IT - 314 Software Engineering

Lab 9 : Mutation Testing

Darshan Gami - 202201205
Lab Group : 3

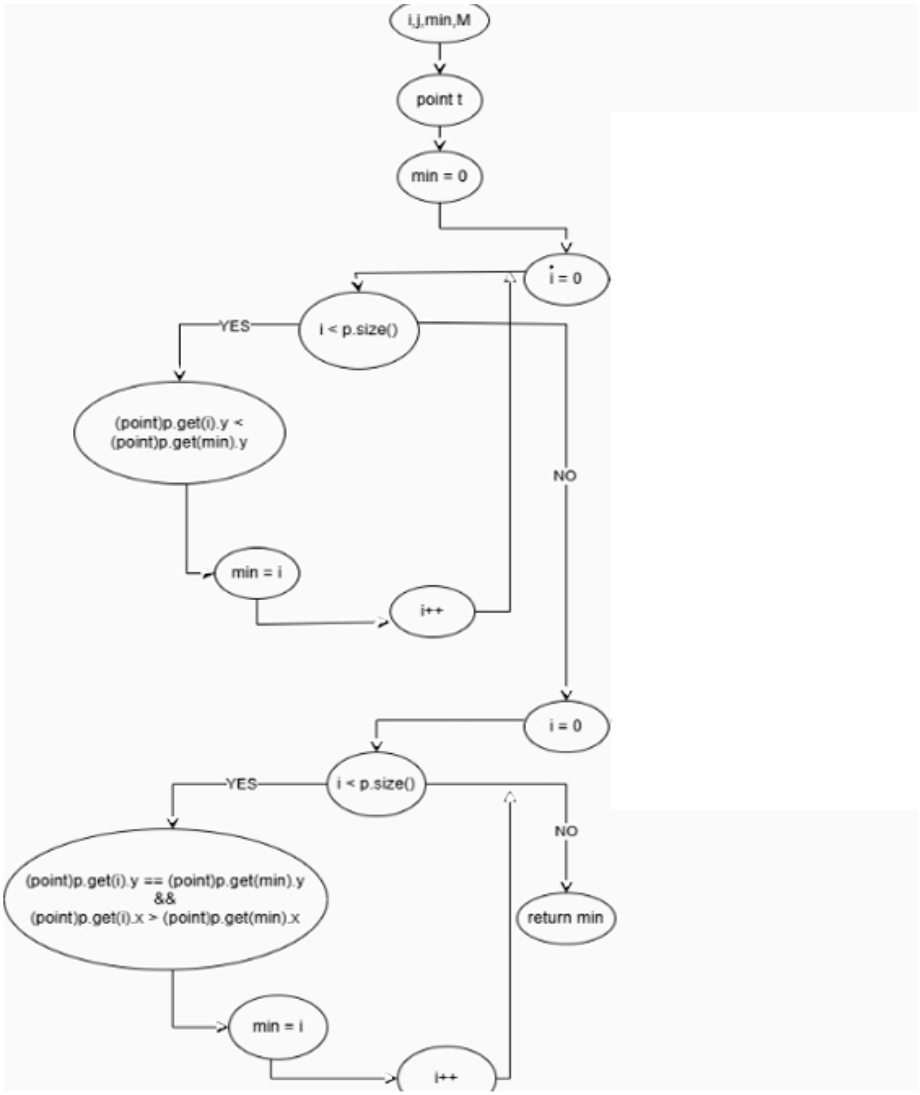
Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() 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.

Code :

```
public class Point {
   double x;
   double y;

   public Point(double x, double y) {
      this.x = x;
      this.y = y;
   }
}

public class ConvexHull {
   public void doGraham(Vector<Point> p) {
      if (p.size() == 0) {
         return;
      }

      int minY = 0;
      for (int i = 1; i < p.size(); i++) {
         if (p.get(i).y < p.get(minY).y ||
           (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x)) {
            minY = i;
         }
      }

      Point temp = p.get(0);
      p.set(0, p.get(minY));
      p.set(minY, temp);
   }
}
```

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.**



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.

a. Statement Coverage

Test Cases for Statement Coverage

Test Case 1: p is an empty vector.
Expected Outcome: The method should terminate immediately without performing any operations.

Test Case 2: p contains only one point, e.g., [(0, 0)].
Expected Outcome: No swapping is necessary as there is only one point. However, all statements should still be executed.

Test Case 3: p contains multiple points with unique y-values, e.g., [(1, 3), (2, 2), (3, 4)].
Expected Outcome: The method should recognize (2, 2) as the point with the minimum y-value and swap it with the point at index 0.

Test Case 4: p contains points with identical y-values but different x-values, e.g., [(3, 2), (1, 2), (2, 2)].
Expected Outcome: Among points with the same y-value, the one with the smallest x-value, (1, 2), should be identified as the minimum and swapped with the first point.

b. Branch Coverage

Test Cases for Branch Coverage

Test Case 1: p is an empty vector.
Branch Tested: The condition if (p.size() == 0) is tested.

Test Case 3: Multiple points with distinct y-values, e.g., [(1, 3), (2, 2), (3, 4)].
Branch Tested: The condition p.get(i).y < p.get(minY).y within the loop is tested.

Test Case 4: Points with the same y-value but different x-values, e.g., [(3, 2), (1, 2), (2, 2)].
Branch Tested: The condition (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x) is tested.

c. Basic Condition Coverage

Test Cases for Basic Condition Coverage

1. **Test Case 3:** Distinct y-values, e.g., `[(1, 3), (2, 2), (3, 4)]`.
   Conditions Tested: `p.get(i).y < p.get(minY).y` (true and false).

2. **Test Case 4:** Same y-values, varying x-values, e.g., `[(3, 2), (1, 2), (2, 2)]`.
Conditions Tested: `p.get(i).y == p.get(minY).y` (true and false) and `p.get(i).x < p.get(minY).x` (true and false).

Summary of Test Cases :

Test Case	Input Points (Vector p)	Expected Behavior	Potential Mutation	Mutation Effect
1	[]	Method returns immediately.	Change the condition if (p.size() == 0) to if (p.size() != 0)	The method will not return immediately, leading to incorrect behavior.
2	[(0, 0)]	No swap; single point remains unchanged.	Remove the check for single-point case.	The method would attempt to perform unnecessary operations.
3	[(1, 3), (2, 2), (3, 4)]	Point (2, 2) is swapped with (1, 3).	Invert the condition if (p.get(i).y < p.get(minY).y) to if (p.get(i).y >= p.get(minY).y)	The wrong point would be identified as the minimum, causing incorrect swapping.
4	[(3, 2), (1, 2), (2, 2)]	Point (1, 2) is swapped with (3, 2).	Invert the condition if (p.get(i).x < p.get(minY).x) to if (p.get(i).x >= p.get(minY).x)	The wrong point with the same y-value would be identified as the minimum.

3. 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.

Mutation 1: Change the Comparison Operator for the y Coordinate

- **Original Code :**

```
if (p.get(i).y < p.get(minY).y || (p.get(i).y == p.get(minY).y &&
 p.get(i).x < p.get(minY).x))
```

- **Mutation:** Change < to <= for y comparison

```
if (p.get(i).y <= p.get(minY).y || (p.get(i).y == p.get(minY).y &&
p.get(i).x < p.get(minY).x))
```

Impact:

This mutation may not always be detected if the minimum y values have different x values because it does not affect the outcome unless there are multiple points with the same y value. In scenarios where the y-values are distinct, the mutation has no impact on the method's behavior.

Detection:

The current test cases might not detect this mutation if there is only one point with the smallest y-coordinate, as the outcome would remain the same. To ensure the mutation is detected, additional test cases are needed where multiple points have the same minimum y value. These test cases should verify that the method correctly selects the point with the smallest x value, ensuring that the mutation is identified.

Mutation 2: Change the x Comparison Operator

- **Original Code:**

```
(p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x)
```

- **Mutation: Change < to > :**

```
(p.get(i).y == p.get(minY).y && p.get(i).x > p.get(minY).x)
```

Impact:

This mutation would reverse the logic for selecting the x-coordinate when y values are equal. Specifically, the mutated code would incorrectly choose the point with the larger x value instead of the smaller one, leading to incorrect behavior in cases where multiple points share the same y value.

Detection:

Our current test case that involves points with the same y values but different x values (Test Case 4) would likely detect this mutation, as it specifically checks for the selection of the smallest x value. However, if our test suite lacks scenarios where points have equal y values, this mutation might go undetected. To ensure robustness, the test suite should include cases that thoroughly test this condition.

Mutation 3: Remove the if (p.size() == 0) Check

- **Original Code :**

```
if (p.size() == 0) {

return;}
```

- **Mutation:** Delete the line if (p.size() == 0) { return; }.

Impact:

Removing the check for an empty vector would cause the method to attempt accessing elements in an empty vector. This would result in an IndexOutOfBoundsException if the vector is empty, leading to a runtime error.

Detection:

Our current test case for an empty vector (p = []) would effectively detect this mutation. The test case is designed to verify that the method handles an empty vector gracefully, and without the check, the mutation would cause an exception to be thrown, revealing the issue.

Mutation 4: Modify the Swap Logic

- **Original Code :**

```
Point temp = p.get(0);

p.set(0, p.get(minY));

p.set(minY, temp);
```

- **Mutation:** Swap p.set(0, p.get(minY)) and p.set(minY, temp) lines

```
Point temp = p.get(0);

 p.set(minY, temp);

 p.set(0, p.get(minY));
```

Impact

This mutation would disrupt the intended swapping logic. If minY == 0, the vector would remain unchanged, which might seem correct but is unintended. If minY != 0, the method could produce unexpected results, as the swapping behavior would not function as intended.

Detection

Our current test cases are likely to detect this mutation when minY is not equal to 0, since the incorrect swapping would result in an improperly arranged vector. However, if the minimum y point is already at index 0, the mutation could go unnoticed. To ensure thorough detection, we should include test cases where minY is not 0, verifying that the swapping logic is executed correctly.

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

- **Test Case 1:** Loop Explored Zero Times

Input: An empty vector p.

Test Code :

```
Vector<Point> p = new Vector<Point>();

ConvexHull ch = new ConvexHull();

ch.doGraham(p);
```

Test Case 1: Empty Vector

- **Input:** A vector with no points (p = []).
- **Expected Result:** The method should return immediately without any processing.

- **Explanation:** Since p.size() == 0, the method will hit the if (p.size() == 0) { return; } condition and exit immediately, skipping the rest of the code.
- **Coverage:**
    - Statement Coverage: Covers the return statement for an empty vector.
    - Branch Coverage: Covers the if (p.size() == 0) branch where the condition is true.
    - Condition Coverage: Fully evaluates the condition p.size() == 0 as true.

```
Vector<Point> p = new Vector<Point>();

p.add(new Point(0, 0));

ConvexHull ch = new ConvexHull();

ch.doGraham(p);
```

Test Case 2: Single Point in Vector

- **Input:** A vector with one point, (0, 0).
- **Expected Result:** Since p.size() is 1, the loop should not be entered, and the method should effectively swap the point with itself, leaving the vector unchanged.
- **Explanation:** With only one point, the loop starting from for (int i = 1; i < p.size(); i++) is not entered because i = 1 is equal to p.size(). Thus, the minimum y point is already at the first position, and swapping it with itself does nothing.
- **Coverage:**
    - Statement Coverage: Covers the initialization of minY and the swap logic.
    - Branch Coverage: Covers the if (p.size() == 0) branch where the condition is false.

- **Condition Coverage:** Evaluates the loop condition for i = 1 as false, covering the scenario where p.size() == 1.

Test Case 3: Loop Explored Twice

- Input: A vector with two points: (1, 1) and (0, 0), where the first point (1, 1) has a higher y-coordinate than the second point (0, 0).
- Test Code:

```
Vector<Point> p = new Vector<Point>();

p.add(new Point(1, 1));

p.add(new Point(0, 0));

ConvexHull ch = new ConvexHull();

ch.doGraham(p)
```

Test Case 3: Two Points, First Point Has Higher Y-Coordinate

- **Input:** A vector with two points: (1, 1) and (0, 0), where the first point (1, 1) has a higher y-coordinate than the second point (0, 0).
- **Expected Result**: The method should enter the loop and find that the second point has a lower y-coordinate than the first, updating minY to 1. A swap should occur, placing (0, 0) at the beginning of the vector.
- **Explanation:** The loop iterates once, comparing the two points. Since (0, 0).y is less than (1, 1).y, minY is updated to 1. The method then swaps the first point with the point at minY.
- **Coverage:**
  - Statement Coverage: Covers all statements, including the loop, comparison, minY update, and swap logic.
  - Branch Coverage: Covers both branches of the if condition within the loop, where p.get(i).y < p.get(minY).y is true.

- **Condition Coverage:** Evaluates both conditions within the if statement:
  - p.get(i).y < p.get(minY).y (true)
  - (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x) (not evaluated in this case as the first condition is true).

Test Case 4: Multiple Points with Distinct Y-Coordinates

- **Input:** A vector with multiple points where the algorithm should correctly identify the point with the lowest y-coordinate and perform the necessary swap.
- **Test Code:**

```
Vector<Point> p = new Vector<Point>();

p.add(new Point(2, 2)); // Point A

p.add(new Point(1, 0)); // Point B (minimum y-coordinate)

p.add(new Point(0, 3)); // Point C

ConvexHull ch = new ConvexHull();

ch.doGraham(p);
```

Test Case: Three Points, Identifying the Minimum Y-Coordinate

- **Input:** A vector with three points: (2, 2), (1, 0), and (0, 3).
- **Expected Result**: The method should iterate through all three points in the vector and identify the second point (1, 0) as having the lowest y-coordinate of 0. Consequently, minY will be updated to reflect this value. After performing the necessary swap, the point (1, 0) will be positioned at the front of the vector, leading to the final order of points being (1, 0), (2, 2), and (0, 3).

- **Explanation:**
  - Initially, minY is set to the y-coordinate of the first point (2, 2), which is 2.
  - As the loop begins, it compares the first point (2, 2) with the second point (1, 0). Since (1, 0).y (0) is less than (2, 2).y (2), minY is updated to 0, and minIndex is set to 1.
  - In the subsequent iteration, the method compares (1, 0) with the third point (0, 3). Here, (0, 3).y (3) is greater than (1, 0).y (0), so no changes are made to minY or minIndex.
  - After the loop concludes, the algorithm swaps the first point (2, 2) with the point at minIndex, resulting in the updated vector order.
- **Coverage:**
  - **Statement Coverage:** All statements executed, including the loop and swap.
  - **Branch Coverage:** Both branches of the if condition are covered:
    - p.get(i).y < p.get(minY).y (true).
    - p.get(i).y >= p.get(minY).y (evaluated).
  - **Condition Coverage:** Evaluates both conditions within the if statement:
    - p.get(i).y < p.get(minY).y (true).
    - p.get(i).y >= p.get(minY).y (evaluated).

Lab Execution

1. **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 documentmention only “Yes” or “No” for each tool).**

   Control Flow Graph Factory :- YES

   Eclipse flow graph generator :- YES

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

   Summary of Minimum Test Cases:

   1. Branch Coverage: 4 test cases
   2. Basic Condition Coverage: 3 test cases
   3. Path Coverage: 2 test cases

   Total Test Cases:

   - 4 (Branch Coverage) + 3 (Basic Condition Coverage) + 2 (Path Coverage) = 9 test cases