



Name : Yash Tarpara

ID : 202201422

Lab : 09 Mutation Testing

Course : IT-314 Software Engineering

Instructor : Prof. Saurabh Tiwari

---

```

Vector doGraham(Vector p) {
    int i,j,min,M;

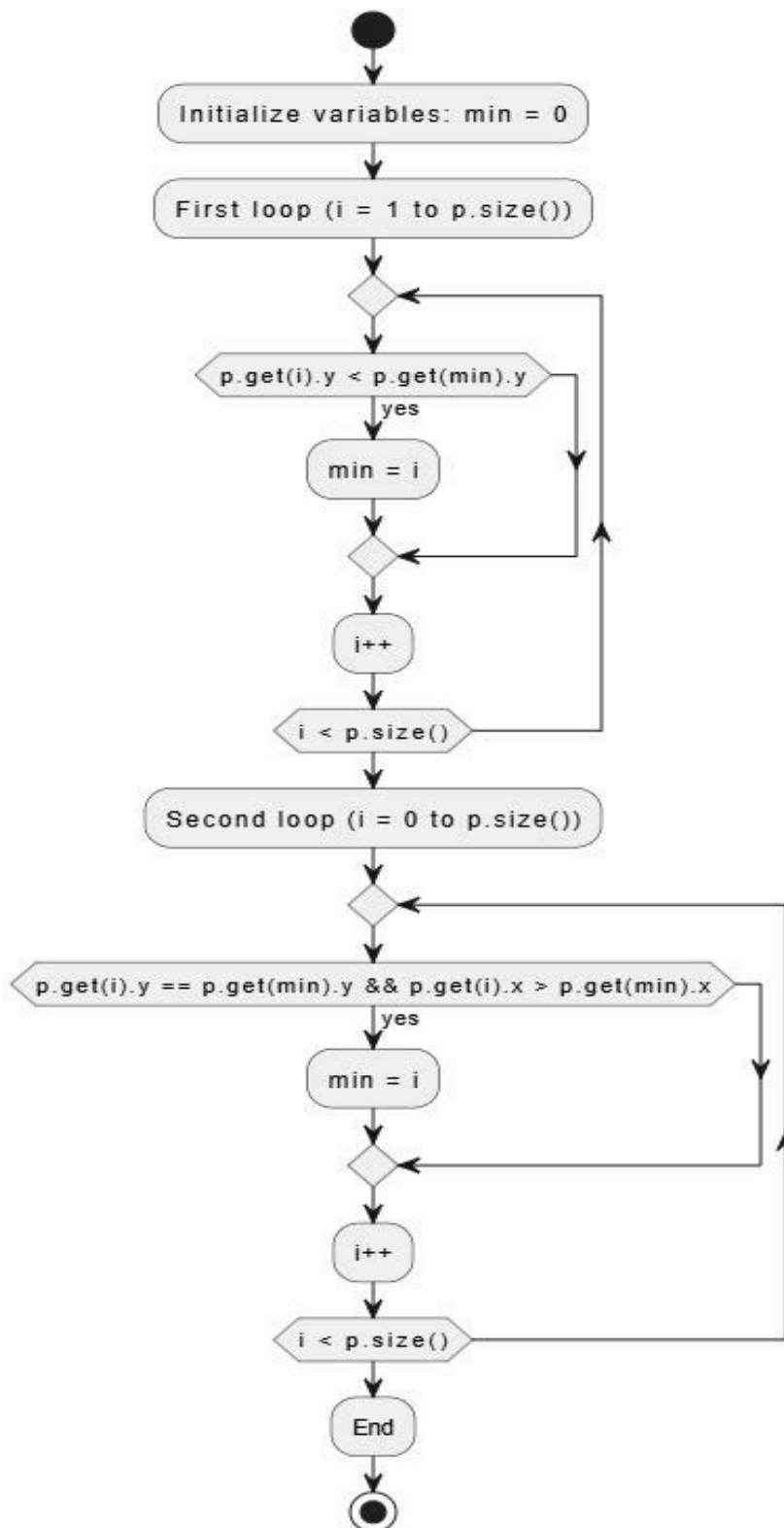
    Point t;
    min = 0;

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

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

```

## Control Flow Graph:



**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.**

**Ans.**

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

**Ans.1). 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 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

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. For Statement Coverage:

- The deletion of the initialization of `minY` might not be detected as it may not cause a direct exception depending on the surrounding logic.

### 2. For Branch Coverage:

- The insertion that forces `minY` to `1` may lead to incorrect outcomes, but if no tests specifically check the positions of points after the execution, it may go unnoticed.

### 3. For Basic Condition Coverage:

- Changing the logical operator from `||` to `&&` does not cause a crash, and the test cases do not validate the correctness of `minY` updating under these conditions, so it may not be detected.

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

**Ans.** Test Case 1: Loop Explored Zero Times

- Input: An empty vector `p`.
- Test: `Vector<Point> p = new Vector<Point>();`
- Expected Result: The method should return immediately without any processing. This covers 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 with one point.
- Test: `Vector<Point> p = new Vector<Point>(); p.add(newPoint(0, 0));`
- Expected Result: The method should not enter the loop since `p.size()` is 1. It should swap the first point with itself, effectively leaving the vector unchanged. This test case covers 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: `Vector<Point> p = new Vector<Point>(); p.add(newPoint(1, 1)); p.add(new Point(0, 0));`
- Expected Result: The method should enter the loop and compare the two points, finding that the second point has a lower y-coordinate. Thus, `minY` should be updated to 1, and a

swap should occur, moving the second point to the front of the vector.

Test Case 4: Loop Explored More Than Twice

- Input: A vector with multiple 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. The second point will have the lowest y-coordinate, so `minY` will be updated to 1. The swap will place 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 match with the CFG generated by Control Flow Graph Factory Tool and Eclipse flow graph generator.**

**Ans. 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.**

**Ans. 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

Q3) and Q4) Same as Part I