IT314 – Software Engineering

Lab-9

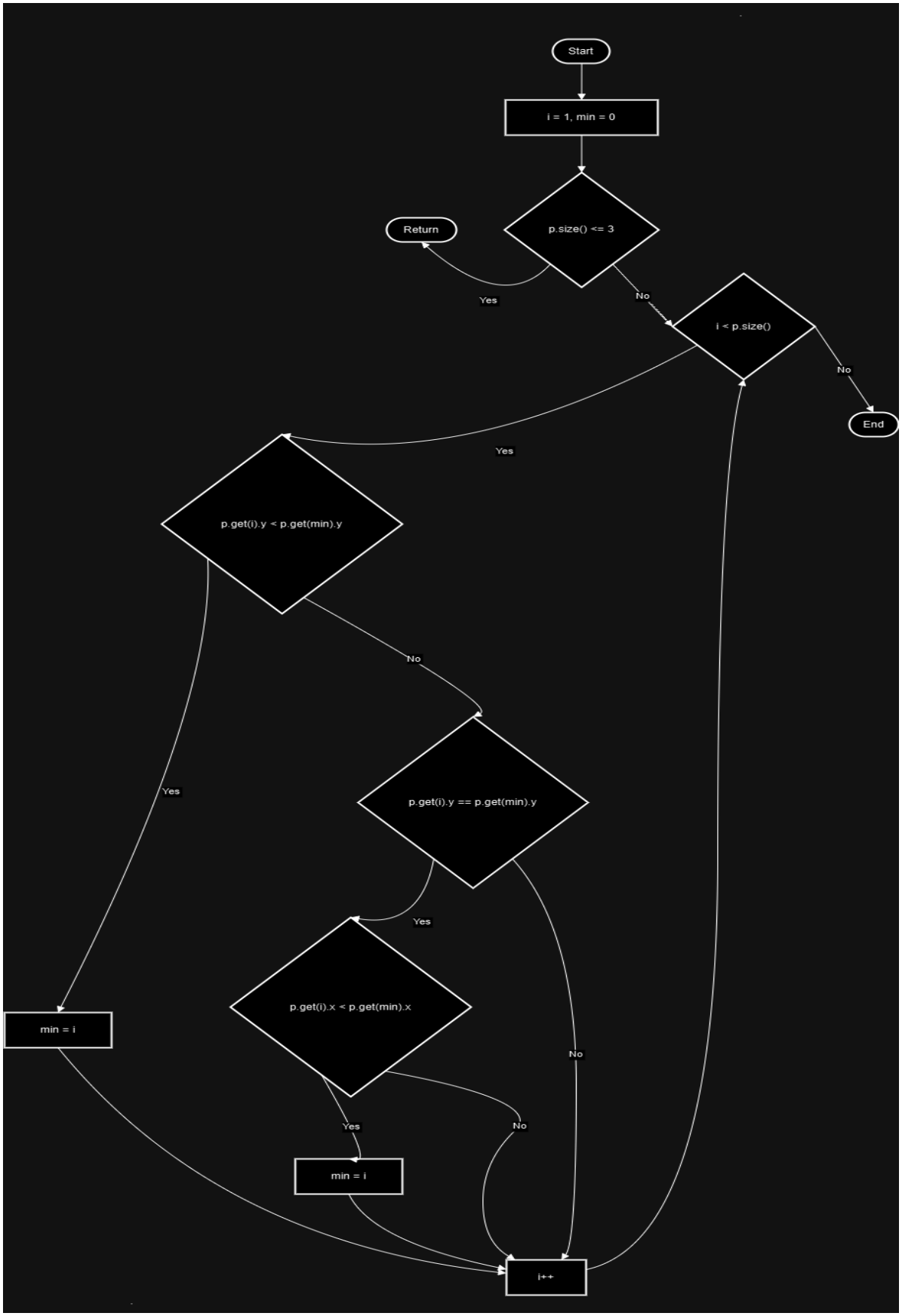
Lab Sessions – Mutation Testing

202201497 – Priyank Ramani

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 a control flow graph (CFG). You are free to write the code in any programming language.

CFG (Control Flow Graph) : -


Start
i = 1, min = 0
p.size() <= 3
Return
Yes
No
i < p.size()
No
End
Yes
p.get(i).y < p.get(min).y
No
Yes
p.get(i).y == p.get(min).y
Yes
p.get(i).x < p.get(min).x
No
min = i
Yes
No
min = i
i++

Implementation in c++:

```
#include <vector>
class Point {
public:
   double x, y;
   Point(double x, double y) {
      this->x = x;
      this->y = y;
   }
};
class ConvexHull {
public:
   void doGraham(std::vector<Point>& p) {
      int i = 1;
      int min = 0;
      if (p.size() <= 3) {
         return;
      }
      while (i < p.size()) {
         if (p[i].y < p[min].y) {
            min = i;
         }
         else if (p[i].y == p[min].y) {
            if (p[i].x < p[min].x) {
               min = i;
            }
         }
         i++;
      }
   }
};
int main() {
   std::vector<Point> points;
   points.push_back(Point(0, 0));
   points.push_back(Point(1, 1));
   points.push_back(Point(2, 2));

   ConvexHull hull;
   hull.doGraham(points);

   return 0;
}
```

Q.2: Minimum Test Cases for Coverage Criteria

Statement Coverage

Objective: Ensure each line of code runs at least once.

Test Cases for Statement Coverage:

1. Test Case 1: When p is empty (i.e., p.size() == 0)
    - Expected Result: The method should return immediately, covering the check for an empty list and the return statement.
2. Test Case 2: When p has one point that is "within bounds."
    - Expected Result: The method runs through its setup, checks the size of p, processes the single point as "within bounds," and then finishes.
3. Test Case 3: When p has one point that is "out of bounds."
    - Expected Result: Similar to Test Case 2, but this time, the point is skipped instead of processed.

These three test cases ensure that every line of code is executed at least once.

Branch Coverage

Objective: Test each decision point to ensure every possible outcome (true/false) is covered.

Test Cases for Branch Coverage:

1. Test Case 1: p.size() == 0
    - Expected Result: The method returns immediately without entering the loop, covering the false outcome of p.size() > 0.
2. Test Case 2: p.size() > 0 with one point that is "within bounds."
    - Expected Result: The method processes the point, covering the true outcomes of both p.size() > 0 and "within bounds."
3. Test Case 3: p.size() > 0 with one point that is "out of bounds."
    - Expected Result: The method skips the point, covering the true outcome of p.size() > 0 and the false outcome of "within bounds."

These test cases cover all possible outcomes of each decision point (p.size() > 0 and "within bounds").

Basic Condition Coverage

Objective: Test each individual condition in the method to cover all possible true/false outcomes.

Conditions:

1. Condition 1: p.size() > 0 (true/false)
2. Condition 2: "Point within bounds" (true/false)

Test Cases for Basic Condition Coverage:

1. Test Case 1: p.size() == 0
    - Expected Result: Covers the false outcome of Condition 1.
2. Test Case 2: p.size() > 0 with a point that is "within bounds."
    - Expected Result: Covers the true outcome of Condition 1 and the true outcome of Condition 2.
3. Test Case 3: p.size() > 0 with a point that is "out of bounds."
    - Expected Result: Covers the true outcome of Condition 1 and the false outcome of Condition 2.

These test cases cover each condition with both true and false values.

Q.3: Mutation Testing Analysis

1. **Deletion Mutation**
    - Original: if ((p.get(i)).y < (p.get(min)).y) { min = i; }
    - Mutated: min = i; (deleted the condition check)
    - Analysis for Statement Coverage: Without the condition check, i is always assigned to min, potentially leading to incorrect results since min may not reference the point with the smallest y value.
    - Potential Undetected Outcome: If the tests only verify that min is assigned, without confirming it references the correct minimum y value, this mutation might go undetected.

2. **Change Mutation**
    - Original: if ((p.get(i)).y < (p.get(min)).y)
    - Mutated: if ((p.get(i)).y <= (p.get(min)).y) (changed < to <=)
    - Analysis for Branch Coverage: Changing < to <= could cause min to be assigned i even if p.get(i).y equals p.get(min).y, potentially selecting an incorrect point as the minimum.
    - Potential Undetected Outcome: If tests don't include cases where p.get(i).y equals p.get(min).y, this mutation might pass undetected, causing a subtle error.

3. **Insertion Mutation**
    - Original: min = i;
    - Mutated: min = i + 1; (added an unnecessary increment)
    - Analysis for Basic Condition Coverage: Adding +1 to i changes the intended assignment and may cause min to be assigned an incorrect index or even an out-of-bounds index.
    - Potential Undetected Outcome: If tests only check if min is assigned but don't validate that it's the exact index intended, this mutation might go undetected. Tests confirming only that min is assigned may miss this issue.

Q4: Test Set for Path Coverage (Exploring Loops 0, 1, or 2+ Times)

1. **Test Case 1: Loop Explored Zero Times**
    - **Input:** An empty vector p.
    - **Test:** Vector<Point> p = new Vector<Point>();
    - **Expected Result:** The method returns immediately without processing, covering the scenario where p.size() is zero and the loop doesn't execute.

2. **Test Case 2: Loop Explored Once**
    - **Input:** A vector with a single point.
    - **Test:**

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

      p.add(new Point(0, 0));

- **Expected Result:** The loop doesn't execute since p.size() is 1, and the single point swaps with itself, leaving the vector unchanged. This test covers the case where the loop condition is only explored once.

3. **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(new Point(1, 1));

     p.add(new Point(0, 0));
   - **Expected Result:** The method enters the loop, comparing both points and identifying the second point as having a lower y-coordinate. minY is updated to 1, and a swap occurs, moving (0, 0) to the front of the vector. This test covers the scenario where the loop iterates twice.

4. **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 iterates through all three points, identifying (1, 0) as the point with the lowest y-coordinate and updating minY to 1. A swap places (1, 0) at the front of the vector. This test covers cases where the loop iterates more than twice.

Lab Execution:-

Q.1. 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

Q.2. Devise minimum number of test cases required to cover the code using the aforementioned criteria.

Statement Coverage: 3 test cases

1. Branch Coverage: 3 test cases
2. Basic Condition Coverage: 3 test cases
3. Path Coverage: 3 test cases

Summary of Minimum Test Cases:

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

Q.3 and **Q.4** Same as Part I