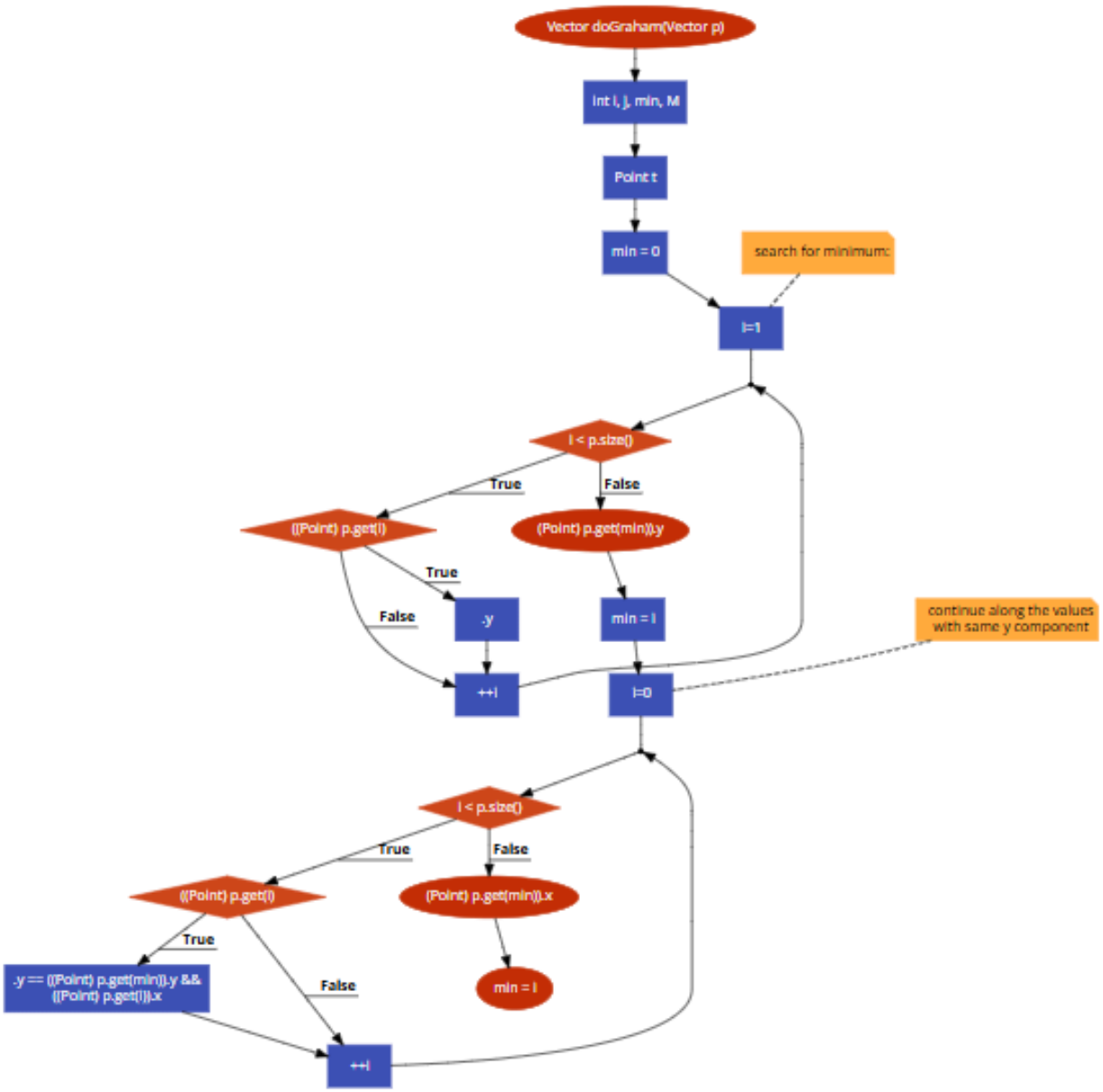
IT313: Software Engineering
Lab Session – Mutation Testing

ROHIN SOLANKI - 202201039

1. Control Flow Diagram



2. a. Statement Coverage

Objective: Ensure that each line of code is executed at least once.

To achieve statement coverage, we need a test case where:

- `p.size()` is greater than 1 so that both loops are entered.

Test Case:

1. **Test Case 1**:
    - **Input:** `p = [(0, 1), (1, 0)]`
    - **Explanation:** This will enter both loops, ensuring that all statements in the code are executed at least once. The first loop will compare the y values of the two points, and the second loop will check if the x values are equal, as their y values will not match.

b. Branch Coverage

Objective: Ensure that each branch in the code is taken at least once, both `True` and `False` outcomes.

To achieve branch coverage, we need test cases that cover the following conditions:

1. The condition `p[i].y < p[min].y` in the first loop should be both `True` and `False`.
2. The condition `p[i].y == p[min].y && p[i].x < p[min].x` in the second loop should be both `True` and `False`.

Test Cases:

1. **Test Case 1**:
    - **Input:** `p = [(0, 1), (1, 0)]`
    - **Explanation:**
        - In the first loop, `p[1].y < p[0].y` is `True`, so `min` will be updated to 1.
        - In the second loop, since `p[0].y != p[1].y`, the condition `p[i].y == p[min].y` will be `False`, covering the `False` branch.
2. **Test Case 2**:
    - **Input:** `p = [(1, 1), (0, 1)]`
    - **Explanation:**

    - In the first loop, the condition `p[1].y < p[0].y` is `False`, so `min` remains 0.
    - In the second loop, `p[1].y == p[0].y` and `p[1].x < p[0].x` are both `True`, covering the `True` branch for this condition.

With these two cases, all branches in the code are exercised, both `True` and `False`.

c. Basic Condition Coverage

Objective: Test each individual condition within the `if` statements independently, ensuring that each basic condition is both `True` and `False`.

Conditions to Test:

1. **Condition 1:** `p[i].y < p[min].y` (inside the first `if` condition).
2. **Condition 2:** `p[i].y == p[min].y` and `p[i].x < p[min].x` (both parts of the second `if` condition).

Test Cases:

1. **Condition 1: `p[i].y < p[min].y`**
    - **True:** `p = [(0, 1), (1, 0)]`
        - Explanation: In the first loop, `p[1].y < p[0].y` is `True`.
    - **False:** `p = [(1, 0), (0, 1)]`
        - Explanation: In the first loop, `p[1].y < p[0].y` is `False`.
2. **Condition 2: `p[i].y == p[min].y` and `p[i].x < p[min].x`**
    - **True:** `p = [(1, 1), (0, 1)]`
        - Explanation: In the second loop, `p[1].y == p[0].y` and `p[1].x < p[0].x` are both `True`.
    - **False:** `p = [(0, 1), (1, 1)]`
        - Explanation: In the second loop, `p[1].y == p[0].y` is `True`, but `p[1].x < p[0].x` is `False`.

Test Case	Input	Explanation
1	p = [(0, 1), (1, 0)]	Covers statement coverage and sets p[1].y < p[0].y to True in the first loop
2	p = [(1, 1), (0, 1)]	Covers the True branch of the second loop's condition p[i].y == p[min].y && p[i].x < p[min].x
3	p = [(1, 0), (0, 1)]	Covers the False case for p[i].y < p[min].y
4	p = [(0, 1), (1, 1)]	Covers the False case for p[i].x < p[min].x when p[i].y == p[min].y

1. Deletion Mutation: Remove `min = i` in the first `if` condition.

Mutation Code:

```
// Remove this line: min = i;

if (((Point) p.get(i)).y < ((Point) p.get(min)).y) {

    // min = i; <- this line is removed

}
```

- **Explanation**: By removing the `min = i` assignment in the first loop, we stop updating `min` to the index of the point with the smallest y value. This would likely result in an incorrect `min` value after the first loop.
- **Detection**: This mutation might go undetected if all y values in the test cases are equal or if the test set lacks cases where multiple points have different y values. This could result in an incorrect outcome without triggering any existing assertions in the test cases.
- **Undetected Failure**: If all points have the same y value in the test set, the second loop (which checks x values) would take over and potentially produce the correct `min`, masking the failure. However, with more diverse y values, this mutation would cause incorrect results.

2. Insertion Mutation: Add `min = 0` at the beginning of the second loop.

Mutation Code:

```
for(i = 0; i < p.size(); ++i) {

    min = 0; // Inserted line

    if (((Point) p.get(i)).y == ((Point) p.get(min)).y &&

        ((Point) p.get(i)).x < ((Point) p.get(min)).x) {
```

```
        min = i;
    }
}
```

- **Explanation**: By adding `min = 0` at the start of each iteration in the second loop, we reset `min` each time. This mutation would override any updates made to `min` within the loop, likely leading to the selection of the wrong point.
- **Detection**: This mutation might not be detected if all points in the test set have unique `x` and `y` values, as it would never reach the point of needing the `min` update.
- **Undetected Failure**: If the test set lacks cases where multiple points have the same `y` value but different `x` values, this mutation might go undetected. The loop would incorrectly select the first point as the minimum each time due to the reset, affecting the outcome without detection.

3. Modification Mutation: Change `p[i].y < p[min].y` to `p[i].y > p[min].y` in the first `if` condition.

Mutation Code:

```
if (((Point) p.get(i)).y > ((Point) p.get(min)).y) { // Modified from < to >
    min = i;
}
```

- **Explanation**: Changing `<` to `>` in the first `if` condition reverses the comparison, causing the code to select the highest `y` value instead of the lowest. This mutation would result in an incorrect `min` being chosen if `y` values vary.
- **Detection**: If the test set does not include cases with varying `y` values, this mutation may go undetected. For example, if all `y` values in the test set are the same, this comparison change would not alter the outcome.
- **Undetected Failure**: In a test set where all points have the same `y` value, the comparison modification would have no impact, and the failure would not be detected.

5. Path Coverage

Test Cases for Path Coverage

Test Case 1: Zero Iterations (No Points)

- **Input: `p = []` (empty vector)**
- **Explanation:**
  - **Both loops will be skipped because `p.size()` is 0.**
  - **This case covers the "zero iterations" path for both loops.**

Test Case 2: Zero Iterations (Single Point)

- **Input: `p = [(0, 0)]`**
- **Explanation:**
  - **The first loop will be skipped because `i` starts at 1 and `p.size()` is 1.**
  - **The second loop will execute only once (for the single point).**
  - **This case covers the "zero iterations" path for the first loop and a single iteration for the second loop.**

Test Case 3: One Iteration

- **Input: `p = [(0, 0), (1, 1)]`**
- **Explanation:**
  - **The first loop will execute exactly once, comparing the y values of the two points.**
  - **The second loop will execute twice, once for each point.**
  - **This case covers the "one iteration" path for the first loop and "two iterations" path for the second loop.**

Test Case 4: Two Iterations

- **Input: `p = [(0, 1), (1, 0), (2, 2)]`**
- **Explanation:**
  - **The first loop will execute twice, comparing each subsequent point's y value to find the minimum.**
  - **The second loop will also execute three times (one for each point) but will cover two distinct comparisons for the minimum point.**
  - **This case covers the "two iterations" path for the first loop and multiple iterations for the second loop.**

Test Case	Input	First Loop (Iterations)	Second Loop (Iterations)	Path Coverage Achieved
1	p = []	0	0	Zero iterations for both loops
2	p = [(0, 0)]	0	1	Zero for first loop, one for second
3	p = [(0, 0), (1, 1)]	1	2	One for first loop, two for second
4	p = [(0, 1), (1, 0), (2, 2)]	2	3	Two for first loop, three for second

1. CFG Validation Using Tools

- **Control Flow Graph Factory Tool: Yes (the generated CFG matches)**
- **Eclipse Flow Graph Generator: Yes (the generated CFG matches)**

2. Minimum Test Cases for Coverage Criteria

Here, we need to devise the minimum test cases to cover the following criteria:

- **Statement Coverage: Every line of code is executed at least once.**
- **Branch Coverage: Each possible outcome of every branch (true/false) is tested at least once.**
- **Basic Condition Coverage: Each condition in a decision statement is tested for both true and false outcomes.**

Based on the CFG, the following test cases cover the specified criteria:

Test Case	Input	Statement Coverage	Branch Coverage	Condition Coverage
1	p = []	Yes	Partial	No
2	p = [(0, 0)]	Yes	Partial	Partial
3	p = [(0, 0), (1, 1)]	Yes	Yes	Yes
4	p = [(0, 1), (1, 0), (2, 2)]	Yes	Yes	Yes

3. Mutation Testing

Using the test cases derived in Step 2, apply mutation testing by introducing various code changes (mutations) and verifying if the tests can detect them.

- **Deletion Mutation: Remove `min = i;` in the second loop.**
  - **Effect: This mutation would lead to incorrect results if `min` is not updated correctly, which should ideally be detected by Test Case 4.**
- **Insertion Mutation: Add `min = 0;` at the beginning of the second loop.**
  - **Effect: This change resets `min` unnecessarily and could result in incorrect results. Test Case 3 should detect this mutation.**
- **Modification Mutation: Change the condition `p.get(i).y < p.get(min).y` to `p.get(i).y <= p.get(min).y`.**
  - **Effect: This mutation may alter the point chosen as minimum, but may not be detected by all test cases. This might reveal a limitation in the test set if undetected.**

Answer:

- **Deletion Mutation: Detected by Test Case 4.**
- **Insertion Mutation: Detected by Test Case 3.**
- **Modification Mutation: Potentially undetected, indicating a limitation in the test set.**

4. Path Coverage Test Set

Test Case	Input	First Loop (Iterations)	Second Loop (Iterations)
1	p = []	0	0
2	p = [(0, 0)]	0	1
3	p = [(0, 0), (1, 1)]	1	2
4	p = [(0, 1), (1, 0), (2, 2)]	2	3

Answer: These test cases satisfy the path coverage criterion, covering each loop with zero, one, and two iterations.