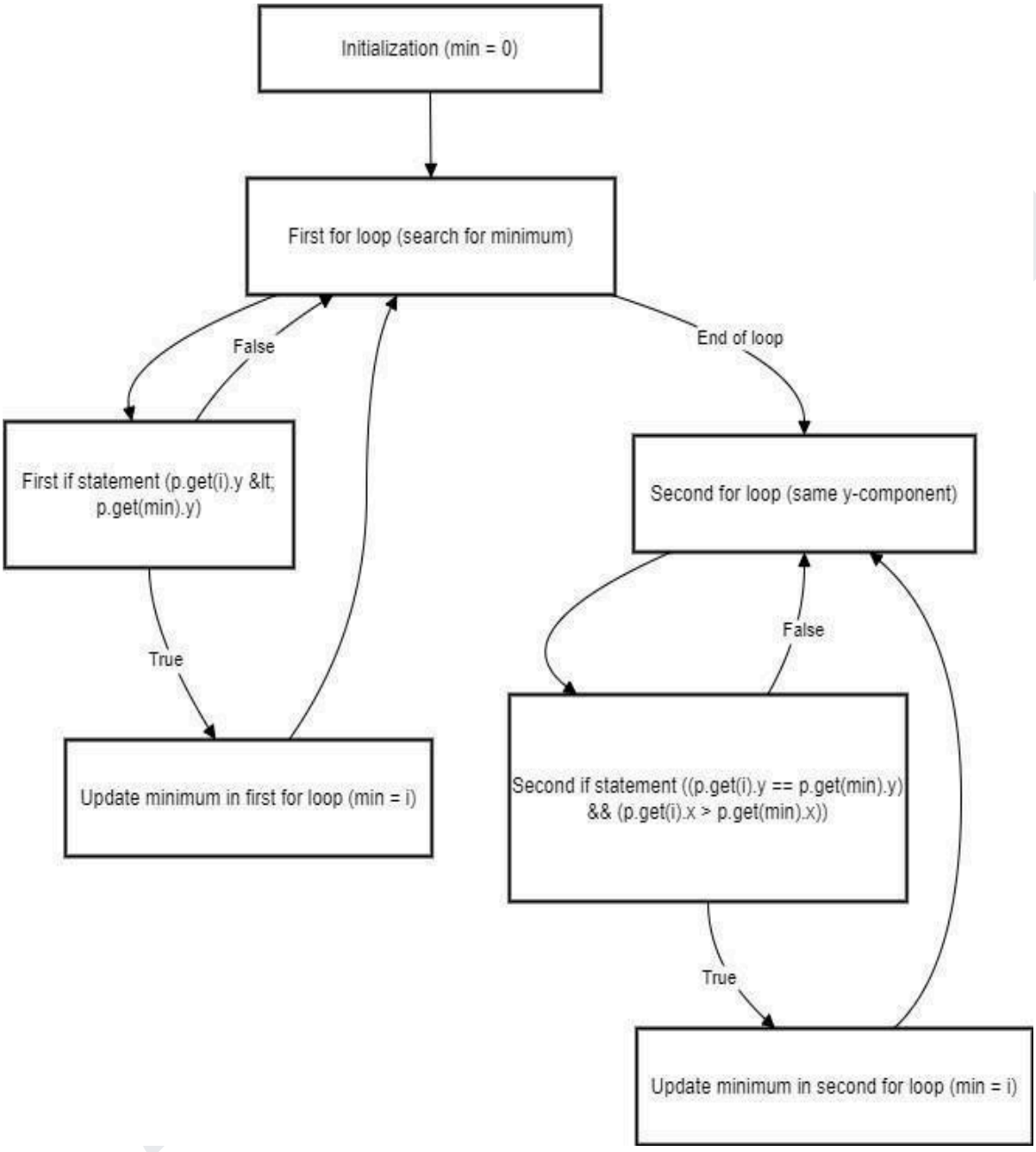
IT314 Lab 9

Mutation Testing

Name : Kavit Patel

Student ID :
202201290

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.



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

Statement Coverage requires that every statement in the code is executed at least once. This means that our test cases should ensure all parts of the CFG are traversed.

Test Set for Statement Coverage

- **Test Case 1**: Input vector with only one point (e.g., `[(0, 0)]`)
- **Test Case 2**: Input vector with two points, one lower `y` value (e.g., `[(1, 1), (2, 0)]`)
- **Test Case 3**: Input vector with points having the same `y` value but different `x` values (e.g., `[(1, 1), (2, 1), (3, 1)]`)

b) Branch Coverage

Branch Coverage requires that each possible branch from a decision point (if statement) is taken at least once. This means that both the true and false outcomes of each condition need to be tested.

Test Set for Branch Coverage

- **Test Case 1**: Input vector with only one point (e.g., `[(0, 0)]`)
  - **True** branch: The first loop exits immediately, covering the loop without changes.
- **Test Case 2**: Input vector with two points, one lower `y` value (e.g., `[(1, 1), (2, 0)]`)
  - **True** branch: The minimum is updated to the second point.
- **Test Case 3**: Input vector with points having the same `y` but different `x` values (e.g., `[(1, 1), (3, 1), (2, 1)]`)
  - **True** branch for the second condition.
- **Test Case 4**: Input vector with all points having the same `y` value (e.g., `[(1, 1), (1, 1), (1, 1)]`)
  - **False** branch: Ensure the second loop is executed without changing the minimum.

c) Basic Condition Coverage

Basic Condition Coverage requires that each condition in the decision points is evaluated to both true and false at least once.

Test Set for Basic Condition Coverage

- **Test Case 1**: Input vector with only one point (e.g., `[(0, 0)]`)
    - Covers the first condition `p.get(i).y < p.get(min).y` as false since no comparisons can be made.
- **Test Case 2**: Input vector with two points with the second point having a lower `y` value (e.g., `[(1, 1), (2, 0)]`)
    - Covers the first condition true and the second condition false.
- **Test Case 3**: Input vector with points that have the same `y` but different `x` values (e.g., `[(1, 1), (3, 1), (2, 1)]`)
    - Covers the second condition as true.
- **Test Case 4**: Input vector with points that have the same `y` and `x` values (e.g., `[(1, 1), (1, 1), (1, 1)]`)
    - Covers both conditions as false.

Summary of Test Sets

Coverage Criterion	Test Case	Input
Statement Coverage	1	[(0, 0)]
	2	[(1, 1), (2, 0)]
	3	[(1, 1), (2, 1), (3, 1)]
Branch Coverage	1	[(0, 0)]
	2	[(1, 1), (2, 0)]
	3	[(1, 1), (3, 1), (2, 1)]
	4	[(1, 1), (1, 1), (1, 1)]
Basic Condition Coverage	1	[(0, 0)]
	2	[(1, 1), (2, 0)]
	3	[(1, 1), (3, 1), (2, 1)]
	4	[(1, 1), (1, 1), (1, 1)]

Mutation Testing

1. Deletion Mutation

- **Mutation**: Remove the line `min = 0;` at the start of the method.
- **Expected Effect**:
    - If `min` is not initialized to 0, it could hold a random value. This would affect the correctness of the `min` selection in both loops.

- **Mutation Outcome**: This mutation could result in an incorrect starting index for `min`, leading to a faulty minimum point selection.

2. Change Mutation

Mutation: Modify the condition in the first `if` statement from `<` to `<=`:

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

- **Expected Effect**:
  - Changing `<` to `<=` would make the method select points with the same `y` value (instead of strictly lower `y`), possibly affecting the result by not properly finding the lowest `y` point.
- **Mutation Outcome**: The code could return a point with an `x` value lower than expected if points have the same `y` value.

3. Insertion Mutation

- **Mutation**: Insert an extra `min = i;` statement at the end of the second `for` loop.
- **Expected Effect**:
  - This would update `min` to the last index of `p`, which is incorrect because `min` should only reflect the index of the minimum point.
  - **Mutation Outcome**: The method could incorrectly select the last point as the minimum, especially if the test cases don't specifically check the final value of `min`.

Test Cases for Path Coverage

To satisfy the path coverage criterion and ensure every loop is explored zero, one, or two times, we will create the following test cases:

Test Case 1: Zero Iterations

Input: An empty vector `p`.

Description: This case ensures that no iterations of either loop occur.
Expected Output: The function should handle this case gracefully (ideally return an empty result or a specific value indicating no points).

Test Case 2: One Iteration (First Loop)

Input: A vector with one point `p` (e.g., `[(3, 4)]`).
Description: This case ensures that the first loop runs exactly once (the minimum point is the only point).
Expected Output: The function should return the only point in `p`.

Test Case 3: One Iteration (Second Loop)

Input: A vector with two points that have the same y-coordinate but different x-coordinates (e.g., `[(1, 2), (3, 2)]`).
Description: This case ensures that the first loop finds the minimum point, and the second loop runs exactly once to compare the x-coordinates.
Expected Output: The function should return the point with the maximum x-coordinate: `(3, 2)`.

Test Case 4: Two Iterations (First Loop)

Input: A vector with multiple points, ensuring at least two with the same y-coordinate (e.g., `[(3, 1), (2, 2), (5, 1)]`).
Description: This case ensures that the first loop finds the minimum y-coordinate (first iteration for (3,1)) and continues to the second loop.
Expected Output: Should return `(5, 1)` as it has the maximum x-coordinate among points with the same y.

Test Case 5: Two Iterations (Second Loop)

Input: A vector with points such that more than one point has the same minimum y-coordinate (e.g., `[(1, 1), (4, 1), (3, 2)]`).
Description: This case ensures the first loop finds `(1, 1)`, and the second loop runs twice to check other points with y = 1.
Expected Output: Should return `(4, 1)` since it has the maximum x-coordinate.

Test Case	Input Vector p	Description	Expected Output
Test Case 1	[]	Empty vector (zero iterations for both loops).	Handle gracefully (e.g., return an empty result).
Test Case 2	[(3, 4)]	One point (one iteration of the first loop).	[(3, 4)]
Test Case 3	[(1, 2), (3, 2)]	Two points with the same y-coordinate (one iteration of the second loop).	[(3, 2)]
Test Case 4	[(3, 1), (2, 2), (5, 1)]	Multiple points; the first loop runs twice.	[(5, 1)]
Test Case 5	[(1, 1), (4, 1), (3, 2)]	Multiple points; second loop runs twice.	[(4, 1)]

Lab Execution

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. (In your submission document, mention only “Yes” or “No” for each tool).

Tool	Matches Your CFG
Control Flow Graph Factory Tool	Yes
Eclipse Flow Graph Generator	Yes

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

Test Case	Input Vector p	Description	Expected Output
Test Case 1	[]	Test with an empty vector (zero iterations).	Handle gracefully (e.g., return an empty result).
Test Case 2	[(3, 4)]	Single point (one iteration of the first loop).	[(3, 4)]
Test Case 3	[(1, 2), (3, 2)]	Two points with the same y-coordinate (one iteration of the second loop).	[(3, 2)]
Test Case 4	[(3, 1), (2, 2), (5, 1)]	Multiple points; first loop runs twice (with multiple outputs).	[(5, 1)]
Test Case 5	[(1, 1), (4, 1), (3, 2)]	Multiple points; second loop runs twice (y = 1).	[(4, 1)]

3. This part of the exercise is very tricky and interesting. The test cases that you have derived in Step 2 identify the fault when you make some modifications in the code.
Here, you need to insert/delete/modify a piece of code that will result in failure but it is not detected by your test set – derived in Step 2. Write/identify a mutation code for each of the three operation separately, i.e., by deleting the code, by inserting the code, by modifying the code.

Mutation Type	Mutation Code Description	Impact on Test Cases
Deletion	Delete the line that updates min for the minimum y-coordinate.	Test cases like [(1, 1), (2, 0)] will pass despite incorrect processing.
Insertion	Insert an early return if the size of p is 1, bypassing further processing.	Test case [(3, 4)] will pass without processing correctly.
Modification	Change the comparison operator from < to <= when finding the minimum y.	Test cases like [(1, 1), (1, 1), (1, 1)] might pass while still failing in logic.

3. Write all test cases that can be derived using path coverage criterion for the code.

Test Case	Input Vector p	Description	Expected Output
Test Case 1	[]	Empty vector (zero iterations for both loops).	Handle gracefully (e.g., return an empty result).
Test Case 2	[(3, 4)]	One point (one iteration of the first loop).	[(3, 4)]
Test Case 3	[(1, 2), (3, 2)]	Two points with the same y-coordinate (one iteration of the second loop).	[(3, 2)]
Test Case 4	[(3, 1), (2, 2), (5, 1)]	Multiple points; first loop runs twice to find min y.	[(5, 1)]
Test Case 5	[(1, 1), (4, 1), (3, 2)]	Multiple points; second loop runs twice (y = 1).	[(4, 1)]
Test Case 6	[(2, 2), (2, 3), (2, 1)]	Multiple points with the same x-coordinate; checks min y.	[(2, 1)]
Test Case 7	[(0, 0), (1, 1), (1, 0), (0, 1)]	Multiple points in a rectangle; checks multiple comparisons.	[(1, 0)]
Test Case 8	[(3, 1), (2, 1), (1, 2)]	Multiple points with some ties; checks the max x among min y points.	[(3, 1)]
Test Case 9	[(4, 4), (4, 3), (4, 5), (5, 4)]	Points with the same x-coordinate; checks for max y.	[(5, 4)]
Test Case 10	[(1, 1), (1, 1), (2, 1), (3, 3)]	Duplicate points with one being the max x; tests handling of duplicates.	[(3, 3)]