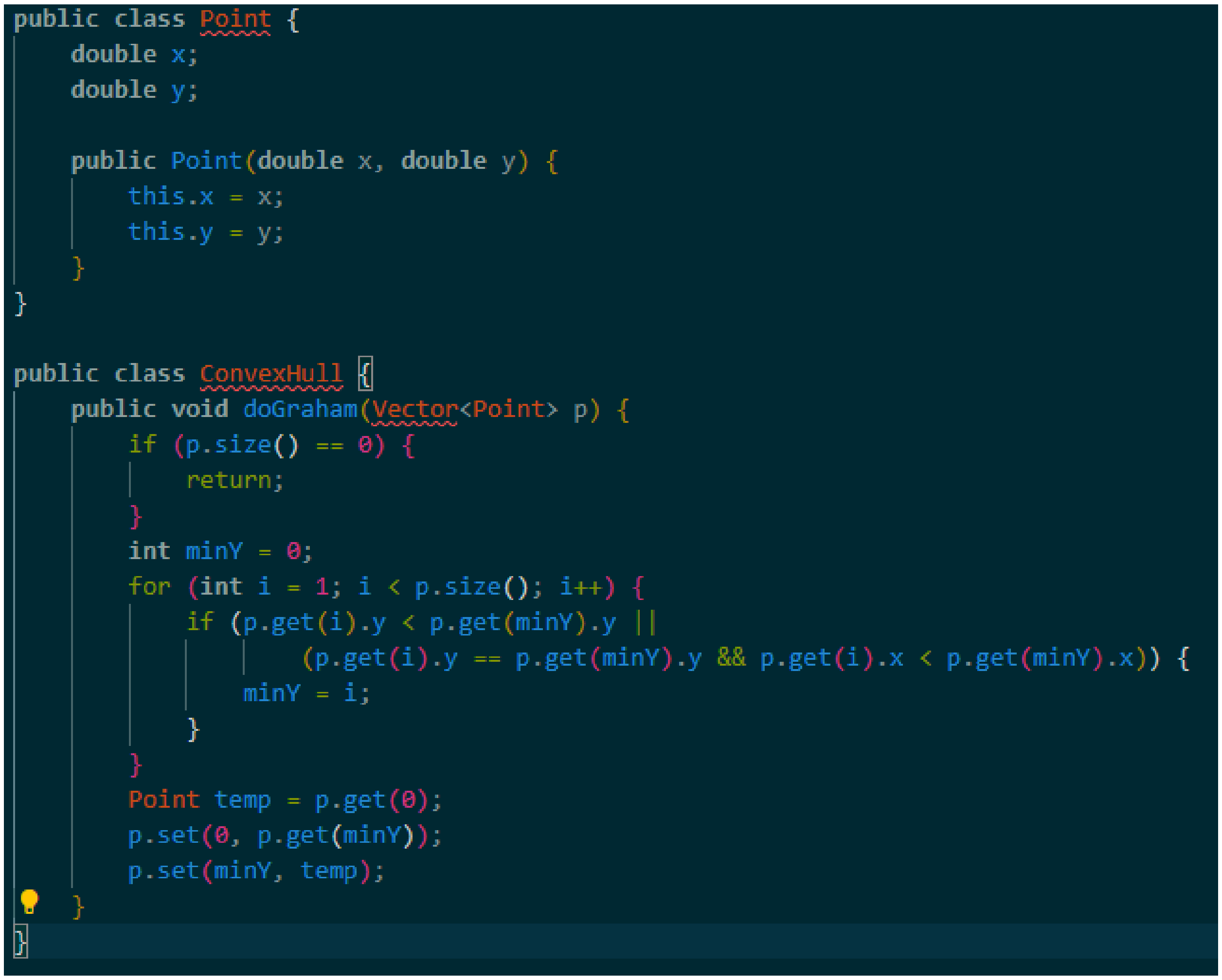
Lab 9

Software Engineering

Meet Andharia, 202201145

5th November, 2024

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.**

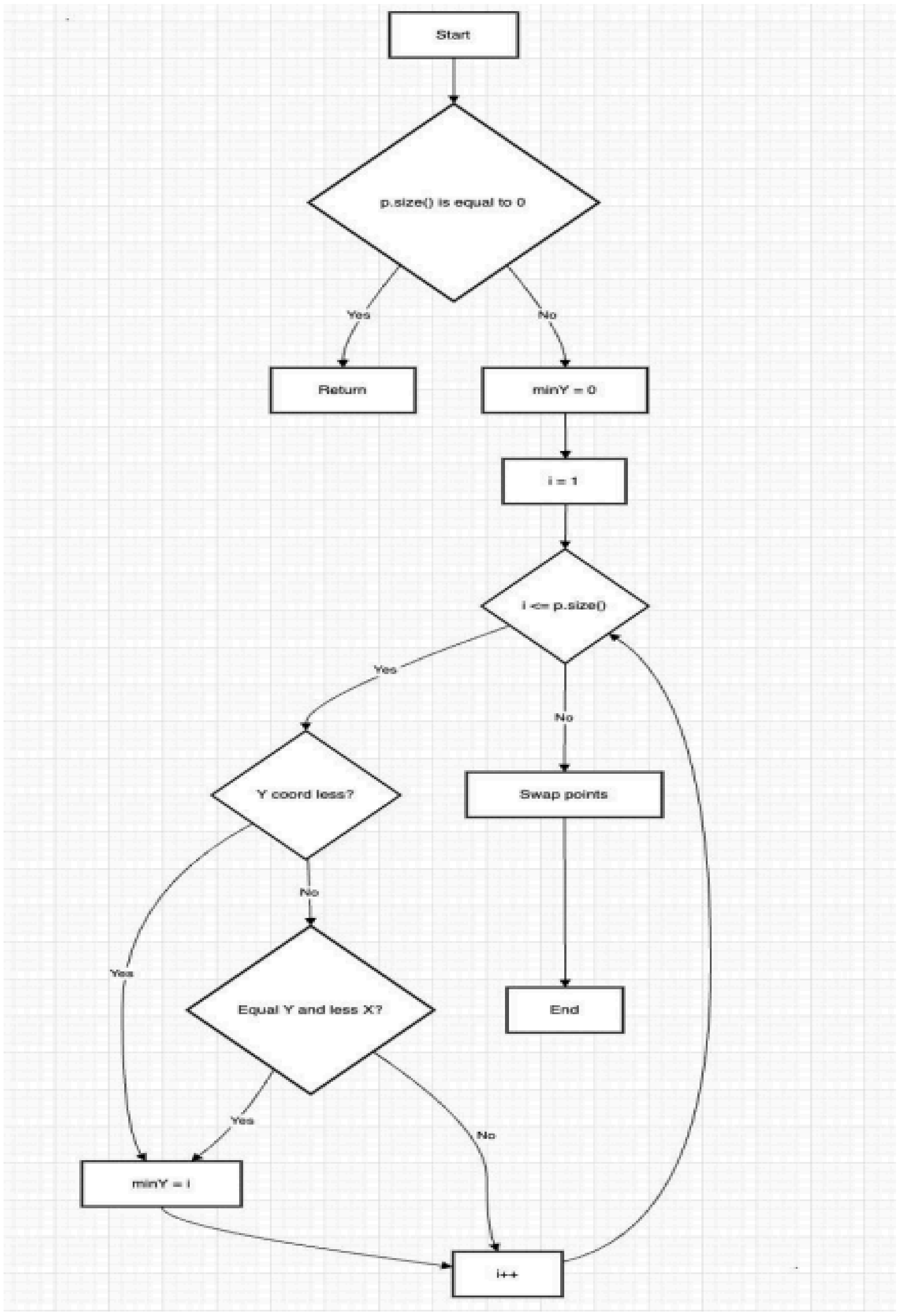


```java
public class Point {
    double x;
    double y;

    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



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.**

<u>1. Statement Coverage:</u>

We must make sure that every line of code is run at least once in order to attain statement coverage.

Test Case:

● **Test 1**: p.size() == 0 indicates that the input vector is empty.

Expected Result: The function ought to return right away without carrying out any additional actions.

● **Test 2**: There is at least one Point element in input vector p.

Expected Result: The function ought to continue locating which point has the lowest y-coordinate.

<u>2. Branch Coverage:</u>

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 Case:

● **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:

To ensure basic condition coverage, we need to test each condition within the program's branches independently. This involves verifying both the true and false outcomes of each condition in isolation. Below are test cases designed to achieve this, along with expected results:

Test Cases:

Test 1: Empty vector p (i.e., p.size() == 0)

Objective: Check if the condition if (p.size() == 0) evaluates to true when the vector is empty.

Expected result: The condition p.size() == 0 is true, so the corresponding branch of the code is executed.

Test 2: Non-empty vector p (i.e., p.size() > 0)

Objective: Ensure that the condition if (p.size() == 0) evaluates to false when the vector is not empty.

Expected result: The condition p.size() == 0 is false, so the alternate branch is executed.

Test 3: Multiple points where the condition p.get(i).y < p.get(minY).y is true

Example: p = [Point(1, 1), Point(0, 0), Point(2, 2)]

Objective: Test the condition p.get(i).y < p.get(minY).y to ensure it correctly identifies the minimum y value and updates minY accordingly.

Expected result: The condition p.get(i).y < p.get(minY).y is true for the second point Point(0, 0), so minY is updated to index 1 (corresponding to that point).

Test 4: Multiple points where the condition p.get(i).y == p.get(minY).y is true, and p.get(i).x < p.get(minY).x is also true

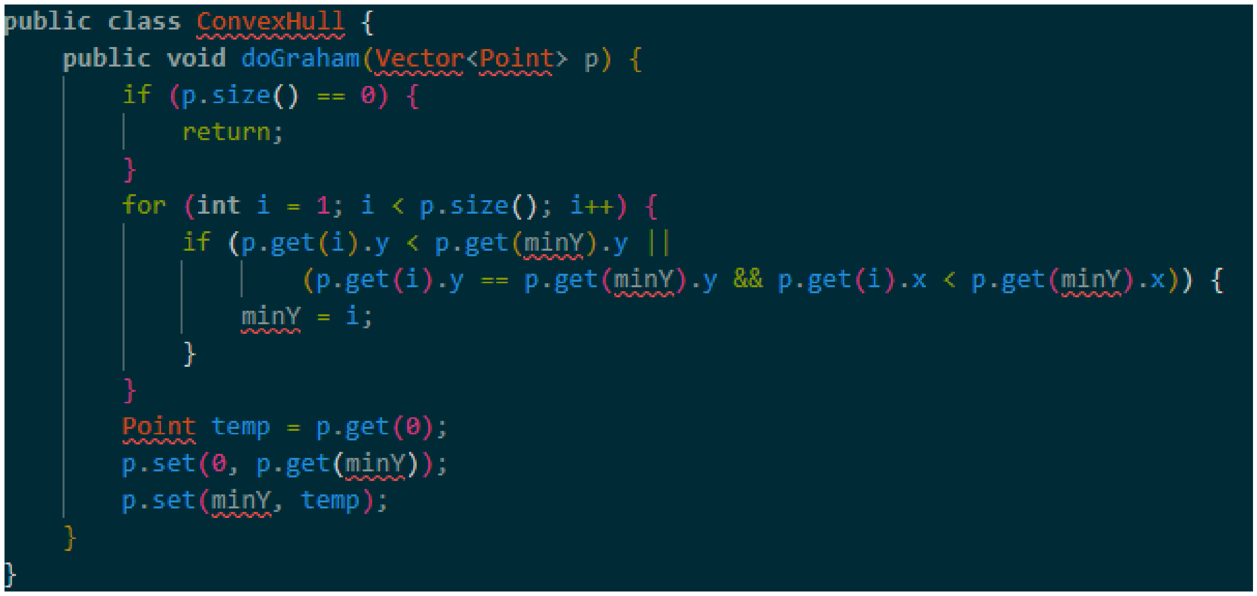
Example: p = [Point(1, 1), Point(0, 1), Point(2, 2)]

Objective: Check that when multiple points have the same y value, the code correctly compares x values to identify the smallest x value.

Expected result: Both conditions p.get(i).y == p.get(minY).y and p.get(i).x < p.get(minY).x are true for the second point Point(0, 1), so minY is updated to index 1 (since 0 < 1).

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.**

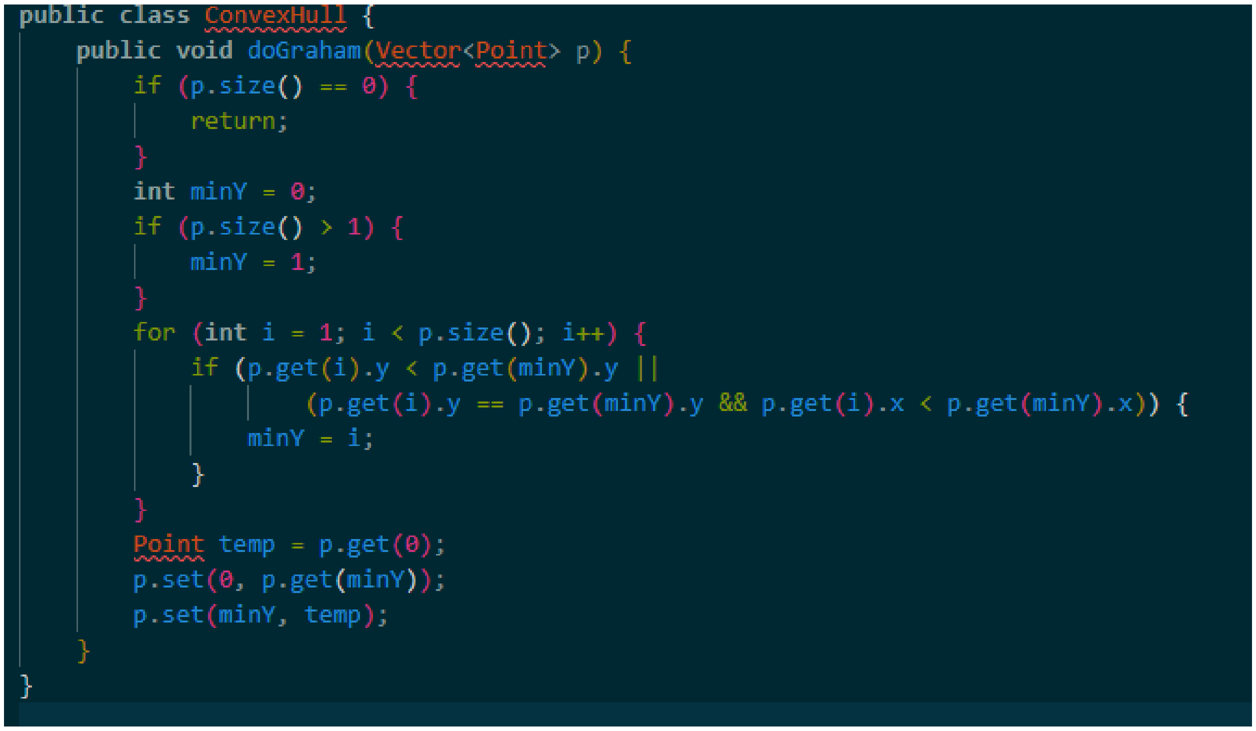
1) Deletion Mutation: Remove the assignment of minY to 0 at the beginning of the method.



```
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);
    }
}
```

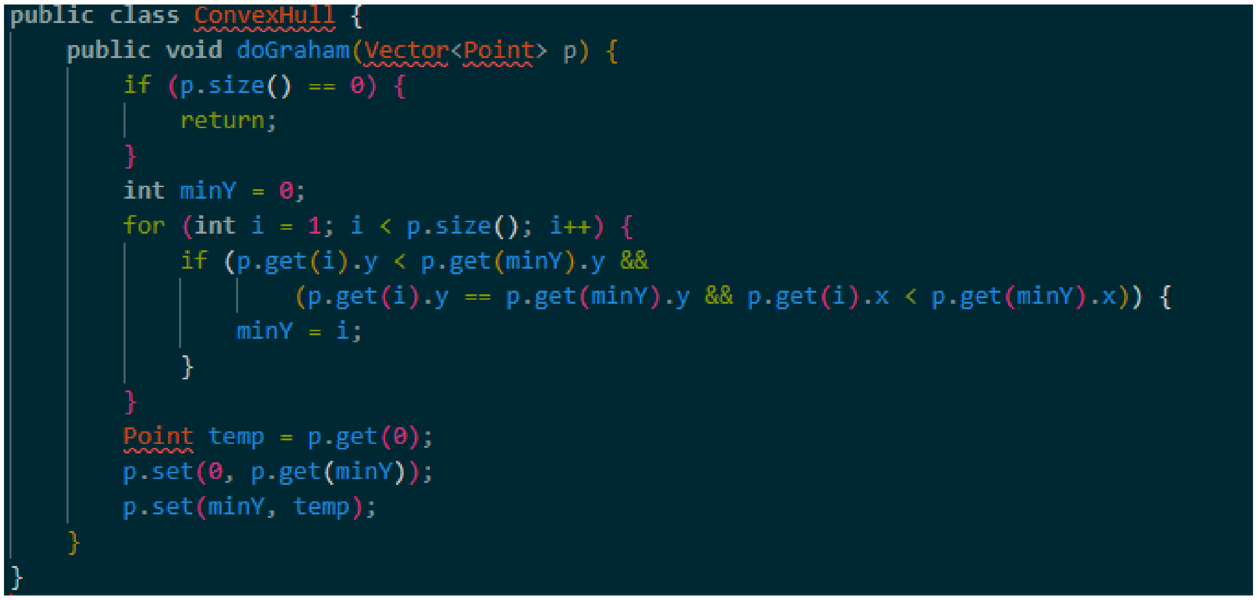
Impact: This mutation causes minY to be uninitialized when accessed, resulting in undefined behavior. Your test cases do not check for proper initialization, which may lead to undetected faults.

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



```
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: 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:

Statement Coverage:

- Potential Issue: The omission of initializing `minY` might not be detected during testing, as it may not trigger a direct error or exception. This could occur if the surrounding logic does not rely on this variable being properly initialized, meaning the issue could go unnoticed during testing.

Branch Coverage:

- Potential Issue: Modifying the code to force to 1 could lead to incorrect results, but if there are no specific tests that check the positions of points after the code executes, this issue might not be identified. In particular, the test suite may not fully verify the correctness of the `minY` value after changes to the logic.

Basic Condition Coverage:

- Potential Issue: Altering the logical operator from `||` to && may not result in a crash or obvious error. However, the test cases might not validate whether `minY` is updated correctly under these new conditions. As a result, the effect of this change may go undetected because the tests do not specifically check if the condition changes impact the correctness of the `minY` value.

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 Not Executed

Input: An empty vector p.

Test: Vector<Point> p = new Vector<Point>();

Expected result: The method should terminate immediately without entering the loop. This scenario tests the case where the vector has no elements, causing the loop to be skipped entirely due to the vector size being zero.

Test Case 2: Loop Iterates Once

Input: A vector with a single point.

Test: Vector<Point> p = new Vector<Point>(); p.add(new Point(0, 0));

Expected result: The method should not enter the loop because the vector size is 1. The first point should remain in place after a redundant swap, leaving the vector unchanged. This test case verifies behavior when the loop runs only once.

Test Case 3: Loop Iterates Twice

Input: A vector containing two points, where the first has a higher y-coordinate than the second.

Test: Vector<Point> p = new Vector<Point>(); p.add(new Point(1, 1)); p.add(new Point(0, 0));

Expected result: The method will enter the loop, compare the two points, and find that the second point has a lower y-coordinate. As a result, the minY index will be updated to 1, and the points will be swapped, placing the second point at the front of the vector. This test covers the case where the loop runs twice.

Test Case 4: Loop Iterates Multiple Times

Input: A vector with several points.

Test: Vector<Point> p = new Vector<Point>(); p.add(new Point(2, 2)); p.add(new Point(1, 0)); p.add(new Point(0, 3));

Expected result: The loop should iterate through all three points. During the iterations, the second point will have the smallest y-coordinate, so minY will be updated to 1. After the swap, the second point, with coordinates (1, 0), should move to the front of the vector.

This test case ensures the loop runs multiple times and performs correct updates.

<u>Lab Execution:</u>

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.

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.

Statement Coverage: 3 test cases

Branch Coverage: 4 test cases

Basic Condition Coverage: 4 test cases

Path Coverage: 3 test cases

Summary of Minimum Test Cases:

Total: 3 (Statement) + 4 (Branch) + 4 (Basic Condition) + 3 (Path)

= 14 test cases

Q3) and Q4) Same as Part I