DD2459: Software Reliability, sofRel22

Lab 1: White-box Testing
Answer all 4 questions.

Introduction:

The *triangle program* is a famous testing problem that originated in Myers classical 1979 textbook on testing. It has appeared in many books and papers since, as it is often a good benchmark for new ideas about testing. The program requirement is defined as follows:

"*The program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral*" (Myers, page 1)

We need to recall some facts from elementary geometry:

1. A *triangle* is a polygon with three sides.

2. The *vertices* of a triangle must not be in a straight line.

3. An *equilateral triangle* has three sides of equal length.

4. An *isosceles triangle* has two sides of equal length.

5. A *scalene triangle* has three sides of different lengths.

The **Triangle Test algorithm** below (hopefully) implements the requirement defined above. Note below that | is the *eager* or *sequential or* operation (aka *classical Boolean disjunction*) and & is the *eager* or *sequential and* operation (aka *classical Boolean conjunction*).

```
enumeration Kind = { scalene, isosceles, equilateral, notriangle,
badside } // a data type definition

Kind triangleTest( s1, s2, s3 : int ) {
      if s1 <= 0 | s2 <= 0 | s3 <= 0
      then return badside
      else
            if s1+s2 <= s3 | s2+s3 <= s1 | s1+s3 <= s2
            then return notriangle
            else
                  if s1==s2 & s2==s3
                  then
                        return equilateral
                  else
                        if s1==s2 | s2==s3 | s1==s3
                        then
                              return isosceles
```

```
                    else
                         return scalene
}
```

Question 1. Draw a condensation graph for the Triangle Test algorithm.

In this exercise, you will write out test requirements as paths through this condensation graph to achieve different levels of control flow coverage. **Make sure to introduce a systematic naming convention for (a) your requirements and (b) your test cases, such as the one used below.**

> **Worked Example: NC TR1:** n_4
>
> is a test requirement for control flow coverage (i.e. a list of node names) that specifies to cover node n_4 in a condensation graph for Algorithm 1, attempting to achieve node coverage (NC).
>
> A test case (i.e. an assignment of values to the program input variables) that satisfies requirement NC TR1 would be
> **NC TC1**: S1 = 1, s2 = 1, s3 = 1.

1.1 (a) Write a set of test requirements that achieve full **node coverage** (NC) for the Triangle Test algorithm.
(b) Write out a minimized set of test cases satisfying the requirements of (a).

1.2. (a) Write out a set of test requirements that achieve full **edge coverage** (EC) for the Triangle Test algorithm.
(b) Write out a minimized corresponding set of test cases.
(c) Why are node coverage and edge coverage the same in this example? Carefully explain your reasoning about this fact.

(Exercise continues on the next page.)

Question 2. In this exercise, you will write out test requirements as logical constraints on the input variable values s1, s2 and s3 to achieve different levels of logic coverage.

Worked Example: **PC TR1**: s1 <= 0 | s2 <= 0 | s3 <= 0

is a test requirement for logic coverage (i.e. a constraint on the input variables of the program) that makes a predicate at a node (which node?) in a condensation graph for Question 1, *true*, in order to achieve full predicate coverage (PC).

Then you must write out a test case that satisfies each requirement. If you can minimize the set of test cases by eliminating redundant test cases that is a (locally) optimal solution. A test case satisfying requirement PC TR1 might be:

PC TC1: s1 = 0, s2 = 0, s3 = 0

which satisfies this test requirement at a *boundary*.

2.1. (a) Write out a set of test requirements that achieve full **predicate coverage** (PC) for the Triangle test algorithm 1. (Recall that non-distributive predicate coverage is sufficient here.) Write the corresponding set of test cases.

(b) Looking back on your answers to 1.1.(b) node coverage (NC) and 2.1.(a) predicate coverage (PC) are these always the same for every condensation graph?

(c) Can you modify your condensation graph for Question 1 in some simple way so that predicate coverage PC and node coverage NC are **not** the same. You do **not** have to preserve the functionality of the program. Verify that your answer is correct by writing out corresponding test suites for your PC and NC requirements that are different.

2.2. (a) Write out a set of test requirements that achieve full **clause coverage** (CC) for the Triangle Test Algorithm, using your condensation graph model.

(b) Write out a corresponding set of test cases.

2.3. (a) Write out a set of test requirements that achieve full **restricted active clause coverage** (RACC) (also known as *unique cause MCDC*) for the Triangle Test Algorithm, using your condensation graph model.

(b) Write out a corresponding set of test cases.

(Exercise continues on the next page.)

Question 3. Consider the following piece of code:

```
x = x+1;

while ( x < -100 | x > 100) {

     if (x < -100) then { x = x+1; } else

          if (x > 100) then { x = x-1; }

}

return x;
```

You can assume that `x:int` is the single input variable to the above program, and that | is the "*lazy or*" operation

(a) Draw a condensation graph for this code.
(b) Define a <u>minimal</u> set TR of test requirements on the input variable x that would achieve full (100%) node coverage for this program. Carefully explain why your test requirement set is actually minimal.

(c) Produce a set TC of test cases that satisfy your test requirements for TR in Part 3.(b).

(d) Would predicate coverage yield a better test suite than your answer to 3.(c)? Motivate your answer.

PTO.

Question 4. Self-Assessment

For each of the five sets of test cases you have produced in Questions 1 and 2 (i.e. for each of the five coverage models NC, EC, PC, CC, RACC) , answer the following 14 self assessment questions. For each coverage model, score 1 point for a requirement that is satisfied (maximum possible is 12 points).

Draw up a table that compares the total score achieved for each of the 5 coverage models. Which coverage model achieves the highest score in your table? What does your table say about the coverage models?

(Exercise continues on the next page.)

1. Do you have a test case that represents a valid scalene triangle?

2. Do you have a test case that represents a valid equilateral triangle?

3. Do you have a test case that represents a valid isosceles triangle?

4. Do you have at least three test cases that represent valid isosceles triangles such that you have tried all three permutations of two equal sides?

5. Do you have a test case in which one side has a zero value?

6. Do you have a test case in which one side has a negative value?

7. Do you have a test case with three integers such that the sum of two is equal to the third?

8. Do you have at least three test cases in category 7 such that you have tried all three permutations where the length of one side is equal to the sum of the lengths of the other two sides?

9. Do you have a test case with three integers greater than zero such that the sum of two numbers is less than the third?

10. Do you have at least three test cases in category 9 such that you have tried all three permutations

11. Do you have a test case in which all sides are zero?

12. Do you have at least one test case specifying non-integer values or does this not make sense?

13. Do you have at least one test case specifying the wrong number of values (2 or less, four or more) or does this not make sense?

14. For each test case, did you specify the expected output from the program in addition to the input values?

Reference: G.J. Myers, *The Art of Software Testing*, John Wiley and Sons, 1979.