DD2459: Software Reliability

Lab 1: White-box Testing the Triangle Example
Version 1.2 2012-01-30

Introduction:
The _triangle program_ is a well-known 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 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_" (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.

We can develop pseudocode for a solution (or a model!) that implements the logic of the problem, as defined above.

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

method TriangleTest( s1, s2, s3 : integer ) : Kind
begin
     if s1 <= 0 or s2 <= 0 or s3 <= 0
     then return badside
     else
          if s1+s2 <= s3 or s2+s3 <= s1 or s1+s3 <= s2
          then return notriangle
          else
               if s1==s2 & s2== s3
               then
                    return equilateral
               else
                    if s1==s2 or s2==s3 or s1==s3
                    then
                         return isosceles
                    else
                         return scalene
end
```

Algorithm 1

Exercises.

1. Draw a condensation graph for Algorithm 1.

In this exercise, you will write out <u>test requirements as</u> <u>paths</u> through this condensation graph to achieve different levels of <u>control flow coverage</u>.

> **Example: NC TR1:** n_1, n_2, n_3, n_4
>
> is a test requirement that attempts to cover 4 nodes (which four?) in a condensation graph for Algorithm 1, attempting to achieve node coverage.

1.1 (a) Write a set of <u>test requirements</u> that achieve **node coverage** (NC) for Algorithm 1.
(b) Write out a minimized set of <u>test cases</u> satisfying the requirements of (a).

1.2. (a) Write out a set of test requirements that achieve **edge coverage** (EC) for Algorithm 1.
(b) Write out a minimized corresponding set of test cases.
(c) Why are node coverage and edge coverage the same in this example?

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

> **Example: PC TR1**: s1 <= 0 │ s2 <= 0 │ s3 <= 0
>
> is a test requirement that makes a predicate (which?) in a condensation graph for Question 1, *true*, attempting to achieve predicate coverage.
>
> 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 **predicate coverage** (PC) for Algorithm 1. (Recall that non-distributive predicate coverage is sufficient here.)
(b) Can you modify the condensation graph in some simple way so that predicate coverage and edge coverage (or node coverage) are not the same?
(c) Write out a corresponding set of test cases.

2.2. (a) Write out a set of test requirements that achieve **clause coverage** (CC) for Algorithm 1.
(b) Write out a corresponding set of test cases.

2.3. (a) Write out a set of test requirements that **restricted active clause coverage** (RACC) (also known as MCDC) for Algorithm 1.
(b) Write out a corresponding set of test cases.

3. Self-Assessment Questions
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 14 points). Which coverage model achieves the highest score?

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? *

13. Do you have at least one test case specifying the wrong number of values (2 or less, four or more) **

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.