IT314-Software Engineering
Lab9-Mutation Testing

Abhinav-202201112

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.

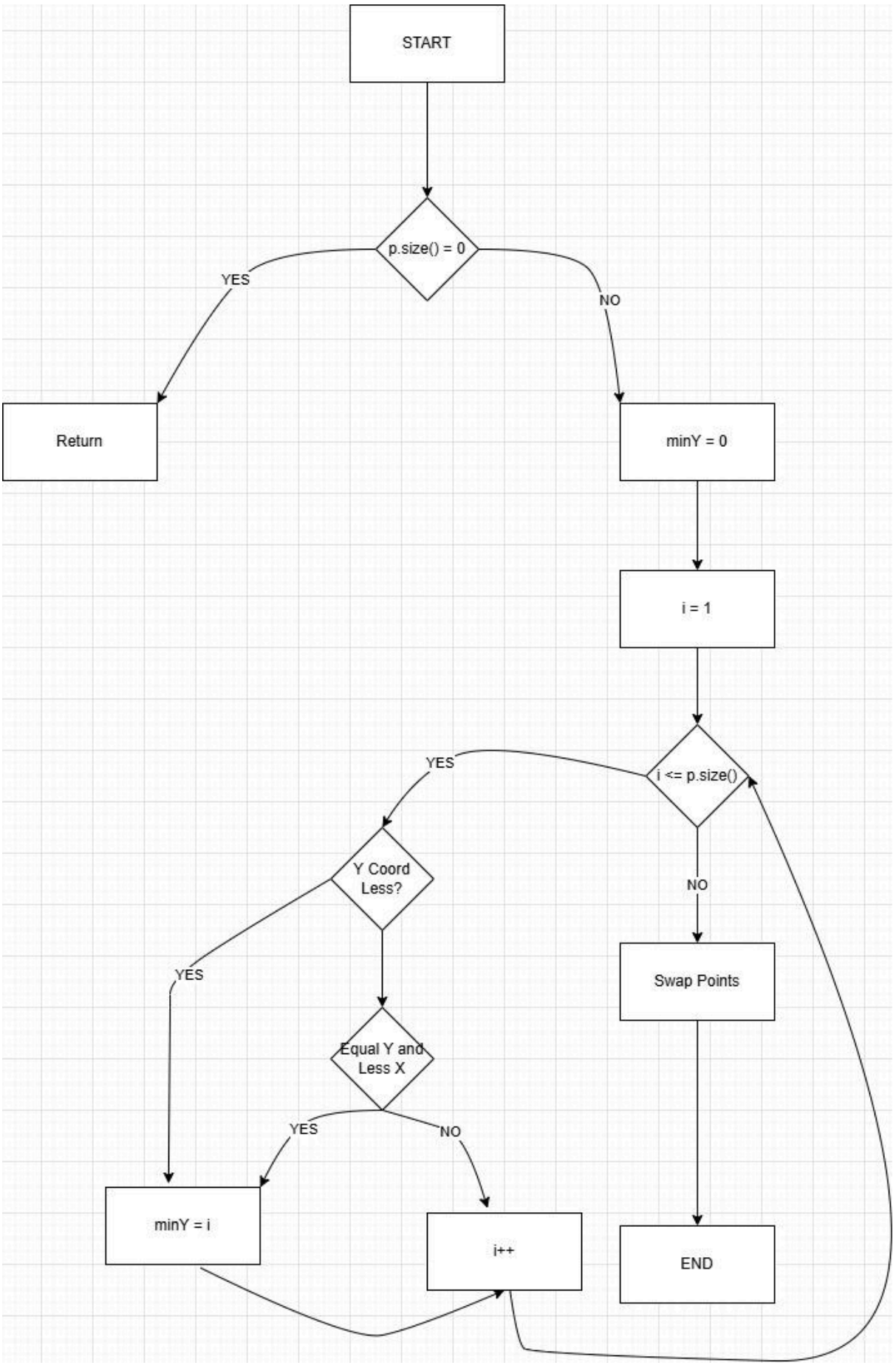
1. Convert the code comprising the beginning of the doGraham method into acontrol flow graph (CFG). You are free to write the code in any programming language.

```
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);
    }
}
```

-*Control Flow graph of above code*


START
p.size() = 0
YES
NO
Return
minY = 0
i = 1
i <= p.size()
YES
NO
Y Coord Less?
YES
Swap Points
Equal Y and Less X
YES
NO
minY = i
i++
END

Q2. Construct test sets for your flow graph that are adequate for the following criteria: a. Statement Coverage. b. Branch Coverage. c. Basic Condition Coverage.

1).Statement Coverage

To achieve statement coverage, we need to ensure that each line of code is executed at least once.

Test Case:

- *Test 1: Input vector p is empty (p.size() == 0)*
  - *Expected result: The function should return immediately without executing further.*
- *Test 2: Input vector p has at least one Point element.*
  - *Expected result: The function should proceed to find the Point with the minimum y-coordinate.*

2).Branch Coverage

For branch coverage, we need to ensure that each decision (branch) in the code is evaluated as both true and false at least once.

Test Cases:

- *Test 1: Empty vector p (p.size() == 0)*
  - *Expected result: if (p.size() == 0) branch is true, and the function returns immediately.*
- *Test 2: Vector p with one Point (e.g., Point(0, 0))*
  - *Expected result: if (p.size() == 0) branch is false, so the function proceeds to the for loop, which does not iterate as there's only one point.*
- *Test 3: Vector p with multiple Points where no point has a smaller y-coordinate than p[0]*
  - *Example: p = [Point(0, 0), Point(1, 1), Point(2,2)]*
  - *Expected result: The if condition inside the for loop is always false, and minY remains 0.*

● *Test 4: Vector p with multiple Points where another point has a smaller y-coordinate than p[0]*
○ *Example: p = [Point(2, 2), Point(1, 0), Point(0,3)]*
Expected result: The if condition inside the for loop becomes true, updating minY.

3).Basic Condition Coverage

For basic condition coverage, we need to test each condition independently within the branches.
Test Cases:
● *Test 1: Empty vector p (p.size() == 0)*
○ *Expected result: if (p.size() == 0) is true.*
● *Test 2: Non-empty vector p (p.size() > 0)*
○ *Expected result: if (p.size() == 0) is false.*
● *Test 3: Multiple points where only p.get(i).y <p.get(minY).y is true*
○ *Example: p = [Point(1, 1), Point(0, 0), Point(2,2)]*
○ *Expected result: The first condition p.get(i).y <p.get(minY).y is true, so minY is updated.*
● *Test 4: Multiple points where p.get(i).y == p.get(minY).y is true, and p.get(i).x < p.get(minY).x is also true*
○ *Example: p = [Point(1, 1), Point(0, 1), Point(2,2)]*
○ *Expected result: Both conditions are true, so minY is updated.*

Q3. 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 test set. You have to use the mutation testing tool.

Deletion Mutation

Mutation: Remove the assignment of minY to 0 at the beginning of the method.

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

Impact:

This mutation leaves minY uninitialized when accessed, leading to undefined behavior. The test cases lack checks for proper initialization, which could allow faults to go undetected.

2). Insertion Mutation

Mutation: Insert a line that overrides minY incorrectly based on a condition that should not occur.

```
public class ConvexHull {
      public void doGraham(Vector<Point> p) {
            if (p.size() == 0) {
               return;
}
              int minY = 0;
                 if (p.size() > 1) {
                minY = 1;
}
                       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);
                }
}
```

3). Modification Mutation

Mutation: Change the logical operator from || to && in the conditional statement.

```
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);
                        }
}
```

Analyzing Detection by Test Cases

1. Statement Coverage:

- Removing the initialization of minY might not be detected, as it may not trigger an immediate exception depending on the surrounding code logic.

2. Branch Coverage:

- Setting minY to 1 through an insertion may result in incorrect outcomes, but if the tests do not specifically verify the positions of points after execution, this issue could remain unnoticed.

3. Basic Condition Coverage:

- Modifying the logical operator from || to && does not lead to a crash, and the test cases do not verify if minY updates correctly under these conditions, so this change might go undetected.

Q4. Create a test set that satisfies the path coverage criterion where every loop is explored at least zero, one or two times.

-To satisfy the path coverage criterion and ensure every loop is explored zero, one, or two times, we will create the following test cases:

Test Case 1: Zero IterationsInput:An empty vector p.
Description: This case ensures that no iterations of either loop occur.
Expected Output: The function should handle this case gracefully (ideally return an empty result or a specific value indicating no points).

Test Case 2: One Iteration (First Loop)
Input: A vector with one point p (e.g., [(3, 4)]).
Description: This case ensures that the first loop runs exactly once (the minimum point is the only point).
Expected Output: The function should return the only point in p

Test Case 3: One Iteration (Second Loop)
Input: A vector with two points that have the same y-coordinate but different x-coordinates (e.g.,[(1, 2), (3, 2)]).
Description: This case ensures that the first loop finds the minimum point, and the second loop runs exactly once to compare the x-coordinates.
Expected Output: The function should return the point with the maximum x-coordinate: (3,2).

Test Case 4: Two Iterations (First Loop)
Input: A vector with multiple points, ensuring at least two with the same y-coordinate (e.g.,[(3, 1), (2, 2), (5, 1)]).

Description: This case ensures that the first loop finds the minimum y-coordinate (first iteration for (3,1)) and continues to the second loop.
Expected Output: Should return (5, 1) as it has the maximum x-coordinate among pointswith the
same y.

Test Case 5: Two Iterations (Second Loop)
Input: A vector with points such that more than one point has the same minimum y-coordinate (e.g., [(1,1), (4, 1), (3, 2)]).
Description: This case ensures the first loop finds (1, 1), and the second loop runs twice tocheck other
points with y = 1.
Expected Output: Should return (4, 1) since it has the maximum x-coordinate.

LAB Execution

Q1). 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.
- Control Flow Graph Factory :- YES
Eclipse flow graph generator :- YES

Q2).Devise minimum number of test cases required to cover the code using the aforementioned criteria.
-Statement Coverage: 3 test cases
1. Branch Coverage: 4 test cases
2. Basic Condition Coverage: 4 test cases
3. Path Coverage: 3 test cases
Summary of Minimum Test Cases:
- *Total: 3 (Statement) + 4 (Branch) + 4 (Basic Condition) + 3 (Path) = 14 test cases*

Q4 and Q4 Same as Part I)