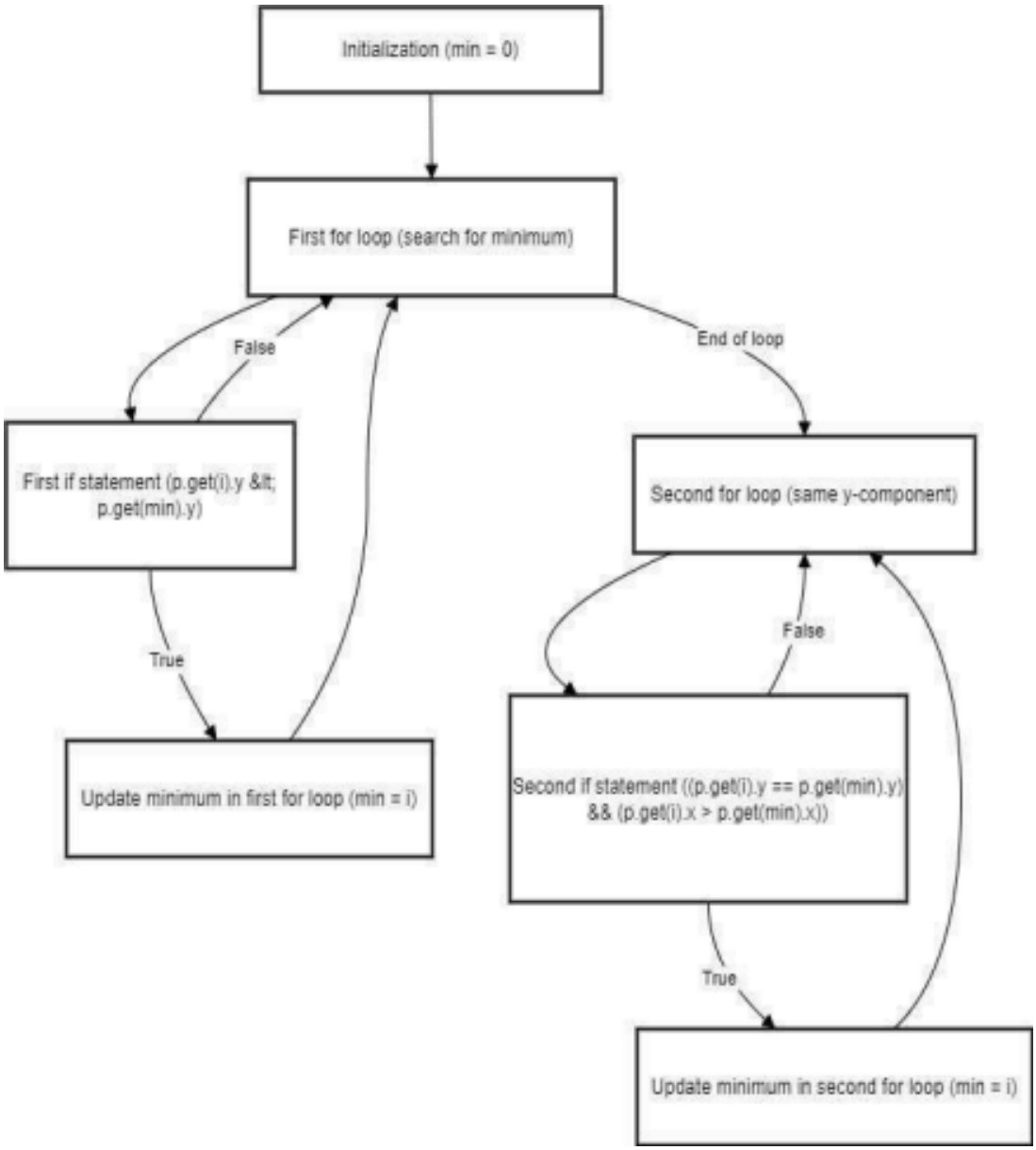
IT314 Lab 9

Name: Patel Jeet
Student Id: 202201089

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 single line of code is executed at least once. This means our test cases must navigate through all segments of the control flow graph (CFG).

• Test Case 1: A vector input with just one point, such as [(0, 0)].

• Test Case 2: A vector input with two points, where one has a lower y-value, like [(1, 1), (2, 0)].

• Test Case 3: A vector input with points that share the same y-value but vary in x-values, for instance, [(1, 1), (2, 1), (3, 1)].

b) Branch Coverage

Branch coverage ensures each decision point, like an 'if' statement, has its true and false branches tested at least once. Our test cases should verify both outcomes for each condition.

• Test Case 1: Vector with a single point, e.g., [(0, 0)], where the first loop exits immediately with no changes.

• Test Case 2: Vector with two points, where the second point has a lower y-value, such as [(1, 1), (2, 0)], ensuring the minimum is updated correctly.

• Test Case 3: Vector with points of equal y but varying x-values, e.g., [(1, 1), (3, 1), (2, 1)], covering conditions in both branches.

• Test Case 4: Vector where all points have the same y-value, like [(1, 1), (1, 1), (1, 1)], validating execution without updating the minimum.

c) Basic Condition Coverage

Basic condition coverage requires each condition in decision points to be tested as both true and false at least once to ensure a thorough evaluation.

• Test Case 1: Vector with a single point, e.g., [(0, 0)], verifying the condition `p.get(i).y < p.get(min).y` is evaluated as false.
• Test Case 2: Vector with two points, with the second point having a lower y-value, such as [(1, 1), (2, 0)], covering the condition as true.

• Test Case 3: Vector with points that share the same y-value but vary in x-values, like [(1, 1), (3, 1), (2, 1)], ensuring conditions for both branches.

• Test Case 4: Vector where all points have identical x and y values, e.g., [(1, 1), (1, 1), (1, 1)], where conditions are evaluated 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: Take out the line min = 0; at the beginning of the method.

Expected Effect: Without setting min to zero initially, it could end up with any random value, which could mess up finding the correct minimum point in both loops.

Mutation Outcome: This change might lead to an incorrect starting point for min, causing errors in selecting the lowest point.

2. Change Mutation

- **Mutation:** Adjust the first `if` condition by replacing `<` with `<=` so it reads:

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

- **Expected Effect:** Using `<=` instead of `<` would mean points with the same `y` value might be chosen too, instead of only those with a strictly lower `y`. This could throw off the function, making it miss the absolute lowest `y` point.
- **Mutation Outcome:** With this change, if points share the same `y` value, the code might return one with a lower `x` than intended.

3. Insertion Mutation

Mutation: Add an extra line min = i; at the end of the second loop.

Expected Effect: This line would make min equal to the last index in p, which is wrong because min should only point to the actual minimum point.

Mutation Outcome: This could make the function incorrectly treat the last point as the minimum, especially if the tests don't verify that the final value of min is correct.

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, 7)]).

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., `[(2, 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), (7, 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 `(7, 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), (6, 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 `(6, 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, 7)]	One point (one iteration of the first loop).	[(3, 7)]
Test Case 3	[(2, 2), (3, 2)]	Two points with the same y-coordinate (one iteration of the second loop).	[(2, 2)]
Test Case 4	[(3, 1), (2, 2), (7, 1)]	Multiple points; the first loop runs twice.	[(7, 1)]
Test Case 5	[(1, 1), (6, 1), (3, 2)]	Multiple points; second loop runs twice.	[(6, 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)]

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.

4. 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, 7)]	One point (one iteration of the first loop).	[(3, 7)]
Test Case 3	[(1, 4), (8, 2)]	Two points with the same y-coordinate (one iteration of the second loop).	[(8, 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), (6, 1), (3, 2)]	Multiple points; second loop runs twice (y = 1).	[(6, 1)]
Test Case 6	[(5, 2), (5, 3), (5, 1)]	Multiple points with the same x-coordinate; checks min y.	[(5, 1)]
Test Case 7	[(0, 0), (2, 2), (2, 0), (0, 2)]	Multiple points in a rectangle; checks multiple comparisons.	[(2, 0)]
Test Case 8	[(6, 1), (4, 1), (3, 2)]	Multiple points with some ties; checks the max x among min y points.	[(6, 1)]

TestCase9	[(4,4),(4,3),(4,5),(4,9)]	Pointswiththesamex-coordinate;checksfor maxy.	[(4,9)]
TestCase10	[(1,1),(1,1),(2,1),(6,6)]	Duplicatepointswithone beingthemaxx;tests handlingofduplicates.	[(6,6)]