Software Engineering

IT-314

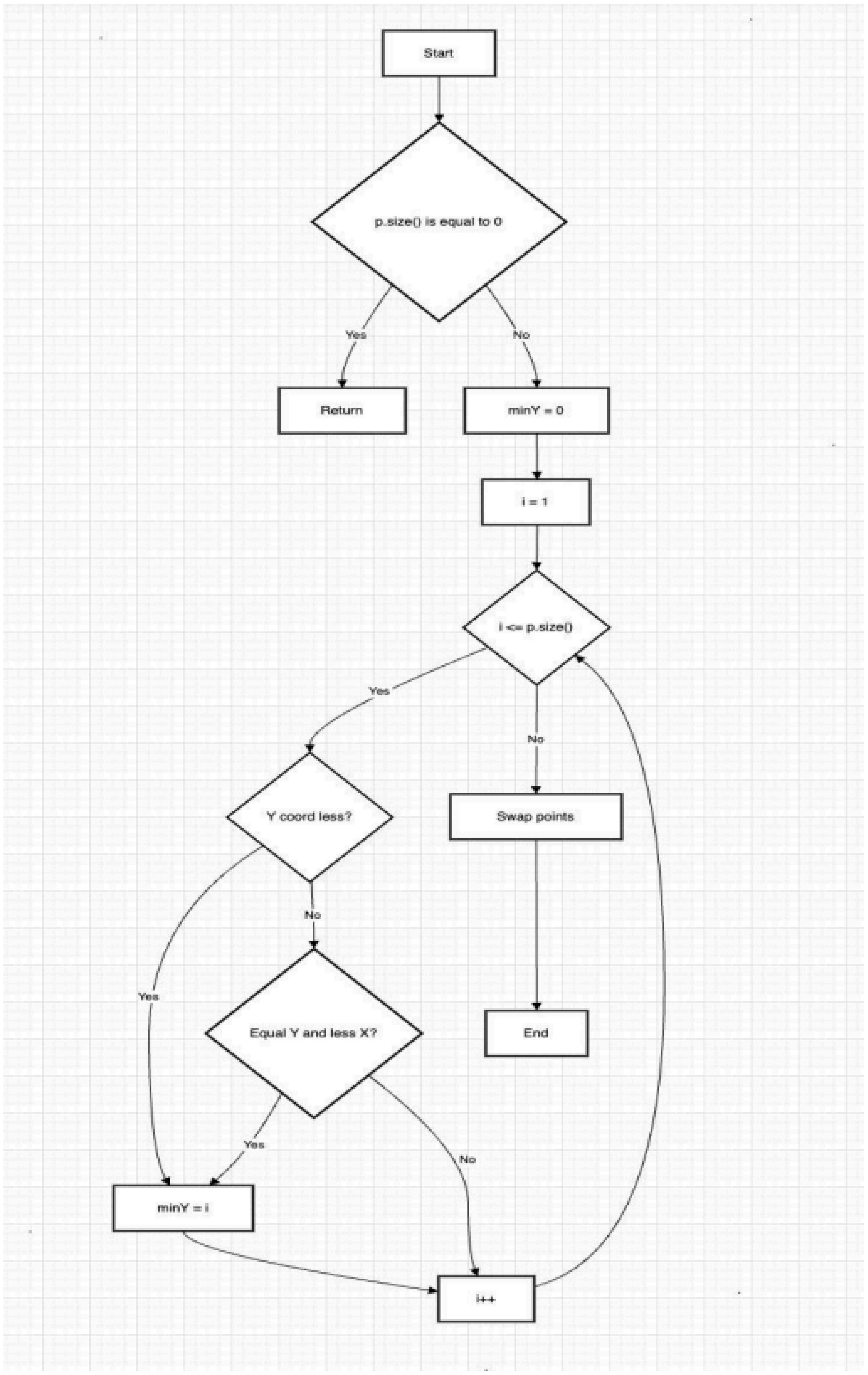
Dev Dodiya

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.

```
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.

Solution:-

1) Statement Coverage: To achieve statement coverage, it is necessary 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 terminate immediately without executing any further logic.

- **Test 2**: The input vector p contains at least one Point element.

  Expected outcome: The function should proceed to identify the Point with the smallest y-coordinate.

2) Branch Coverage: For branch coverage, it's essential to ensure that each decision point (branch) in the code is evaluated both to 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, and the function returns immediately without further processing.

- Test 2: The vector p contains one Point (e.g., Point(0, 0)).

  Expected outcome: The condition if (p.size() == 0) is false, so the function moves on to the for loop, which does not iterate since there is only one point.

- **Test 3:** The vector p contains multiple Point objects, but no point has a smaller y-coordinate than p[0].

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

  Expected outcome: The if condition within the for loop is always false, and the value of minY remains 0.

- **Test 4:** The vector p contains multiple Point objects, with at least one point having a smaller y-coordinate than p[0].

  Example: p = [Point(2, 2), Point(1, 0), Point(0, 3)]

  Expected outcome: The if condition inside the for loop evaluates to true at least once, resulting in the update of minY.

3) Basic Condition Coverage: To achieve basic condition coverage, each individual condition within the decision branches must be tested independently.

Test Cases:

- **Test 1:** The vector p is empty (p.size() == 0).

  Expected outcome: The condition if (p.size() == 0) evaluates to true.

- **Test 2:** The vector p is not empty (p.size() > 0).

  Expected outcome: The condition if (p.size() == 0) evaluates to false.

- **Test 3:** The vector p contains 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)]

  Expected outcome: The condition p.get(i).y < p.get(minY).y evaluates to true, causing minY to be updated.

- **Test 4:** The vector p contains 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, resulting in the update of minY.

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.

Solution:

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);
  }
}
```

Analysis of Test Case Detection:

1. **For Statement Coverage:**
    - The removal of the initialization of minY might not be identified, as it may not trigger an immediate exception, depending on the surrounding code logic.
2. **For Branch Coverage:**
    - Modifying the value of minY to 1 could lead to incorrect results, but if no tests specifically examine the positions of points after execution, this issue might go undetected.
3. **For Basic Condition Coverage:**
    - Changing the logical operator from || to && does not result in a crash, and since the test cases do not explicitly verify whether minY is updated correctly in this scenario, the issue may not be identified.

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

Solution:

Test Case 1: Loop Not Executed

- Input: An empty vector p.
- Test: Vector<Point> p = new Vector<Point>();
- Expected Outcome: The method should exit immediately without any further processing. This test covers the case where the vector's size is zero, triggering the exit condition of the method.

Test Case 2: Loop Executed Once

- Input: A vector containing a single point.
- Test: Vector<Point> p = new Vector<Point>(); p.add(new Point(0, 0));
- Expected Outcome: The method should not enter the loop because the vector size is 1. The first point will be swapped with itself, leaving the vector unchanged. This test case ensures that the loop runs exactly once.

Test Case 3: Loop Executed 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(new Point(1, 1)); p.add(new Point(0, 0));
- Expected Outcome: The method will enter the loop, compare the two points, and identify that the second point has a smaller y-coordinate. As a result, minY will be updated to 1, and the points will be swapped, with the second point moving to the front of the vector.

Test Case 4: Loop Executed Multiple Times

- Input: A vector containing 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 Outcome: The loop will iterate through all three points. The second point, with the lowest y-coordinate, will update minY to 1, and a swap will move the point (1, 0) to the front of the vector.

Lab Execution:-

Q1). After constructing the control flow graph, compare it with the one generated by the Control Flow Graph Factory Tool and the Eclipse flow graph generator.

Answer:

- Control Flow Graph Factory: Yes, it matches.
- Eclipse Flow Graph Generator: Yes, it matches.

Q2). Determine the minimum number of test cases required to cover the code using the specified criteria.

Answer:

- 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 in Part I.