It-314

Software Engineering

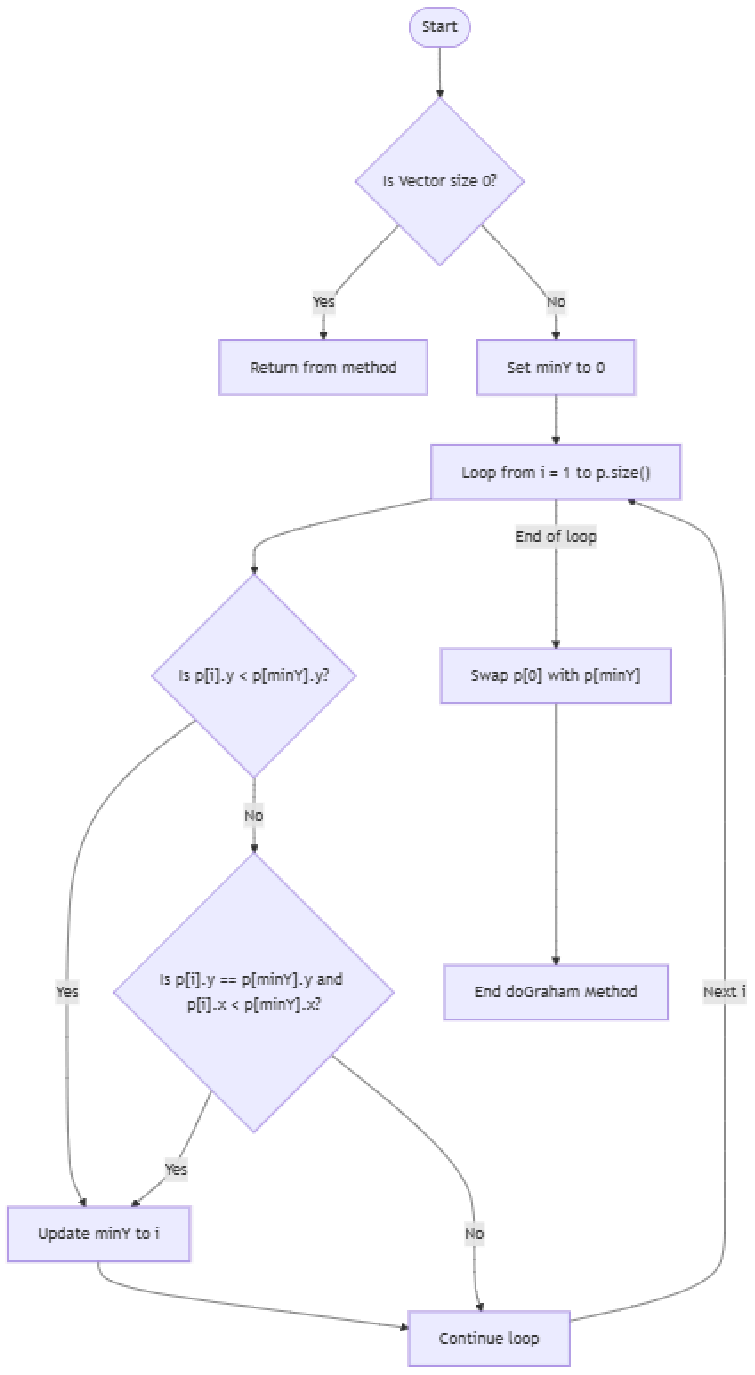
Meet Katharotiya - 202201157

Lab09

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.

```java
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);
    }
}
```

Below is the Control Flow graph of above code



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, it is essential to ensure that every line of code is executed at least once.

Test Cases:

- **Test 1:** The input vector `p` is empty (`p.size() == 0`).
  →**Expected Outcome:** The function should return immediately without executing any further code.
- **Test 2:** The input vector `p` contains at least one `Point` element.
  →**Expected Outcome:** The function should proceed to identify the `Point` with the minimum y-coordinate.

2. Branch Coverage

For branch coverage, it is necessary to confirm that each decision point (branch) in the code is evaluated as both true and false at least once.

Test Cases:

- **Test 1:** The vector `p` is empty (`p.size() == 0`).
  →**Expected Outcome:** The condition `if (p.size() == 0)` evaluates to true, leading the function to return immediately.

- **Test 2:** The vector p contains a single `Point` (e.g., `Point(0, 0)`).
  →**Expected Outcome:** The function should continue to the subsequent steps, evaluating additional conditions.
- **Test 3:** The vector p consists of multiple `Points`, where none have a smaller y-coordinate than `p[0]`.
  Example: `p = [Point(0, 0), Point(1, 1), Point(2, 2)]`
  →**Expected Outcome:** The condition inside the for loop remains false, causing `minY` to retain its value of 0.
- **Test 4:** The vector p contains multiple `Points`, with at least one having a smaller y-coordinate than `p[0]`.
  Example: `p = [Point(2, 2), Point(1, 0), Point(0, 3)]`
  →**Expected Outcome:** The condition within the for loop evaluates to true, resulting in an update of `minY`.

3. Basic Condition Coverage

To achieve basic condition coverage, each condition within the branches must be tested independently.

Test Cases:

- **Test 1:** The input vector p is empty (`p.size() == 0`).
  →**Expected Outcome:** The condition `if (p.size() == 0)` evaluates to true.
- **Test 2:** The input vector p is non-empty (`p.size() > 0`).
  →**Expected Outcome:** The condition `if (p.size() == 0)` evaluates to false.
- **Test 3:** Multiple points where only the condition `p.get(i).y < p.get(minY).y` is true.

Example: `p = [Point(1, 1), Point(0, 0), Point(2, 2)]`
→**Expected Outcome:** The condition `p.get(i).y < p.get(minY).y` is true, prompting an update to `minY`.

- **Test 4:** Multiple points where both `p.get(i).y == p.get(minY).y` and `p.get(i).x < p.get(minY).x` are true.
  Example: `p = [Point(1, 1), Point(0, 1), Point(2, 2)]`
  →**Expected Outcome:** Both conditions evaluate to true, leading to an update of `minY`.

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

1). Deletion Mutation Mutation:

→Remove the assignment of minY to 0 at the beginning of the method.

```
// Original

int minY = 0;

// Mutated - deleted the assignment

// int minY;
```

Impact: This mutation causes minY to be uninitialized when accessed, resulting in undefined behavior. Your test cases do not check for proper initialization, which may lead to undetected faults.

2)Insertion Mutation

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

```
// Original

for (int i = 1; i < p.size(); i++) {

    }

}

// Mutated - incorrectly overriding minY

for (int i = 1; i < p.size(); i++) {

    }

    // Inserting an incorrect override of minY

    minY = 1; // This line incorrectly sets minY to 1

}
```

3). Modification Mutation

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

```
// Original

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;

    }

}

// Mutated - changing || to &&

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;

    }}
```

Analysing Detection by Test Cases

1. **Statement Coverage:**
   - The removal of the initialization of `minY` may go undetected, as it might not trigger a direct exception depending on the surrounding logic.
2. **Branch Coverage:**
   - The insertion that sets `minY` to 1 could result in incorrect outcomes. However, if there are no tests specifically verifying the positions of the points after execution, this issue might remain unnoticed.
3. **Basic Condition Coverage:**
   - Modifying the logical operator from `||` to && does not lead to a crash, and since the test cases do not assess whether `minY` updates correctly under these new conditions, this mutation may also go undetected.

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

Test Set for Path Coverage Criterion

Test Case 1: Loop Explored Zero Times

- **Input:** An empty vector p.

Test:
Expected Result: The method should return immediately without executing any further processing. This scenario addresses the condition where the vector size is zero, leading to the exit condition of the method.

Test Case 2: Loop Explored Once

- **Input:** A vector containing one point.

Test:

- **Expected Result:** The method should not enter the loop since `p.size()` is 1. It should perform a swap of the first point with itself, effectively leaving the vector unchanged. This case tests the scenario where the loop iterates once.

Test Case 3: Loop Explored Twice

- **Input:** A vector with two points where the first point has a higher y-coordinate than the second.

Test:

- **Expected Result:** The method should enter the loop, compare the two points, and find that the second point has a lower y-coordinate. Consequently, `minY` should be updated to 1, leading to a swap that moves the second point to the front of the vector.

Test Case 4: Loop Explored More Than Twice

- **Input:** A vector with multiple points.

Test:

- **Expected Result:** The loop should iterate through all three points. The second point has the lowest y-coordinate, causing `minY` to be updated to 1. The swap will position the point with coordinates (1, 0) at the front of the vector.

Lab Execution

Q1). After generating the control flow graph, check whether your CFG matches with the CFG generated by Control Flow Graph Factory Tool and Eclipse flow graph generator.

→Control Flow Graph Factory :- YES

Q2).Devise minimum number of test cases required to cover the code using the aforementioned criteria.

→ Statement Coverage: 3 test cases

→Branch Coverage: 3 test cases

→Basic Condition Coverage: 3 test cases

→Path Coverage: 3 test cases

Summary of Minimum Test Cases:

Total: 3 (Statement) + 3 (Branch) + 2 (Basic Condition) + 3 (Path) = 11 test cases

Q3) and Q4) Same as Part I