Khushveen Kaur Umra

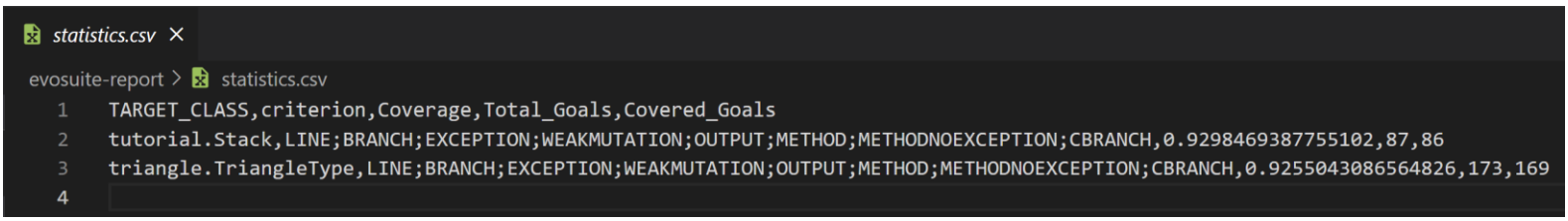
COMS 417 Sec 1

23rd February 2023

Assignment 2 – Evosuite

3a) Copy the contents of the statistics.csv file.

TARGET_CLASS	criterion	Coverage	Total_Goals	Covered_Goals
tutorial.Stack	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.929846939	87	86
triangle.TriangleType	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.925504309	173	169



b) How many tests are there for each program?

- ➔ Tutorial.Stack has: 5 tests
- ➔ Triangle.TriangleType has: 19 tests

c) Show an example of one test for ISOSCELES

→ The following tests are generated for an ISOSCELES triangle in the code.

i)
```
@Test(timeout = 4000)
Public void test01() throws Throwable {
Triangle triangle0= TriangleType.triangle(1334, 1334, 1352);
assertEquals(Triangle.ISOSCELES, triangle0);
}
```

ii)
```
@Test(timeout = 4000)
Public void test08() throws Throwable {
Triangle triangle0= TriangleType.triangle(2689, 1, 2689);
assertEquals(Triangle.ISOSCELES, triangle0);
}
```

iii)
```
@Test(timeout = 4000)
Public void test09() throws Throwable {
Triangle triangle0= TriangleType.triangle(4453, 4453, 1289);
assertEquals(Triangle.ISOSCELES, triangle0);
}
```

d) Show an example of a stack test that maximizes the stack

➔ The following portion of the code maximizes the stack by repeatedly pushing objects until the capacity is exceeded.

```
69    @Test(timeout = 4000)
70    public void test4()  throws Throwable  {
71        Stack<Object> stack0 = new Stack<Object>();
72        Object object0 = new Object();
73        stack0.push(object0);
74        stack0.push(object0);
75        Stack<String> stack1 = new Stack<String>();
76        stack0.push(stack1);
77        stack0.push(object0);
78        stack0.push(stack1);
79        Stack<Integer> stack2 = new Stack<Integer>();
80        stack0.push(object0);
81        stack0.push(object0);
82        stack0.push(o: "^EZ*");
83        stack0.push(stack2);
84        Integer integer0 = new Integer((-1));
85        stack0.push(integer0);
86        stack0.push(object0);
87        stack0.push(integer0);
88        // Undeclared exception!
89        try {
90          stack0.push(object0);
91          fail("Expecting exception: RuntimeException");
92
93        } catch(RuntimeException e) {
94          //
95          // Stack exceeded capacity!
96          //
97          verifyException("tutorial.Stack", e);
98        }
99    }
```

5a) Look more closely at the triangle tests. There is a fault in the program for the isosceles test. What happens with the test generation tool when it creates tests for the isosceles branch? State why this is a limitation of automated test generation tools.

→ There is a fault in the program for the isosceles test, where it incorrectly checks if s1 is equal to s2 twice, instead of also checking if s2 is equal to s3 . Due to the mistake, the program incorrectly classifies some triangles as not isosceles, when based on the inputs, they should be an isosceles triangle. This is a limitation in the implementation of the isosceles test. Here, when the test generation tool created tests for the isosceles branch, it generated test cases that satisfied the faulty condition for an isosceles branch, where the program returned incorrect results for some isosceles triangles.

For example, in the generated test cases, test number # 03 gives the inputs as (2, 1, 1), and expects it to be an invalid triangle, whereas in reality, it should be classified as an isosceles triangle.

Now, when the test generation tool creates tests for an isosceles triangle, it creates test cases that abide by the rules and the code path for the isosceles triangle condition, in this case, ((s1 == s2) || (s1 == s2) || (s1== s3)). But, the automated test generation tools are not able to detect any faults in the implementation of the code, for example, in this code, the generation tool was not able to detect the fault in the condition statement. This is a limitation of automated test generation tools because they only rely on the correctness of the code, and does not always cover all the possible execution paths, or does not always generate all possible test cases.

5a) Take a screenshot of the 'statistics.csv' file.

TARGET_CLASS	criterion	Coverage	Total_Goals	Covered_Goals
tutorial.Stack	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.929846939	87	86
triangle.TriangleType	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.925504309	173	169
power.Power	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.915443789	205	200

evosuite-report > statistics.csv

```
1  TARGET_CLASS,criterion,Coverage,Total_Goals,Covered_Goals
2  tutorial.Stack,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9298469387755102,87,86
3  triangle.TriangleType,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9255043086564826,173,169
4  power.Power,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.915443789456081,205,200
5
```

b) Provide a screenshot(s) of the test file

```java
package power;

import org.junit.Test;
import static org.junit.Assert.*;
import static org.evosuite.runtime.EvoAssertions.*;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.junit.runner.RunWith;
import power.Power;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true,
useVFS = true, useVNET = true, resetStaticState = true, separateClassLoader = true)

public class Power_ESTest extends Power_ESTest_scaffolding {

  @Test(timeout = 4000)
  public void test00()  throws Throwable  {
      int int0 = Power.inverse((-1568), 1817);
      assertEquals(1817, int0);
  }

  @Test(timeout = 4000)
  public void test01()  throws Throwable  {
      int int0 = Power.power((-1), 0);
      assertEquals((-1), int0);
  }

  @Test(timeout = 4000)
  public void test02()  throws Throwable  {
      int int0 = Power.power(5, 10);
      assertEquals(9765625, int0);
  }
```

```java
@Test(timeout = 4000)
public void test03()  throws Throwable  {
    int int0 = Power.power(0, (-177));
    assertEquals((-1), int0);
}

@Test(timeout = 4000)
public void test04()  throws Throwable  {
    int int0 = Power.power(1817, 5);
    assertEquals((-1), int0);
}

@Test(timeout = 4000)
public void test05()  throws Throwable  {
    int int0 = Power.power(0, 10);
    assertEquals(0, int0);
}

@Test(timeout = 4000)
public void test06()  throws Throwable  {
    int int0 = Power.inverse(1148, (-2592));
    assertEquals(0, int0);
}

@Test(timeout = 4000)
public void test07()  throws Throwable  {
    // Undeclared exception!
    Power.inverse(1410065408, (-1));
}

@Test(timeout = 4000)
public void test08()  throws Throwable  {
    int int0 = Power.inverse(4, (-1163));
    assertEquals((-213168335), int0);
}
```

```
75      @Test(timeout = 4000)
76      public void test09()  throws Throwable  {
77          int int0 = Power.inverse(10, 0);
78          assertEquals(1, int0);
79      }
80
81      @Test(timeout = 4000)
82      public void test10()  throws Throwable  {
83          int int0 = Power.power(2, 2);
84          assertEquals(4, int0);
85      }
86
87      @Test(timeout = 4000)
88      public void test11()  throws Throwable  {
89          int int0 = Power.power((-1163), 1817);
90          assertEquals((-1), int0);
91      }
92
93      @Test(timeout = 4000)
94      public void test12()  throws Throwable  {
95          int int0 = Power.power(10, (-1));
96          assertEquals((-1), int0);
97      }
98
99      @Test(timeout = 4000)
100     public void test13()  throws Throwable  {
101         Power power0 = new Power();
102     }
103 }
```

c) Has the faulty branch from power(int, int) been covered? If so, does it find the fault? If not, explain.

if (right <= 0)

{ rslt = -1; }

➔ Yes, the faulty branch from 'power (int, int)' has been covered by the automated generated test cases, specifically in the following test case:

```
@Test(timeout = 4000)
public void test03() throws Throwable {
int int0 = Power.power(0, (-177));
assertEquals((-1), int0);
}
```

The automated generated test cases do find the fault, as the right value is less than or equal to 0 in the test case, and the faulty branch is executed, setting the rslt to -1, rather than genuinely calculating the result.

6) Compare the code coverage report with the one you got for assignment one (the coverage before you fixed the faults and after you added new coverage.) Include a screenshot of the coverage report for the evosuite tests and discuss the difference from your original, manually created coverage, (if any).

→ After manually adding the tests inputs and oracle generated by evosuite in jacoco tool, I discovered that out of the 13 tests, 3 tests failed. It also missed more instructions and branches in comparison, for both the functions power and inverse. Although the % percentage for both evosuite and jacoco were nearly similar, jacoco managed to point out the failed test cases.

For assignment 1, before correcting the faults, I was able to achieve a 100% coverage for the inverse function, but by using the generated test cases, I was only able to achieve about 85% coverage. Although, the interesting fact was that my test cases from assignment 1 and the generated test cases for the power function, had the same % coverage, due to the faulty condition statement in Power.java.

Coverage report for automated generated test cases using evosuite:

TARGET_CLASS	criterion	Coverage	Total_Goals	Covered_Goals
tutorial.Stack	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.929846939	87	86
triangle.TriangleType	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.925504309	173	169
power.Power	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.915443789	205	200

Coverage report for manually created coverage using jacoco:

Power

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
power(int, int)		92%		93%	1	9	1	10	0	1
inverse(int, int)		85%		75%	1	3	1	6	0	1
Power()		100%		n/a	0	1	0	1	0	1
Total	6 of 65	90%	2 of 20	90%	2	13	2	17	0	3

Coverage report for manual test cases from assignment 1 before correcting the faults:

Power

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
power(int, int)		92%		87%	2	9	1	10	0	1
inverse(int, int)		100%		100%	0	3	0	6	0	1
Power()		100%		n/a	0	1	0	1	0	1
Total	3 of 65	95%	2 of 20	90%	2	13	1	17	0	3