

SE465 Notes

Minyang Jiang

January 11, 2017

0.1 Example 1

```
static public int findLast(int[] x, int y) {
    for (int i = x.length - 1; i > 0; i--) {
        if (x[i] == y) {
            return i;
        }
    }
    return -1;
}

@Test
public void testFindLast() {
    int[] x = new int[] {2, 3, 5};
    assertEquals(0, FindLast.findLast(x, 2));
}
```

1. Identify and fix the fault
for loop condition should be $i \geq 0$
2. If possible, identify a test case that does not exercise the fault
x is null
3. if possible, identify a test case that exercise the fault, but no error state
findLast([1, 2, 3], 2) will return -1
4. if possible, identify a test case that results in an error, but no failure
trying to find something not there ([2], 5)
5. Identify the first error state

0.2 Example 2

```
class LineSegment:
    def __init__(self, x1, x2):
        self.x1 = x1; self.x2 = x2;

    def intersect(a, b):
        return (a.x1 < b.x2) & (a.x2 > b.x1);
```

Establishing correctness of intersect:

- case analysis of the inputs

Other answers

- execute every statement of the unit under test
- feed random inputs
- check all outputs
- check values of each clause

rename inputs:

$a = a.x_1$ $b = b.x_1$
 $A = a.x_2$ $B = b.x_2$

- assume all points are distinct
- assume $a < b$ (we'll check both ways when constructing test cases)

- assume $a < A, b < B$

aAbB
abAB
abBA

run this test as 'python line-intersection-test.py'

```
from line_intersection import *
import unittest

class TestIntersection(unittest.TestCase):
    def test_aAbB(self):
        a = LineSegment(0,2)
        b = LineSegment(3,7)
        self.assertFalse(intersect(a,b))
        self.assertFalse(intersect(b,a))

    def test_abAB(self):
        a = LineSegment(0,4)
        b = LineSegment(3,7)
        self.assertTrue(intersect(a,b))
        self.assertTrue(intersect(b,a))

    def test_abBA(self):
        a = LineSegment(0,4)
        b = LineSegment(1,2)
        self.assertTrue(intersect(a,b))
        self.assertTrue(intersect(b,a))

    def test_equality(self):
        a = LineSegment(0,2)
        b = LineSegment(2,4)
        self.assertTrue(intersect(a,b))      # A = b
        self.assertTrue(intersect(b,a))      # B = a
        a = LineSegment(2,2)
        b = LineSegment(0,4)
        self.assertTrue(intersect(a,b))      # a = A
        self.assertTrue(intersect(b,a))      # b = B
        a = LineSegment(0,2)
        b = LineSegment(0,4)
        self.assertTrue(intersect(a,b))      # a = b
        self.assertTrue(intersect(b,a))      # b = a

if __name__ == '__main__':
    unittest.main()
```

0.3 •

Static:

- find faults
example:
 - type checking
 - dead code analysis
- code inspection functionality and style
- program verification

Dynamic

- observe failures
- must generate inputs
what are expected outputs?
- easy to run the program
- keywords
 - white-box testing
 - black-box testing

static techniques tradeoff:

- exhaustive
- subject to false positives

words I don't like

complete testing

~~exhaustive testing~~
~~full coverage~~

First big question: When should I stop testing?

1. when I run out of time
open-ended exploratory testing
for automatic input generation
2. when I'm close enough to being exhaustive
explored enough (all) of
behaviours / use cases
program states
inputs
statements / branches

observability, controllability

0.4 Coverage

- idea: find reduced space + cover it with test cases

Test Requirement (TR) - an element of an artifact that a test case must satisfy