EE 360T/382V Software Testing Midterm-1 Solutions

Not: Solutions (Except the true/false question) are posted according to the order for version-a.
True/false question answers are available for both version-a and version-b.

Question 1 – True/Flase

For Version A:

1) F
2) F
3) T
4) F
5) T

For Version B:

1) T
2) F
3) T
4) F
5) F

Question 2 –Sorted Linked List

(a) [2 points] Write a JUnit test public void t1() that adds the null literal into a list with one node with element "1", and checks if SortedList.add() works as specified.

```
@Test(expected=IllegalArgumentException.class)
public void t1() {
    SortedList l = new SortedList();
    Node n1 = new Node(1);
    l.header = n1;
    l.size = 1;
    l.add(null);
}
```

or

```java
@Test
public void t1() {
    SortedList l = new SortedList();
    Node n1 = new Node(1);
    l.header = n1;
    l.size = 1;
    try {
        l.add(null);
    } catch (IllegalArgumentException e) {
        return;
    }
    fail(); // Can be assertTrue(false) or assertFalse(true)
}
```

(b) [4 points] Write a JUnit test public void t2() that adds a node with element “2” into a list with one node with element “1”, and checks if SortedList.add() works as specified. Please check values for all declared fields of list and node objects.

```java
@Test
public void t2() {
    SortedList l = new SortedList();
    Node n1 = new Node(1);
    Node n2 = new Node(2);
    l.header = n1;
    l.size = 1;
    l.add(n2);
    assertEquals(l.header, n1);
    assertEquals(n1.next, n2);
    assertNull(n2.next);
    assertEquals(n1.elem, 1);
    assertEquals(n2.elem, 2);
    assertEquals(2, l.size);
}
```

(c) [6 points] Find a fault in SortedList.add() method, briefly describe the problem, and write a Junit test public void t3() that fails due to the fault. Make sure your test passes once the fault is fixed.

When a node is to be inserted to the first position in the list, line 9 will throw a NullPointerException because n1 is null.

```
@Test
public void t3() {
    SortedList l = new SortedList();
    Node n1 = new Node(1);
    l.add(n1);
}
```

(d) [4 points] Describe how to fix the fault in SortedList.add(). Please be specific, e.g., replace line 1 with …, or insert … after line 1.

Replace line 9 with:

```
if(n1 == null) {
    header = node;
} else {
    n1.next = node;
}
```

(e) [6 points] Implement the method SortedList.repOK() as specified. You can import java.util libraries as needed.

```
public boolean repOK() {
    Set<Node> set = new HashSet<>();
    Node n = header;
    while(n != null) {
        if(!set.add(n)) {
            return false;
        }
        if(n.next != null && n.next.elem < n.elem) {
            return false;
        }
        n = n.next;
    }
    return set.size() == size;
}
```

Question 3 – Fault/Failure

(a)

line 6: change continue to break,

other solutions are accepted, such as:

change line 5 to return i
change line 6 to return j
change iteration from arr.length-1 to 0
etc.

(b) any case that the integer array does not contain the target

```
assertEquals(find(new int[]{0,0}, 1), -1);
```

(c) any case that the target appears only once in the integer array

```
assertEquals(find(new int[]{0,1}, 0), 0);
```

(d) any case that the target appears multiple times in the integer array and

```
assertEquals(find(new int[]{0,0,1}, 0), 0);
```

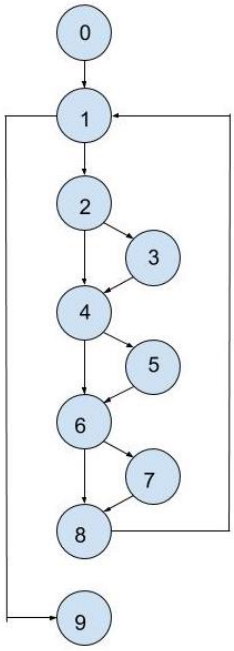
Question 4 – Finding the prime paths

Size 6:	1-2-3-4-5-6
	1-2-3-4-5-7
Size 5:	1-2-3-4-7
	1-3-4-5-6
	1-3-4-5-7
Size 4:	1-3-4-7
	4-5-6-4
	5-6-4-7
	5-6-4-5
	6-4-5-6
	6-4-5-7

Total: 11 prime paths

Question 5 – Sequencing Constraint

(a) [4 points] Draw a control flow graph (CFG) for keepPositiveEvenNumbers() and give each node a unique label. In addition, label nodes where remove() or next() are invoked.



remove(): 3, 5, 7
next(): 2

(b) [4 points] Assume we know the graph structure from part (a), which includes the nodes where remove() or next() are invoked, but do not know the other source-code-level details. List all execution paths with at most 1 loop iteration such that the sequencing constraints of Iterator may be violated. Among the paths that may violate a sequencing constraint, identify all that are infeasible according to the given source-code or state that no such infeasible path exists.

0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 1 -> 9
Infeasible. (3, 5) cannot both be executed, neither can (5, 7).
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 8 -> 1 -> 9
Infeasible. (3, 5) cannot both be executed.
0 -> 1 -> 2 -> 3 -> 4 -> 6 -> 7 -> 8 -> 1 -> 9
Feasible.
0 -> 1 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 -> 1 -> 9
Infeasible. (5, 7) cannot both be executed.

(c) [4 points] Write a JUnit test that executes a path in keepPositiveEvenNumbers() that violates the sequencing constraints of class Iterator. You do not need to write assertions for this test.

```
@org.junit.Test
public void violateSequencingConstraint() {
    java.util.List<Integer> list = new java.util.LinkedList<Integer>();
    list.add(-1);
    getPositiveEvenNumbers(list);
}
```

Question 6 – Prime numbers

(a)

```
@Test public void t0(){
        PrimeNums pm = new PrimeNums();
        assertEquals(3, pm.augment());
        assertTrue(pm.repOk());
        assertEquals(5, pm.augment());
        assertTrue(pm.repOk());
        assertEquals(7, pm.augment());
        assertTrue(pm.repOk());
}
```

(b) There are a lot of ways to write this method, here are some for instance.

```
public int augment() {
        Node node = head;
        int i = tail.elem + 1;
        while (node != null ) {
                if (i % node.elem == 0) {
                        node = head;
                        i++;
                } else {
                        node = node.next;
                }
        }
        Node newNode = new Node();
        newNode.elem = i;
        newNode.next = null;                                    // not necessary
        tail.next = newNode;
        tail = newNode; size++;
        return i;
}
```

```
public int augment() {
	Node node = head;
	int i = tail.elem + 1;
	for (; ; i++ ) {
		while (node != null && i % node.elem != 0) {
			node = node.next;
		}
		if (node == null) break;		// found the next prime else node = head;
	}
	Node newNode = new Node();
	newNode.elem = i;
	newNode.next = null; // not necessary
	tail.next = newNode;
	tail = newNode;
	size++;
	return i;
}
```

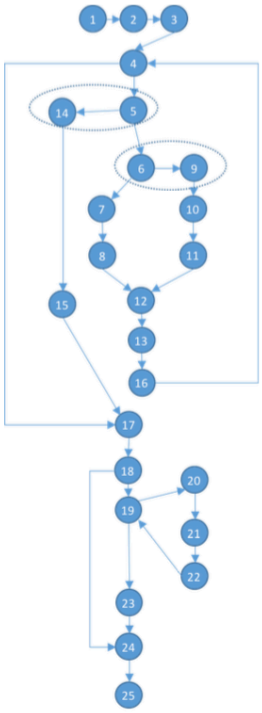
```
public int augment() {
	Node node = head;
	int newPrime = tail.elem;
	while (true) {
		newPrime++;
		int divisor = 2;
		for (;divisor < newPrime; divisor++ ){
			if (newPrime % divisor == 0) break;
		}
		if (divisor == newPrime) break;		// next prime found
	}
	Node newNode = new Node();
	newNode.elem = newPrime;
	newNode.next = null;				// not necessary
	tail.next = newNode;
	tail = newNode;
	size++;
	return newPrime;

}
```

```
public int augment() {
        int newPrime = tail.elem + 1;
        while (!isPrime(newPrime)) newPrime++;        // for(;!isPrime(newPrime);i++);
        Node newNode = new Node();
        newNode.elem = newPrime;
        newNode.next = null;
        tail.next = newNode;
        tail = newNode;
        size++;
        return newPrime;
}

// checks whether the input value is a prime number.
private boolean isPrime(int val) {
        for (int i = 2; i < val; i++) { // val/2; sqrt(val);
                if (val % i == 0) return false;
        }
   return true;
}
```

Question 7 – du/path



use def

```
    public void merge(int[] nums1, int m, int[] nums2, int n) {
/* 1*/      int p1 = m - 1;
/* 2*/      int p2 = n - 1;
/* 3*/      int i = m + n -1;
/* 4*/      while (i >= 0){
/* 5*/          if (p1 >= 0 && p2 >= 0){
/* 6*/              if (nums1[p1] > nums2[p2]){
/* 7*/                  nums1[i] = nums1[p1];
/* 8*/                  p1--;
/* 9*/              } else {
/*10*/                  nums1[i] = nums2[p2];
/*11*/                  p2--;
/*12*/              }
/*13*/              i--;          i = i - 1;
/*14*/          } else {
/*15*/              break;
/*16*/          }
/*17*/      }
/*18*/      if (p1 < 0) {
/*19*/          while (p2 >= 0){
/*20*/              nums1[i] = nums2[p2];
/*21*/              p2--;
/*22*/              i--;
/*23*/          }
/*24*/      }
/*25*/  }
```

1. 3, 13, 22
2. 4, 7, 10, 13, 20, 22
3. 1) 13, (16,) 4, 5, (14,) 15, (17,) 18, 19, 20, 21, 22
   2) 1, 2, 3, 4, 5, 6, 7, 8, (12,) [13, (16, 17,) 4, 5, (14,) 15, (17,) 18, 19, 20, 21, 22], (23,) 19, (23, 24, 25)
   or 1, 2, 3, 4, 5, 6, (9,) 10, 11, (12,) [13, (16, 17,) 4, 5, (14,) 15, (17,) 18, 19, 20, 21, 22], (23,) 19, (23, 24, 25)

   lines in () is ignorable, lines in [] is the sub-path the same as in 1).