Lab - 9
Mutation Testing

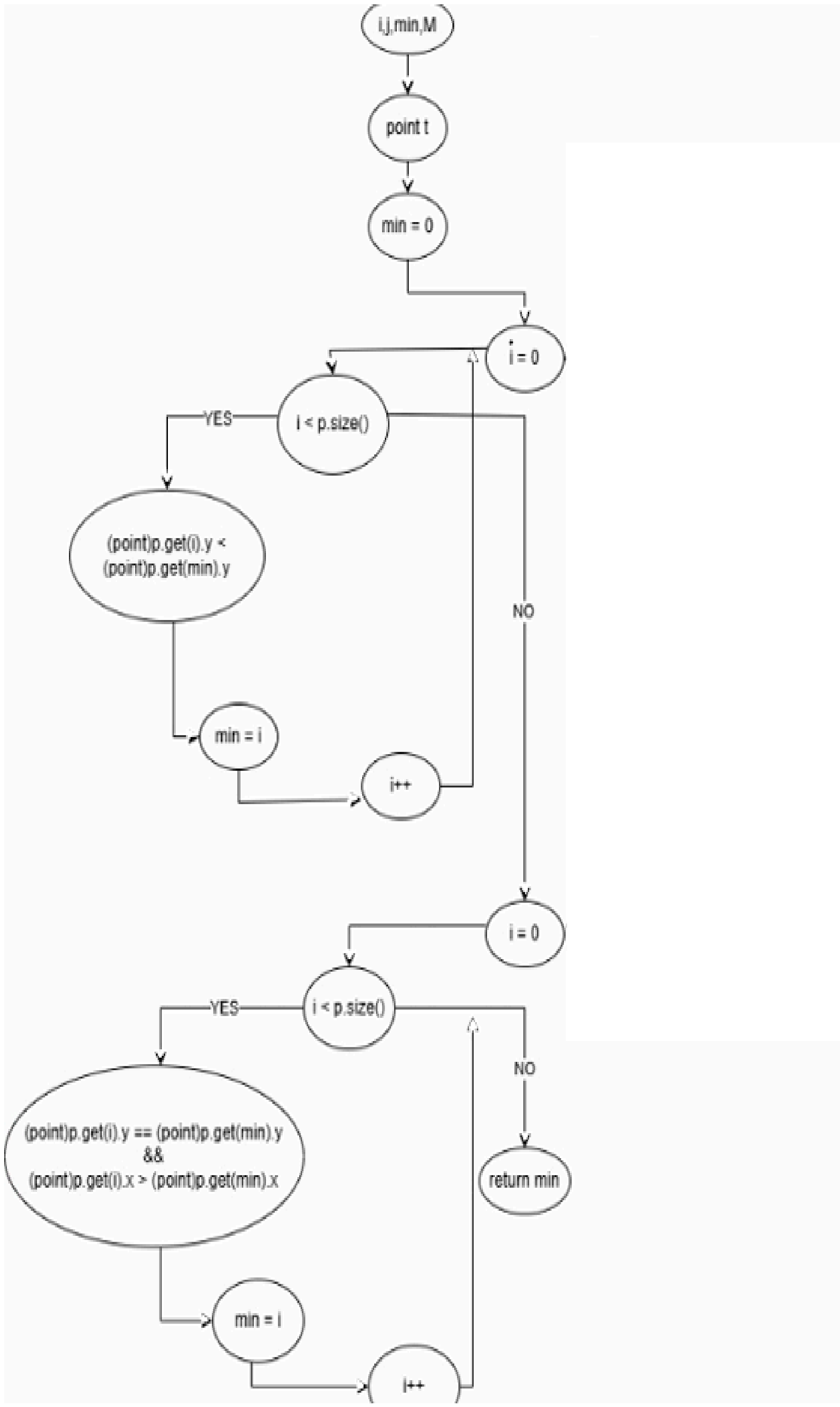
IT - 314
Software Engineering

202201215
Kamal Akbari

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.

```
#include <vector>
#include <algorithm>

struct Point {
    int x, y;
};

vector<Point> doGraham(std::vector<Point> p) {
    int min = 0;

    for (int i = 1; i < p.size(); ++i) {
        if (p[i].y < p[min].y) {
            min = i;
        }
    }


    for (int i = 0; i < p.size(); ++i) {
        if (p[i].y == p[min].y && p[i].x > p[min].x) {
            min = i;
        }
    }


    swap(p[0], p[min]);

    return p;
}
```

1. Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG)



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.

a. Statement Coverage

Test Cases for Statement Coverage

1. **Test Case 1:** `p` is an empty vector.
Expected Behavior: The method should return immediately without performing any operations.

2. **Test Case 2:** `p` contains a single point, e.g., `[(0, 0)]`.
Expected Behavior: No swap is needed as there's only one point, but all statements should be executed.

3. **Test Case 3:** `p` contains multiple points with distinct y-values, e.g., `[(1, 3), (2, 2), (3, 4)]`.
Expected Behavior: The method should identify the point `(2, 2)` as the minimum and swap it with the point at index 0.

4. **Test Case 4:** `p` contains points with the same y-value but different x-values, e.g., `[(3, 2), (1, 2), (2, 2)]`.
Expected Behavior: Among the points with the same y-value, the one with the smallest x-value `(1, 2)` should be identified as the minimum and swapped with the first point.

b. Branch Coverage

Test Cases for Branch Coverage

1. **Test Case 1:** `p` is an empty vector.Branch Tested: `if (p.size() == 0)`

2. **Test Case 3:** Multiple points with distinct y-values, e.g., `[(1, 3), (2, 2), (3, 4)]`.

Branch Tested: `p.get(i).y < p.get(minY).y` in the loop.

3. Test Case 4: Points with the same y-value but different x-values, e.g., `[(3, 2), (1, 2), (2, 2)]`.
Branch Tested: `(p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x)`

c. Basic Condition Coverage

Test Cases for Basic Condition Coverage

1. Test Case 3: Distinct y-values, e.g., `[(1, 3), (2, 2), (3, 4)]`.
Conditions Tested: `p.get(i).y < p.get(minY).y` (true and false).

2. Test Case 4: Same y-values, varying x-values, e.g., `[(3, 2), (1, 2), (2, 2)]`.
Conditions Tested: `p.get(i).y == p.get(minY).y` (true and false) and `p.get(i).x < p.get(minY).x` (true and false).

Summary of Test Cases :

Test Case 1

- Input Points (Vector p): `[]`
- Expected Behavior: Method returns immediately.
- Potential Mutation: Change the condition `if (p.size() == 0)` to `if (p.size() != 0)`.
- Mutation Effect: The method will not return immediately, leading to incorrect behavior.

Test Case 2

- Input Points (Vector p): `[(0, 0)]`
- Expected Behavior: No swap; single point remains unchanged.
- Potential Mutation: Remove the check for single-point case.
- Mutation Effect: The method would attempt to perform unnecessary operations.

Test Case 3

- Input Points (Vector p): `[(1, 3), (2, 2), (3, 4)]`
- Expected Behavior: Point `(2, 2)` is swapped with `(1, 3)`.
- Potential Mutation: Invert the condition `if (p.get(i).y < p.get(minY).y)` to `if (p.get(i).y >= p.get(minY).y)`.
- Mutation Effect: The wrong point would be identified as the minimum, causing incorrect swapping.

Test Case 4

- Input Points (Vector p): `[(3, 2), (1, 2), (2, 2)]`
- Expected Behavior: Point `(1, 2)` is swapped with `(3, 2)`.
- Potential Mutation: Invert the condition `if (p.get(i).x < p.get(minY).x)` to `if (p.get(i).x >= p.get(minY).x)`.
- Mutation Effect: The wrong point with the same y-value would be identified as the minimum

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.

a. Deletion Mutation:
// Original

if p[i].y < p[min_idx].y:
min_idx = i
// Mutated - deleted the condition check
min_idx = i

Analysis for Statement Coverage:

- **Impact of Removing Condition Check:** If the condition check is removed, the code will always assign \`i\` to \`min\`, potentially leading to an incorrect result. However, this issue might not be detected if the test cases only verify that `min` is assigned without specifically validating the correct selection of the minimum `y` value.

- **Risk of Undetected Errors:** If the test set only checks that `min` has been assigned, without confirming that the assigned value is indeed the correct minimum, this error may go unnoticed.

a. Change Mutation:

// Original

if p[i].y < p[min_idx].y:

// Mutated - changed < to <=
if p[i].y <= p[min_idx].y:

Analysis for Branch Coverage:

- Changing < to <= could cause the code to mistakenly assign min = i even if p.[i].y equals p.[min_idx].y, potentially selecting an incorrect point as the minimum.
- Potential Undetected Outcome: If the test set does not specifically validate cases where p.[i].y equals p.[min_idx].y, the mutation could produce a subtle fault without detection.

b. Insertion Mutation:
// Original

min_idx = i

// Mutated - added unnecessary increment

min_idx = i+1

Analysis for Basic Condition Coverage:

- Adding an unnecessary increment (i + 1) changes the intended assignment, leading min to point to an incorrect index, potentially out of the array bounds.
- Potential Undetected Outcome: If the test set does not validate that min is correctly assigned to the expected index without additional increments, this mutation might not be detected. Tests only checking if min is assigned (rather than validating correctness) might miss this error.

4. 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(new Point(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(new Point(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.

<u>Lab Execution:-</u>

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.

Ans. Control Flow Graph Factory :- 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: 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