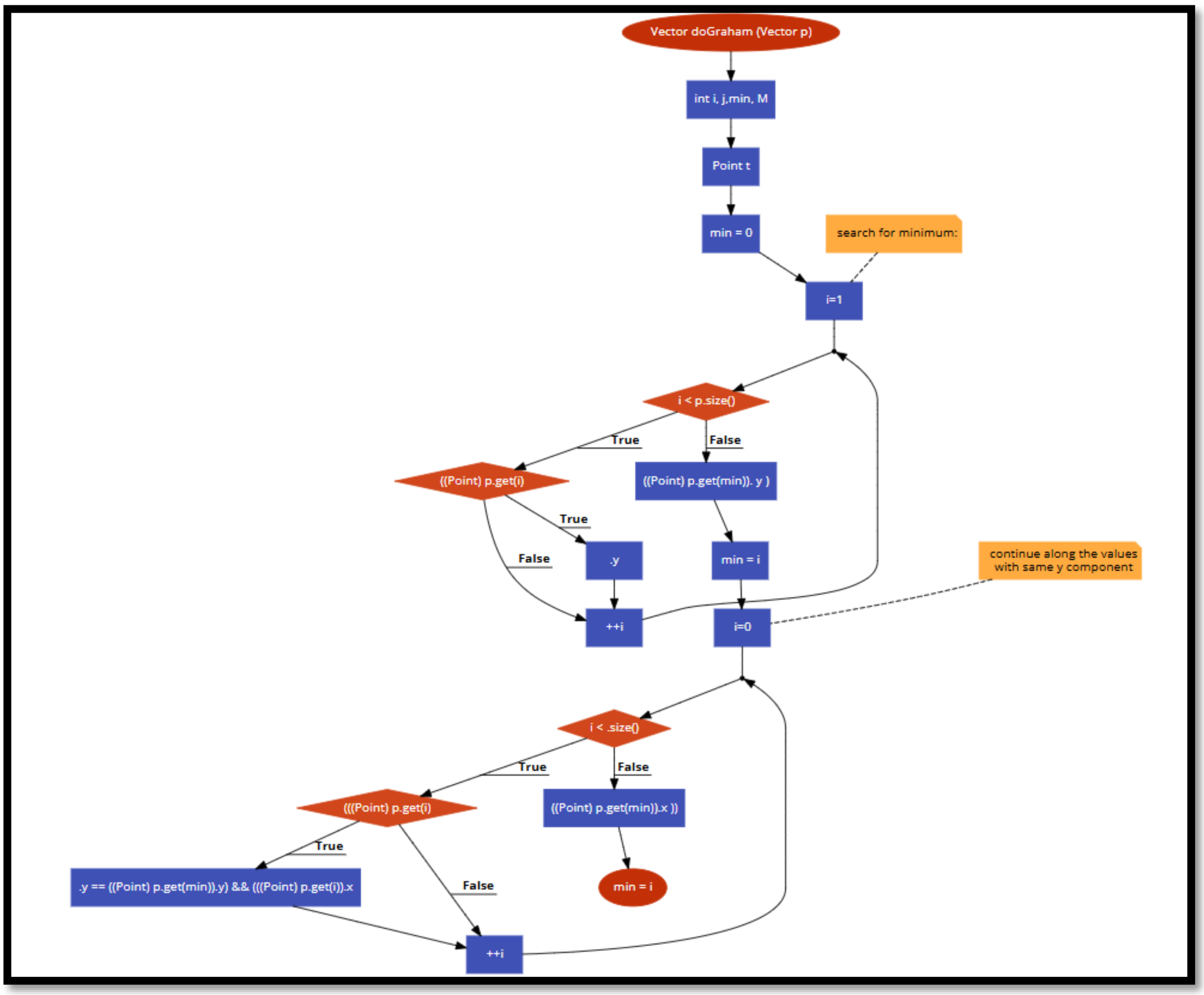
IT314 Software Engineering

Lab 9: Mutation Testing

Name: Rit Rajendra Trambadia **Id: 202201424**

➢ Control Flow Graph:



Vector doGraham (Vector p)
int i, j,min, M
Point t
min = 0
search for minimum:
i=1
i < p.size()
True
False
((Point) p.get(i)
((Point) p.get(min)). y)
True
False
.y
min = i
continue along the values with same y component
++i
i=0
i < .size()
True
False
(((Point) p.get(i)
((Point) p.get(min)).x))
True
.y == ((Point) p.get(min)).y) && (((Point) p.get(i)).x
False
min = i
++i

➢ Test Set Construction

1. Statement Coverage

To achieve **Statement Coverage**, every line of code must be executed at least once. This requires that:

- The Vector p has at least two points.
- The for loops execute at least one iteration.
- The condition inside each if statement is evaluated (even if it doesn't result in updating min).

Test Set:

1. p = [(0, 1), (1, 2)] (two points with different y values).
   - This will execute both for loops and cover all statements in the code.

2. Branch Coverage

For **Branch Coverage**, we need to ensure that each possible branch (true or false) in every conditional statement is tested.

In this code, we have the following branches:

- The if condition in the first for loop (((Point) p.get(i)).y < ((Point) p.get(min)).y):
  - **True**: The y of the current point is less than the y of min.
  - **False**: The y of the current point is not less than the y of min.
- The if condition in the second for loop (((Point) p.get(i)).y == ((Point) p.get(min)).y && ((Point) p.get(i)).x > ((Point) p.get(min)).x):
  - **True**: The y values are equal, and the x of the current point is greater than the x of min.
  - **False**: Either the y values are not equal, or the x of the current point is not greater.

Test Set:

1. p = [(0, 1), (1, 0)] — First point has y = 1, second point has y = 0.
   - In the first loop, min should update to index 1 (branch true).
   - In the second loop, min should not update (branch false for both conditions).
2. p = [(0, 0), (1, 0), (2, 0)] — All points have the same y = 0, but different x values.
   - In the first loop, min will not change from 0 (branch false).
   - In the second loop, min should update to the point with the highest x (branch true for x condition).

3. Basic Condition Coverage

For **Basic Condition Coverage**, each condition within a compound conditional statement (those using &&) must be tested with both true and false outcomes independently.

This includes:

- (p.get(i)).y < (p.get(min)).y in the first for loop.
    - True and False cases need to be tested.
- (p.get(i)).y == (p.get(min)).y and (p.get(i)).x > (p.get(min)).x in the second for loop.
    - Each of these sub-conditions needs to be true and false independently.

Test Set:

1. p = [(0, 1), (1, 0)]
    - Covers y comparison in the first loop with true and false cases.
2. p = [(0, 0), (1, 0), (2, 0)]
    - All points have y = 0, with different x values, covering the y equality and x comparison in the second loop independently.
3. p = [(0, 0), (1, 1), (2, 0)]
    - This will test:
        - First loop with both true (second point) and false outcomes for y < min.y.
        - Second loop where some points have equal y and others do not.

➢ Mutation Testing

1. Mutation by Deletion

Original Code Snippet:

```
if (((Point) p.get(i)).y < ((Point) p.get(min)).y) {
    min = i;
}
```

Mutation:

```
// Deleting the condition that finds the minimum y value
// if (((Point) p.get(i)).y < ((Point) p.get(min)).y) {
//     min = i;
// }
```

Impact of Deletion:

- By removing this condition, the code will no longer correctly identify the point with the minimum y value. Instead, min will remain as initialized (0). If the first point has a larger y value than others, this mutation will lead to incorrect results.

Test Cases Affected:

- This mutation may not be detected by the existing test cases, especially if the first point already has the minimum y value. Thus, all test cases with a unique minimum y would pass without triggering the fault.

Affected Test Cases:

- p = [(0, 1), (1, 0)]
    - Failure: The code will incorrectly select (0, 1) (first point) as the minimum, despite (1, 0) having a smaller y-coordinate.

2. Mutation by Insertion

Original Code Snippet:

```
for(i=1; i < p.size(); ++i) {
    if (((Point) p.get(i)).y < ((Point) p.get(min)).y) {
        min = i;
    }
}
```

Mutation:

```
for(i=1; i < p.size(); ++i) {
    // Inserting a condition that overrides the found minimum
    if (((Point) p.get(i)).y <= ((Point) p.get(min)).y) {
        min = i;
    }
    if (p.size() > 10) { // This condition does nothing in context
        min = 0; // Reset min arbitrarily when more than 10 points
    }
}
```

Impact of Insertion:

- The newly inserted condition (if (p.size() > 10) { min = 0; }) will cause min to reset to 0 if there are more than 10 points in the vector. This introduces a fault where the logic for finding the minimum point is overridden under specific conditions, leading to incorrect behaviour when many points are passed.

Test Cases Affected:

- The existing test cases may pass if they do not include scenarios with more than 10 points. Therefore, scenarios with fewer points might not reveal the fault.

Affected Test Cases:

- None of the provided test cases are affected since all test cases contain fewer than 10 points.

3. Mutation by Modification

Original Code Snippet:

```
if (((Point) p.get(i)).y == ((Point) p.get(min)).y && (((Point) p.get(i)).x >((Point) p.get(min)).x)) {
    min = i;
}
```

Mutation:

```
// Modifying the comparison to incorrectly select the minimum
if (((Point) p.get(i)).y == ((Point) p.get(min)).y && (((Point) p.get(i)).x <((Point) p.get(min)).x)) { // Changed '>' to '<'
    min = i;
}
```

Impact of Modification:

- Changing the comparison from > to < means that in the event of a tie for the y values, the code will now select the point with the smallest x instead of the largest. This could lead to incorrect behaviour in scenarios where points with the same y value exist but differ in x.

Test Cases Affected:

- Test cases specifically designed to test ties in y values will be affected. For instance, cases like [(1, 1), (2, 1), (3, 1)] where x values differ but y values are the same will pass even though the logic is flawed.

Affected Test Cases:

- p = [(0, 0), (1, 0), (2, 0)]
    - Failure: Instead of selecting (2, 0), the code will select (0, 0) due to the mutation that prioritizes the smallest xxx value.

➢ Path Coverage Derivation

Test Case 1: Empty List (Zero Iterations for Both Loops)

- **Description**: An empty list should result in no iteration in either loop.
- **Input**: points = []
- **Expected Output**: Return value (or behaviour) may vary depending on error handling (e.g., return None or raise an error).

Test Case 2: Single Point (Zero Iterations for the First Loop, One Iteration for the Second Loop)

- **Description**: With only one point, the first loop won't execute because i starts from 1, which is equal to the list length. The second loop will execute once.
- **Input**: points = [Point(1, 1)]
- **Expected Output**: 0 (index of the only point)

Test Case 3: Two Points with Different y Values (One Iteration for the First Loop, One Iteration for the Second Loop)

- **Description**: With two points, the first loop will execute once (checking the second point). The second loop will also execute once since the second point has a different y value.
- **Input**: points = [Point(1, 2), Point(3, 1)]
- **Expected Output**: 1 (index of the point with the smallest y coordinate)

Test Case 4: Two Points with the Same y Value (One Iteration for the First Loop, Two Iterations for the Second Loop)

- **Description**: With two points that have the same y value, the first loop will execute once to find the minimum y. The second loop will execute twice, as both points have the same y value, and it will select the one with the higher x.
- **Input**: points = [Point(1, 1), Point(2, 1)]
- **Expected Output**: 1 (index of the point with the highest x for the same y)

Test Case 5: Three Points with Different y Values (Two Iterations for the First Loop, One Iteration for the Second Loop)

- **Description**: With three points, each with a unique y value, the first loop will iterate twice to identify the minimum y. The second loop will execute once since no other point has the same y as the minimum.
- **Input**: points = [Point(1, 2), Point(3, 1), Point(4, 3)]
- **Expected Output**: 1 (index of the point with the smallest y coordinate)

Test Case 6: Three Points with Same y Value (One Iteration for the First Loop, Three Iterations for the Second Loop)

- **Description**: With three points that all have the same y value, the first loop will execute once to find the minimum y. The second loop will execute three times, as each point has the same y, and it will select the one with the highest x.
- **Input**: points = [Point(1, 1), Point(3, 1), Point(2, 1)]
- **Expected Output**: 1 (index of the point with the highest x value)