IT-314: Software Engineering

Lab Assignment: 09

Title: Mutation Testing

Lab Group: G2

Name: Parth Vadodaria

ID: 202201174

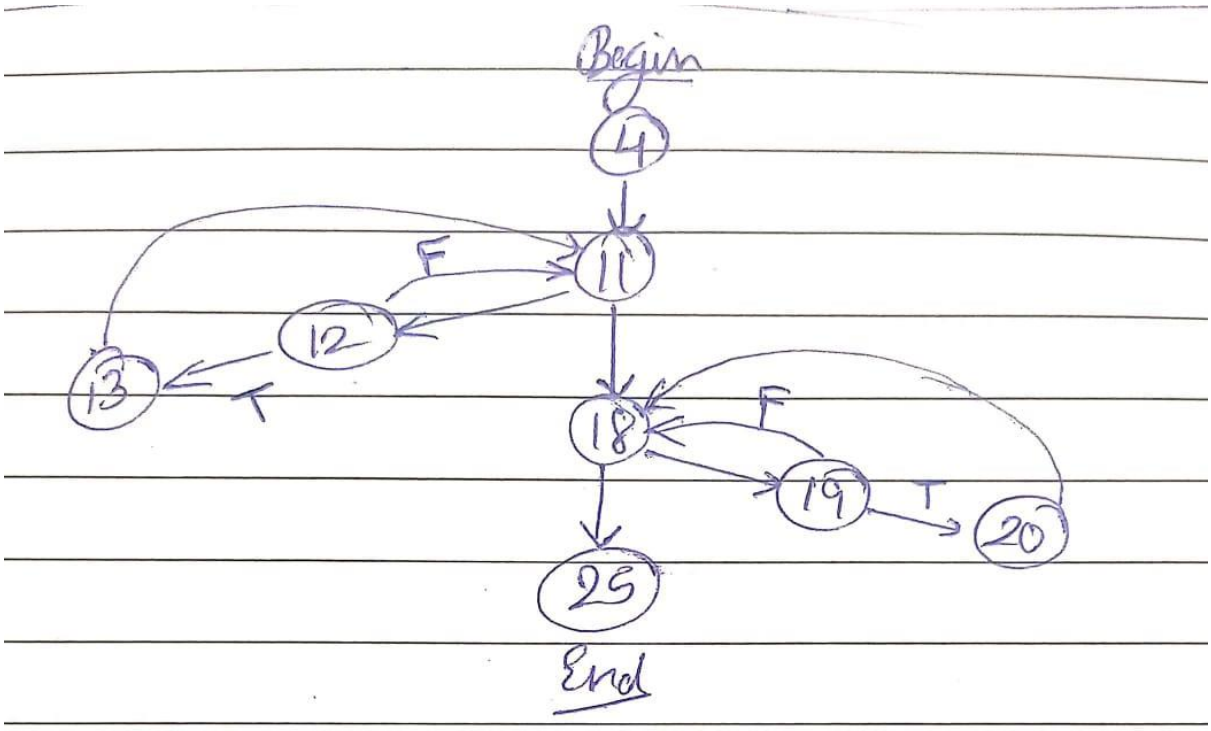
Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code, so the focus is on creating test sets that satisfy some particular coverage criteria.

```
import java.util.ArrayList;

2 usages
public class Lab9 {
    3 usages
    public int doGraham(ArrayList<Point> p) {
        int i, j, min, M;

        Point t;
        min = 0;

        // search for minimum:
        for (i = 1; i < p.size(); ++i) {
            if (p.get(i).y < p.get(min).y) {
                min = i;
            }
        }

        // continue along the values with same y component
        for (i = 0; i < p.size(); ++i) {
            if ((p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x)) {
                min = i;
            }
        }

        return min;
    }
}
```

```java
11 usages
public class Point{
    3 usages
    int x;
    5 usages
    int y;

    6 usages
    Point(int x, int y){
        this.x=x;
        this.y=y;
    }

}
```

For the given code fragment, you should carry out the following activities.

1. Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG). You are free to write the code in any programming language.



Note:

The numbers in the nodes are written according to the lines in the written code. Also, 4->11 means all the lines between them are covered as well. Similarly for other nodes as well.
T -> represents true for if condition
F -> represents false for if condition or else part.

2. Construct test sets for your flow graph that are adequate for the following criteria:

    a. Statement Coverage.
    b. Branch Coverage.
    c. Basic Condition Coverage
    d. Path Coverage

{(3,9), (4,7), (5,7)} is the input to the programme which cover all the coverages 100%.
Explanation:

- Path 4 to 11 will be executed definitely.
- Now comes the for loop at line 11 which will run 2 times here:
    - For i=1, min=0 which is (3,9) and 7<9 which satisfies the if condition in line 12. It will lead change in min to 1 in line 13.
    - For i=2, 7<7 is false which leads to false condition and returns to 11.
    - From the above two points, it can be claimed that the required two branches at line 12 are covered. Similarly, the basic condition is also covered in if statement at line 12.
- The execution moves forward to line 18. Loop at line 18 which will run 3 times here:
    - For i=0, min=1, the first condition in if statement at line 19 results false and execution of for loop moves forward.
    - For i=1, the first condition in if statement at line 19 results true but second false.
    - For i=2, the first condition and second condition in if statement at line 19 results true. Hence, the statement at line 20 is executed.

- From the above three points, it can be claimed that the required two branches at line 19 are covered. Similarly, the two basic conditions are also covered in if statement at line 19.

- Here comes the end to the programme.
- As per explanation, all the statements, branches and conditions are covered successfully.
- At the end all the paths to the control flow graph are also covered.

3. Create a test set that satisfies the path coverage criterion where every loop is explored at least zero, one or two times.

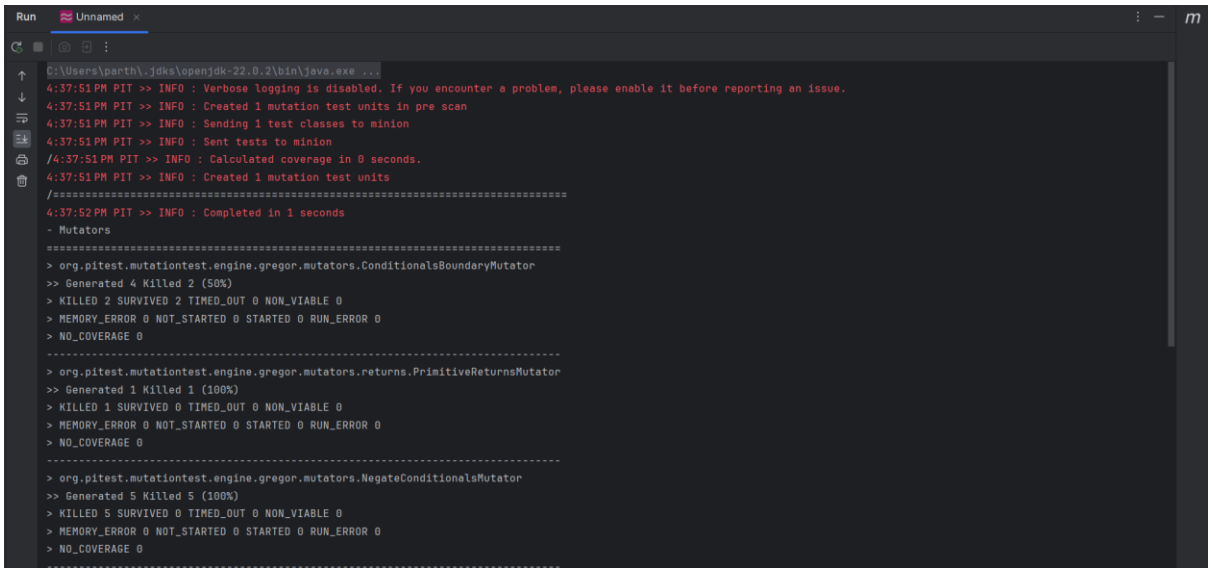
   - Zero Time: **{(0,0)}**
   - One Time: **{(3,9), (4,7)}**
   - Two or more times: **{(3,9), (4,7), (5,7)}**

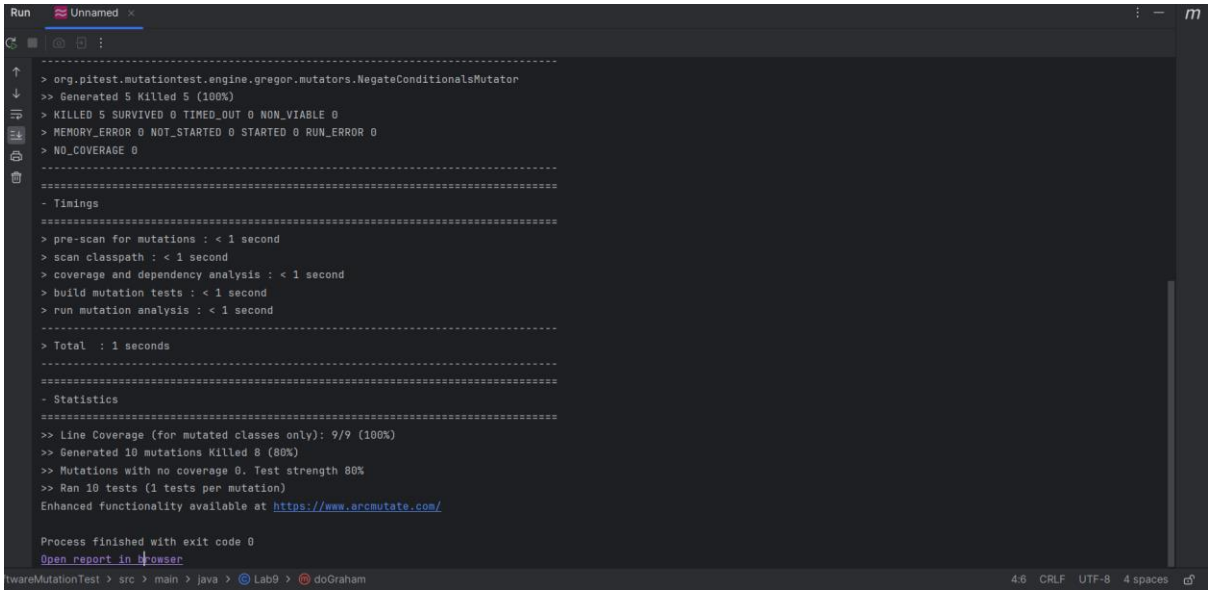
4. For the test set you have just checked can you find a mutation of the code (i.e. the deletion, change or insertion of some code) that will result in failure but is not detected by your test set. You have to use the mutation testing tool.

Junit Test:

```java
class Lab9Test {
    4 usages
    Lab9 l;
    @BeforeEach
    void setUp() { l=new Lab9(); }

    @Test
    void doGrahamTest() {
        ArrayList<Point> p=new ArrayList<>();
        p.add(new Point( x: 3, y: 9));
        p.add(new Point( x: 4, y: 7));
        p.add(new Point( x: 5, y: 7));
        assertEquals( expected: 2,l.doGraham(p));
    }

    @Test
    void doGrahamTest1() {
        ArrayList<Point> p=new ArrayList<>();
        p.add(new Point( x: 0, y: 0));
        assertEquals( expected: 0,l.doGraham(p));
    }

    @Test
    void doGrahamTest2() {
        ArrayList<Point> p=new ArrayList<>();
        p.add(new Point( x: 3, y: 9));
        p.add(new Point( x: 4, y: 7));
        assertEquals( expected: 1,l.doGraham(p));
    }
}
```

Mutation Result (Pitest tool is used):





Lab9.java

```
1   import java.util.ArrayList;
2
3   public class Lab9 {
4       public int doGraham(ArrayList<Point> p) {
5           int i, j, min, M;
6
7           Point t;
8           min = 0;
9
10          // search for minimum:
11 2        for (i = 1; i < p.size(); ++i) {
12 2            if (p.get(i).y < p.get(min).y) {
13                  min = i;
14              }
15          }
16
17          // continue along the values with same y component
18 2        for (i = 0; i < p.size(); ++i) {
19 3            if ((p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x)) {
20                  min = i;
21              }
22          }
23
24 1        return min;
25      }
26  }
```

Mutations

```
11  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
12  1. negated conditional → KILLED
    2. changed conditional boundary → SURVIVED
18  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
19  1. negated conditional → KILLED
    2. negated conditional → KILLED
    3. changed conditional boundary → SURVIVED
24  1. replaced int return with 0 for Lab9::doGraham → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- Lab9Test.[engine:junit-jupiter]/[class:Lab9Test]/[method:doGrahamTest()] (18 ms)

Report generated by PIT 1.15.8

Reason of Survival of the mutations for test case {(3,9), (4,7), (5,7)}:

- Conditional Boundary Mutations on Line 12:
  - One mutation changes the comparison in the first loop from < to <=.
  - This change would not affect the outcome, as there are no points with y=7 until the loop already assigns min = 1 due to a smaller y value.
  - Since this condition is not triggered, this mutation does not affect the result, and thus it survives.
- Conditional Boundary Mutations on Line 19:
  - Similarly, a mutation could change the comparison of x values from > to >= in the second loop.
  - With points (4, 7) and (5, 7), this change would not affect the outcome because the comparison (4,7) < (5,7) will still hold true due to x=5 being greater than x=4.
  - The mutated condition does not produce a different outcome, so the test case does not detect this mutation, and it survives.

5. After generating the control flow graph, check whether your CFG match with the CFG generated by Control Flow Graph Factory Tool and Eclipse flow graph generator. (In your submission document, mention only “Yes” or “No” for each tool).

   YES

6. Devise minimum number of test cases required to cover the code using the aforementioned criteria.

   Please refer Junit test image in Q4.

7. This part of the exercise is very tricky and interesting. The test cases that you have derived in Step 2 are then used to identify the fault when you make some modifications in the code. Here, you need to insert/delete/modify a piece of code that will result in failure but it is not detected by your test set – derived in Step 2. Write/identify a mutation

code for each of the three operations separately, i.e., by deleting the code, by inserting the code, by modifying the code.

Please refer the reason of the survival of the mutation test in Q4.

8. Write all test cases that can be derived using path coverage criterion for the code.

Please refer Q2 d. Path coverage and its explanation.