,\ 5,\ 6,\ 1,\ 7]$
$t6 = [0,\ 1,\ 2,\ 3,\ 2,\ 4,\ 6,\ 1,\ 2,\ 4,\ 5,\ 6,\ 1,\ 7]$

Graph II.
$N = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$
$N_0 = \{1\}$
$N_f = \{6\}$
$E = \{(1,2),\ (2,3),\ (2,6),\ (3,4),\ (3,5),\ (4,5),\ (5,2)\}$
$def(x) = \{1,3\}$
$use(x) = \{3,6\}$ // Assume the use of x in 3 precedes the def
Test Paths:
$t1 = [1,\ 2,\ 6]$
$t2 = [1,\ 2,\ 3,\ 4,\ 5,\ 2,\ 3,\ 5,\ 2,\ 6]$
$t3 = [1,\ 2,\ 3,\ 5,\ 2,\ 3,\ 4,\ 5,\ 2,\ 6]$
$t4 = [1,\ 2,\ 3,\ 5,\ 2,\ 6]$

Graph III.
$N = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$
$N_0 = \{1\}$
$N_f = \{6\}$
$E = \{(1,2),\ (2,3),\ (3,4),\ (3,5),\ (4,5),\ (5,2),\ (2,6)\}$
$def(x) = \{1,\ 4\}$
$use(x) = \{3,\ 5,\ 6\}$

Test Paths:
$t_1 = [1,\ 2,\ 3,\ 5,\ 2,\ 6]$
$t_2 = [1,\ 2,\ 3,\ 4,\ 5,\ 2,\ 6]$

Graph IV.
$N = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$
$N_0 = \{1\}$
$N_f = \{6\}$
$E = \{(1,2),\ (2,3),\ (2,6),\ (3,4),\ (3,5),\ (4,5),\ (5,2)\}$
$def(x) = \{1,\ 5\}$
$use(x) = \{5,\ 6\}$ // Assume the use of x in 5 precedes the def
Test Paths:
$t1 = [1,\ 2,\ 6]$
$t2 = [1,\ 2,\ 3,\ 4,\ 5,\ 2,\ 3,\ 5,\ 2,\ 6]$
$t3 = [1,\ 2,\ 3,\ 5,\ 2,\ 3,\ 4,\ 5,\ 2,\ 6]$

   (a) Draw the graph.
   (b) List all of the du-paths with respect to x. (Note: Include all du-paths, even those that are subpaths of some other du-path).
   (c) For each test path, determine which du-paths that test path tours. For this part of the exercise, you should consider both direct touring and sidetrips. Hint: A table is a convenient format for describing this relationship.
   (d) List a minimal test set that satisfies *all defs* coverage with respect to x. (Direct tours only.) Use the given test paths.
   (e) List a minimal test set that satisfies *all uses* coverage with respect to x. (Direct tours only.) Use the given test paths.
   (f) List a minimal test set that satisfies *all du-paths* coverage with respect to x. (Direct tours only.) Use the given test paths.

Solution:
Solution for Graph I:

(a) *See the graph tool at* `www.introsoftwaretesting.com`
(b) *x has 5 du-paths, as enumerated below:*

i	*[0, 1, 7]*
ii	*[0, 1, 2, 4, 5]*
iii	*[3, 2, 4, 5]*
iv	*[3, 2, 4, 6, 1, 7]*
v	*[3, 2, 4, 5, 6, 1, 7]*

(c) *The numbers in the table below correspond to the du-paths in the previous table. The table indicates whether each test path tours each du-path with or without a sidetrip.*

	direct	*w/ sidetrip*
t_1	*i*	
t_2		*i*
t_3	*ii*	*i*
t_4	*iv*	*i*
t_5	*iii, v*	*i, ii*
t_6		*i, ii, iii, iv, v*

(d) *This question has multiple possible answers. Either* t_1 *or* t_3 *can be used to directly tour a path that satisfies all-defs for the def at node 0, and either* t_4 *or* t_5 *can be used to directly tour a path that satisfies all-defs for the def at node 3.*
Possible answers: $\{t_1,\ t_4\}$ *or* $\{t_1,\ t_5\}$ *or* $\{t_3,\ t_4\}$ *or* $\{t_3,\ t_5\}$
(e) *This question only has one possible answer:* $\{t_1,\ t_3,\ t_5\}$
(f) *This question only has one possible answer:* $\{t_1,\ t_3,\ t_4,\ t_5\}$

Solution (Instructor only):
Solution for Graph II:

(a) See the graph tool at `www.introsoftwaretesting.com`

(b) x has 6 du-paths, as enumerated below:

i	*[1, 2, 3]*
ii	*[1, 2, 6]*
iii	*[3, 4, 5, 2, 3]*
iv	*[3, 4, 5, 2, 6]*
v	*[3, 5, 2, 3]*
vi	*[3, 5, 2, 6]*

(c) The numbers in the table below correspond to the du-paths in the previous table. The table indicates whether each test path tours each du-path with or without a sidetrip.

	direct	*w/ sidetrip*
t_1	*ii*	
t_2	*i, iii, vi*	
t_3	*i, iv, v*	
t_4	*i, vi*	

Note that, for example, although t_2 does not tour path (v) with a sidetrip, it does tour path (v) with a detour.

(d) This question has multiple possible answers. All test paths use the def in 1*, and test paths $\{t_2\}$, $\{t_3\}$, $\{t_4\}$ each use the def in* 3*. Possible answers: $\{t_2\}$, $\{t_3\}$, or $\{t_4\}$.*

(e) This question only has two possible answers. $\{t_1\}$ is required for the def in 1 *to reach the use in* 6*. Either t_2 or t_3 is required for the def in* 3 *to reach the use in* 3*. Possible answers: $\{t_1, t_2\}$ or $\{t_1, t_3\}$.*

(f) This question has one possible answer: $\{t_1, t_2, t_3\}$. t_1 is required for path (ii). t_2 is required for path (iii). t_3 is required for path (iv). Since t_1, t_2, and t_3 together tour all six du-paths, t_4 is not needed.

Solution:
Solution for Graph III: Note that this exercise is the same as Graph II, except that the def/use sets are slightly different.

(a) See the graph tool at `www.introsoftwaretesting.com`

(b) x has 6 du-paths, as enumerated below:

i	*[1, 2, 3]*
ii	*[1, 2, 3, 5]*
iii	*[1, 2, 6]*
iv	*[4, 5]*
v	*[4, 5, 2, 3]*
vi	*[4, 5, 2, 6]*

(c) The numbers in the table below correspond to the du-paths in the previous table. The table indicates whether each test path tours each du-path with or without a sidetrip.

	direct	*w/ sidetrip*
t_1	*i, ii*	*iii*
t_2	*i, iv, vi*	

Note that neither t_1 nor t_2 tours du-path (v), either directly or with a sidetrip. Also note that neither t_1 nor t_2 tours du-path (iii), except with a sidetrip.

(d) This question has one possible answer: $\{t_2\}$.

(e) Since the given test set does not have a test path that directly tours from the def in node 1 to the use in node 6, this aspect of the question is unsatisfiable for direct touring. However, if we consider best-effort touring, then test set $\{t_1, t_2\}$ achieves all-uses.

(f) Since the given test set does not have a test path that tours du-path (v), this question is unsatisfiable. To directly tour the given $du - paths$, we would two additional test paths. An example ADUP adequate test set (direct touring) is: $\{t_1, t_2, [1, 2, 6], [1, 2, 3, 4, 5, 2, 3, 5, 2, 6]\}$.

Solution (Instructor only):

Solution for Graph IV: Note that this exercise is the same as Graph II, except that the def/use sets are slightly different.

(a) See the graph tool at `www.introsoftwaretesting.com`

(b) x has 6 du-paths, as enumerated below:

i	*[1, 2, 3, 4, 5]*
ii	*[1, 2, 3, 5]*
iii	*[1, 2, 6]*
iv	*[5, 2, 3, 4, 5]*
v	*[5, 2, 3, 5]*
vi	*[5, 2, 6]*

(c) The numbers in the table below correspond to the du-paths in the previous table. The table indicates whether each test path tours each du-path with or without a sidetrip.

	direct	*w/ sidetrip*
t_1	*iii*	
t_2	*i, v, vi*	
t_3	*ii, iv, vi*	

Note that, for example, although t_2 does not tour path (ii) with a sidetrip, it does tour path (ii) with a detour.

(d) This question has two possible answers: $\{t_2\}$ or $\{t_3\}$.

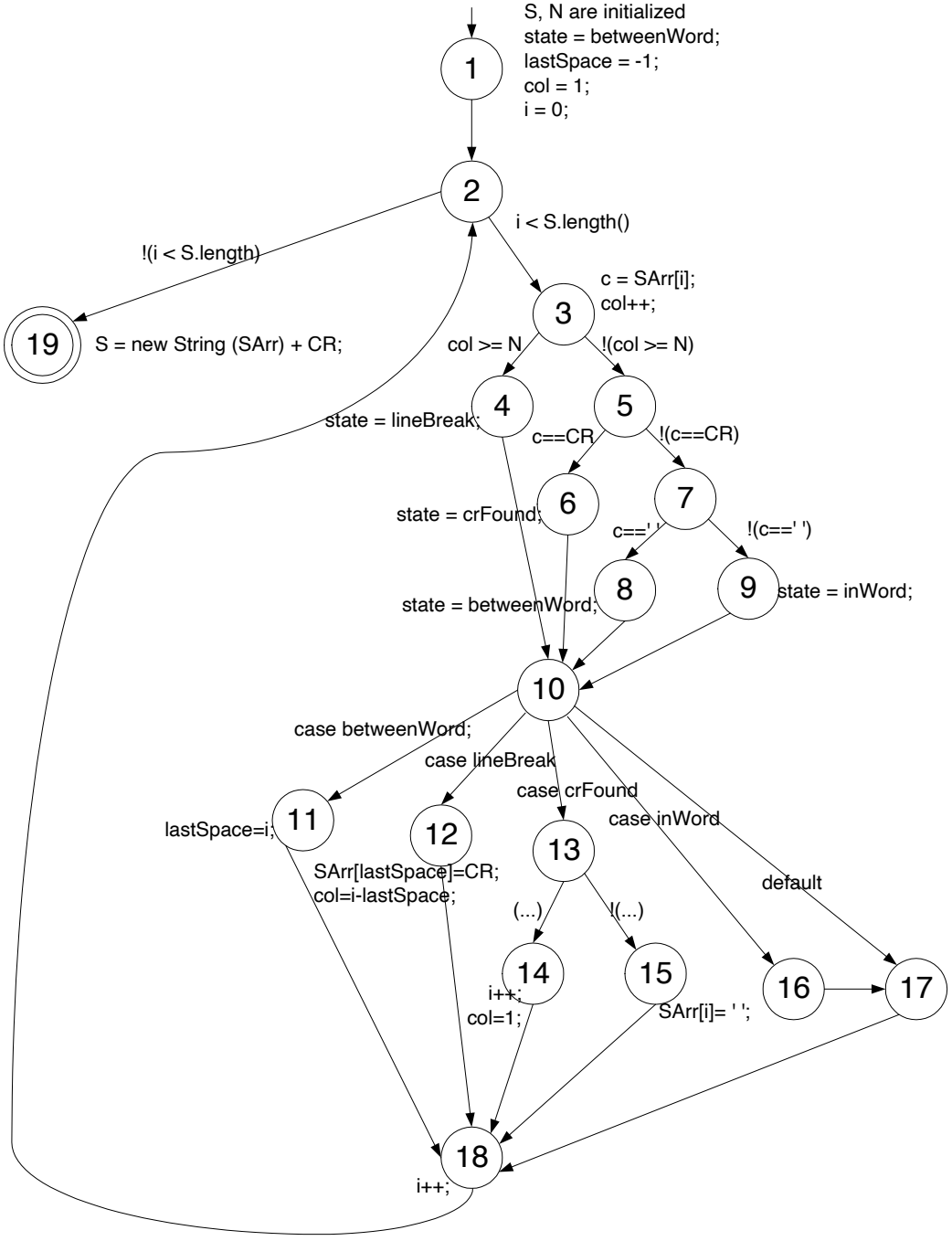
(e) This question two possible answers: $\{t_1, t_2\}$ or $\{t_1, t_3\}$.

(f) This question has one possible answer: $\{t_1, t_2, t_3\}$.

```
 1. /** ***************************************************
 2.  *  Rewraps the string (Similar to the Unix fmt).
 3.  *  Given a string S, eliminate existing CRs and add CRs to the
 4.  *  closest spaces before column N.  Two CRs in a row are considered to
 5.  *  be "hard CRs" and are left alone.
 7. ******************************************************** */
 6.
 8. static final char CR = '\n';
 9. static final int inWord      = 0;
10. static final int betweenWord = 1;
11. static final int lineBreak   = 2;
12. static final int crFound     = 3;
13. static private String fmtRewrap (String S, int N)
14. {
15.    int state = betweenWord;
16.    int lastSpace = -1;
17.    int col = 1;
18.    int i = 0;
19.    char c;
20.
21.    char SArr [] = S.toCharArray();
22.    while (i < S.length())
23.    {
24.       c = SArr[i];
25.       col++;
26.       if (col >= N)
27.          state = lineBreak;
28.       else if (c == CR)
29.          state = crFound;
30.       else if (c == ' ')
31.          state = betweenWord;
32.       else
33.          state = inWord;
34.       switch (state)
35.       {
36.       case betweenWord:
37.          lastSpace = i;
38.          break;
39.
40.       case lineBreak:
41.          SArr [lastSpace] = CR;
42.          col = i-lastSpace;
43.          break;
44.
45.       case crFound:
46.          if (i+1 < S.length() && SArr[i+1] == CR)
47.          {
48.             i++; // Two CRs => hard return
49.             col = 1;
50.          }
51.          else
52.             SArr[i] = ' ';
53.          break;
54.
55.       case inWord:
56.       default:
57.          break;
58.       }  // end switch
59.       i++;
60.    }  // end while
61.    S = new String (SArr) + CR;
62.    return (S);
63. }
```

(a) Draw the control flow graph for the **fmtRewrap()** method.

Solution:



Note that in the `switch` *statement, there is a separate node for the case* `inWord`*, which, due to Java semantics, falls through directly to the* `default` *case.*

(b) For **fmtRewrap()**, find a test case such that the corresponding test path visits the edge that connects the beginning of the **while** statement to the **S = new**

String(SArr) + CR; statement **without** going through the body of the while loop.

Solution:
There is only one test that does this - the empty string `S`*. It doesn't matter what* `N`*, the output line length, is. The resulting path is* $[1, 2, 19]$.

(c) Enumerate the test requirements for Node Coverage, Edge Coverage, and Prime Path Coverage for the graph for **fmtRewrap()**.

Solution:

- *Node Coverage:* { *1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19* }
- *Edge Coverage:* { *(1,2), (2,3), (2,19), (3,4), (3,5), (4,10), (5,6), (5,7), (6,10), (7,8), (7,9), (8,10), (9,10), (10,11), (10,12), (10,13), (10,16), (10,17), (11,18), (12,18), (13,14), (13,15), (14,18), (15,18), (16,17), (17,18), (18,2)* }
- *Prime Path Coverage: There are 403 prime paths for* `fmt`*; we don't list them here. See the graph tool at* `http://www.introsoftwaretesting.com` *for a complete enumeration*
  To get a grasp on why there are so many prime paths, consider the "loop" prime paths starting at node 2. It is possible to take any of 4 paths through the `if else` *statement, and then combine each of these with 6 paths through the case statement and return to node 2, yielding 24 prime paths. For node 3, the analysis is more complex: 4 choices combined with 6 choices to node 2, combined with 2 more choices (3 or 19), for a total of 48 prime paths.*

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

Solution:

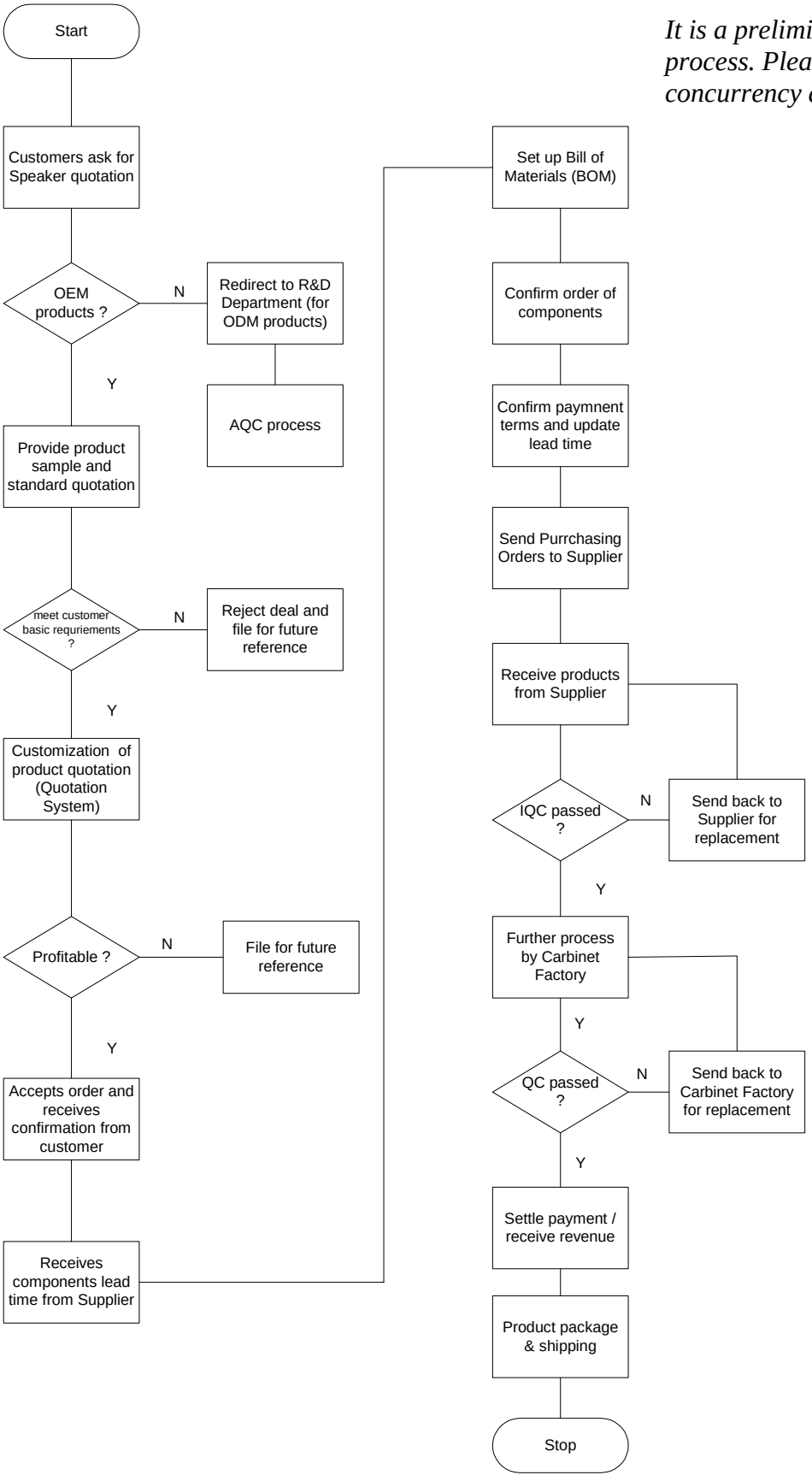
Achieving Node Coverage without also achieving Edge Coverage is not possible for the given graph, since visiting every node implies visiting every edge. Test paths are available online. See the graph tool at `http://www.introsoftwaretesting.com`.

(e) List test paths that achieve Edge Coverage but not prime Path Coverage on the graph.

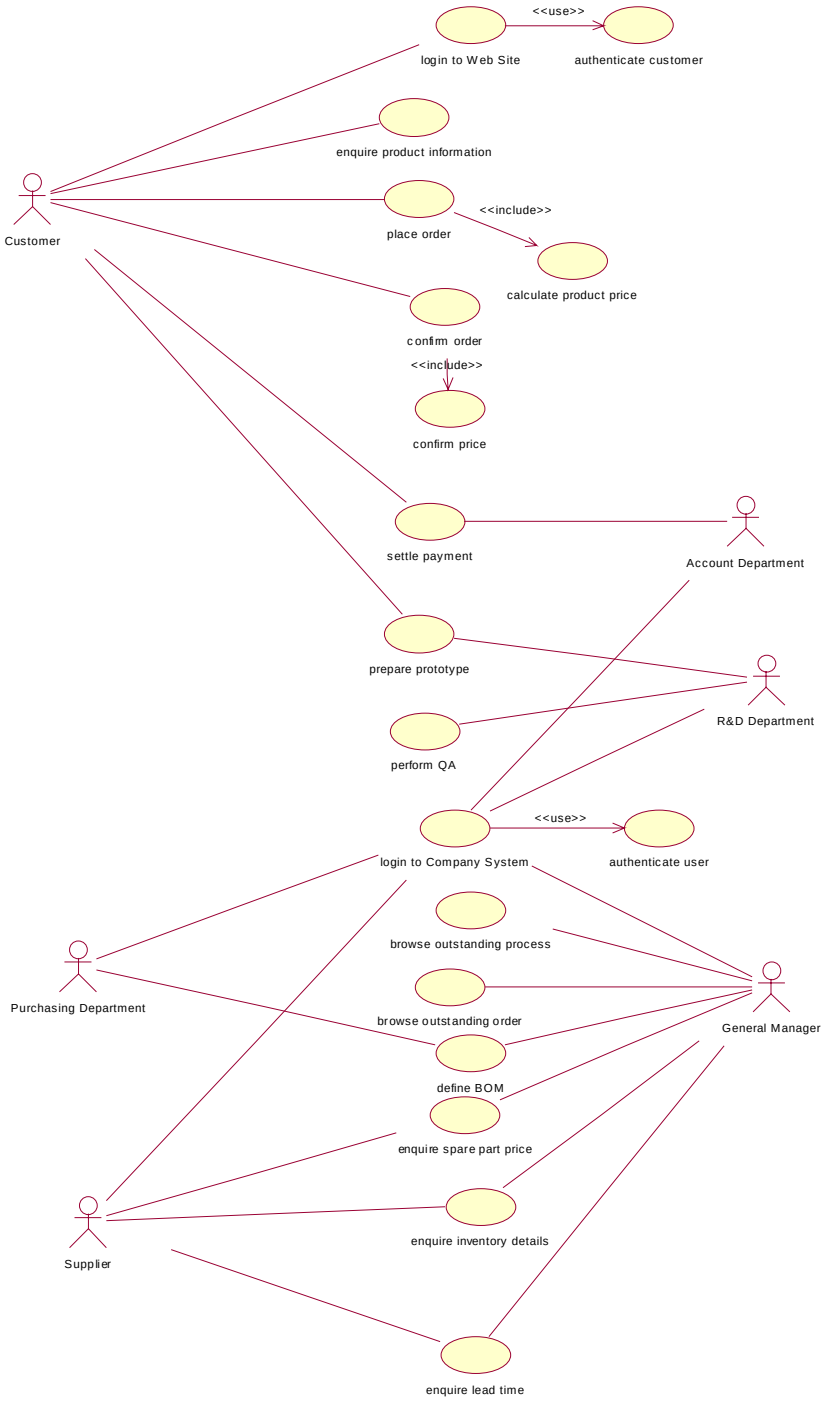
Solution:
This is certainly possible; you can find test paths for Edge Coverage and Prime Path Coverage online. See the graph tool at `http://www.introsoftwaretesting.com`.

7. Use the following method **printPrimes()** for questions a-f below.

It is a preliminary business process. Please refine it with concurrency and swinlanes



3. Use Case Diagram



4. Use Case Model

A. Actor Description

A.1 Customer

A customer represents a person or a Company or some representatives (i.e. a buying agent) within a business organization. Customer makes use of this system to order and enquire for information of our products from our Company by Fax and Internet as well as to send order and settle with payment and invoices.

A.2 Account Department

The Account Department is a division of our Company responsible for checking account balances.

A.3 R & D Department

The R&D department in our Company is responsible for the research and development of new product in the Audio Industry. The department also prepares prototype and Quality Assurance of the final product before delivery to Customers.

A.4 General Manager

General Manager is responsible in overlook of the complete process and monitor the appropriated Departments and Suppliers in carry out their duties. i.e. the Ordering system, planning for the new product development, liaise with Customers on price and product features, liaise with Suppliers on price and delivery, BOM approval etc.

A.5 Purchasing Department

Purchasing Department is responsible for the sourcing and purchasing of parts and products for our Company. The Department calculates of the delivery time and work out schedule and price with the suppliers in according with the BOM prepared by the General Manager.

A.6 Supplier

The Supplier represents a Company or a Factory, which supplies our Company with parts or semi-finished product in a finishing product. The suppliers also provide us with services in Original Equipment Manufacturer (OEM) according to our requirements.

B. Use Case Description

B.1 Enquire product information

Purpose: Allow Customers to visit our Web Site to view our latest Audio products, they can request for sample and pricing information

Actor: Customer

Preconditions: Not Applicable

Flow of events:

1. Visit our Company web site.
2. Click on product information enquiry button on Web.
3. Customer can enquire specific product by clicking the highlighted product
4. Customer could contact us by entering the contact information on the Web.
5. Sales representatives from our Company would contact corresponding potential customers for their ordering / pricing or sample requests.

B.2 Place order

Purpose: Allow Customers to place order from our Web Site.

Actor: Customer

Preconditions: Not Applicable

Flow of events:

1. Visit our Company web site.
2. Click on Product Ordering button on Web.
3. Select the products (highlighted product codes) they want to order.
4. Enter quantity and date of delivery on the corresponding columns.
5. For existing customers, enter customer number and for new customers, enter their contact information. Customer number will be generated for new customers who have placed order successfully.
6. Sales representatives from our Company would contact corresponding potential customers to confirm order procedure.

B.3 Calculate product price

Purpose: Allow Customers to calculate product price from our Web Site.

Actor: Customer

Preconditions: Not Applicable

Flow of events:

1. Visit our Company web site.
2. Click on Product Ordering button on Web.
3. Select the products (highlighted product codes) they want to order.
4. Customer calculates the product prices interactively through the Web.

B.4 Confirm order

Purpose: Allow Customers to confirm order from our Web Site.

Actor: Customer

Preconditions: Customers with orders placed

Flow of events:

1. Visit our Company web site.
2. Click on Product Ordering button on Web.
3. Select the products (highlighted product codes) they want to order.
4. Enter quantity and date of delivery on the corresponding columns.
5. For existing customers, enter customer number and for new customers, enter their contact information. Customer number will be generated for new customers who have placed order successfully.
6. Click Confirm Order button on Web.
7. Acknowledgement will be sent to Customer by e-mail.

B.5 Confirm price

Purpose: Allow Customers to confirm price from our Web Site.

Actor: Customer

Preconditions: Customers with orders placed

Flow of events:

1. Visit our Company web site.
2. Click on Product Ordering button on Web.
3. Select the products (highlighted product codes) they want to order.
4. Enter quantity and date of delivery on the corresponding columns.
5. For existing customers, enter customer number and for new customers, enter their contact information. Customer number will be generated for new

customers who have placed order successfully.
6. Click Confirm Order button on Web.
7. Acknowledgement will be sent to Customer by e-mail.

B.6 Settle Payment

Purpose: Allow Customers to enquire payment settlement from our Web Site.

Actor: Customer, Account Department

Preconditions: Customers with sales history

Flow of events:

1. Visit our Company web site.
2. Enter Customer Number and passwor.
3. If authentication is unsuccessful, no further access to the system is allowed.
4. If authentication is successful click on Payment Settlement Enquiry button on Web.
5. The payment transactions will be displayed by transaction dated in descending order.
6. The payment types include cash payment, check payment and credit card payment. However, credit card payment should not exceed credit limit.

B.7 Prepare prototype

Purpose: Allow Customers to request product prototypes from our Web Site.

Actor: Customer, R & D Department

Preconditions: Must be registered Customers (i.e. with valid Customer Number)

Flow of events:

1. Visit our Company web site
2. Click on Product Ordering button on Web.
3. Select the products (highlighted product codes) they want to order.
4. Enter quantity and date of delivery on the corresponding columns.
5. For existing customers, enter customer number and for new customers, enter their contact information. Customer number will be generated for new customers who have placed order successfully.
6. Click Confirm Order button on Web.
7. Acknowledgement will be sent to Customer by e-mail.