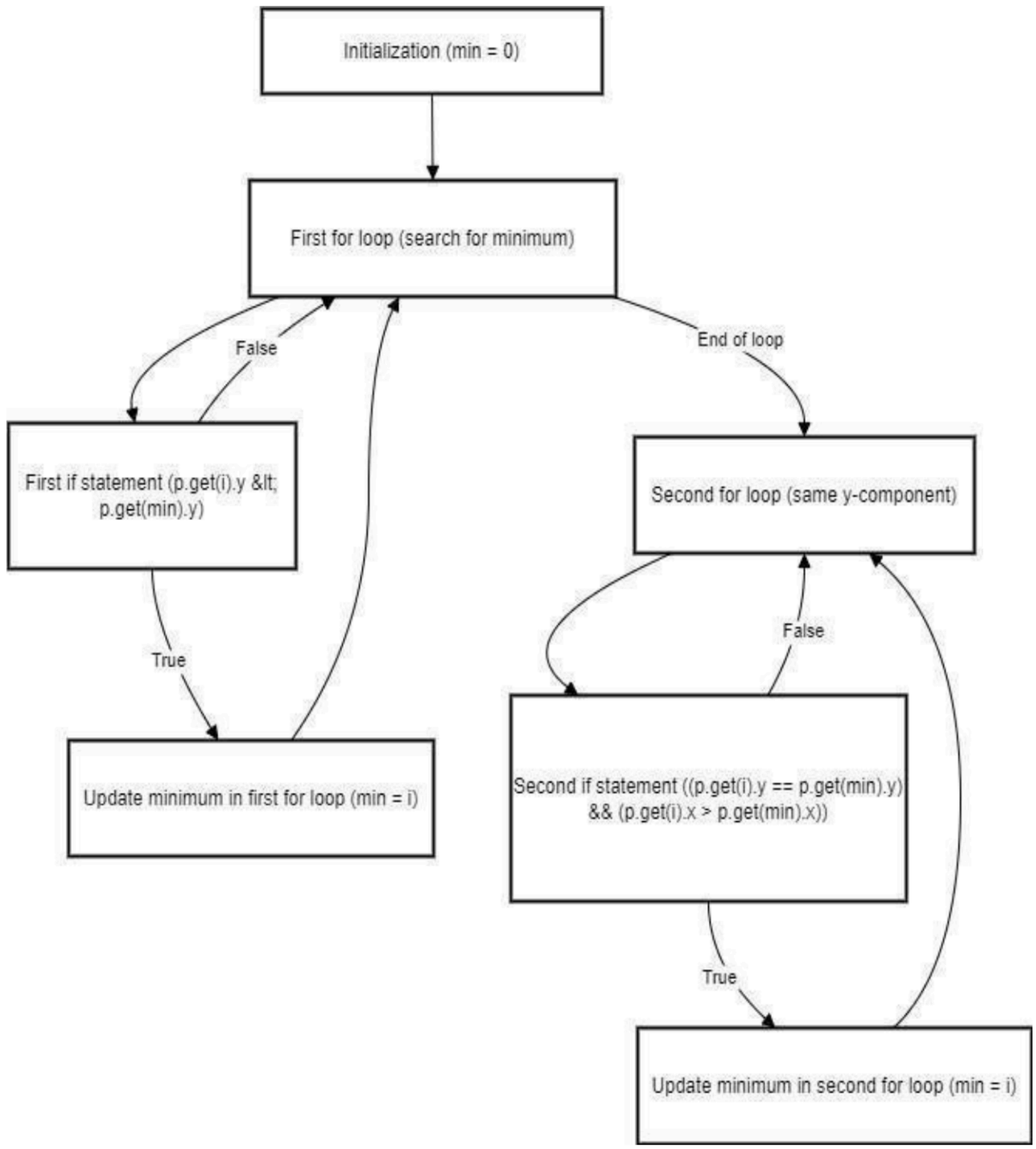
IT314 - Lab 9
Vraj Patel - 202201401

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.

To construct adequate test cases for a flow graph, we will apply three code coverage criteria: Statement Coverage, Branch Coverage, and Basic Condition Coverage.

a) Statement Coverage

Statement Coverage ensures that every line of code is executed at least once. Our test cases must traverse all paths in the control flow graph (CFG).

- **Test Case 1:** A vector with a single point, e.g., `[(0, 0)]`.
- **Test Case 2:** A vector with two points, where one has a lower y-value, such as `[(1, 1), (2, 0)]`.
- **Test Case 3:** A vector with points sharing the same y-value but different x-values, for instance, `[(1, 1), (2, 1), (3, 1)]`.

b) Branch Coverage

Branch Coverage requires testing each decision point so that both true and false outcomes are evaluated. This ensures that all branches of the code are executed.

- **Test Case 1:** A vector with a single point, e.g., `[(0, 0)]`, where the loop exits immediately without changes.
- **Test Case 2:** A vector with two points, with the second point having a lower y-value, such as `[(1, 1), (2, 0)]`, ensuring the minimum is updated correctly.
- **Test Case 3:** A vector with points having the same y-value but varying x-values, e.g., `[(1, 1), (3, 1), (2, 1)]`, covering different outcomes for each branch.
- **Test Case 4:** A vector with all points having the same y-value, like `[(1, 1), (1, 1), (1, 1)]`, validating that the code runs without changing the minimum.

c) Basic Condition Coverage

Basic Condition Coverage requires each condition in a decision point to be evaluated as both true and false at least once. This covers all possible outcomes of each condition.

- **Test Case 1:** A vector with a single point, e.g., `[(0, 0)]`, ensuring the condition `p.get(i).y < p.get(min).y` evaluates as false.

- **Test Case 2:** A vector with two points, with the second point having a lower y-value, like `[(1, 1), (2, 0)]`, testing the condition as true.
- **Test Case 3:** A vector with points sharing the same y-value but differing x-values, e.g., `[(1, 1), (3, 1), (2, 1)]`, covering conditions in both true and false branches.
- **Test Case 4:** A vector where all points have identical x and y values, such as `[(1, 1), (1, 1), (1, 1)]`, where the condition is consistently false.

Mutation Testing

1. Deletion Mutation
   - Mutation: Remove the line `min = 0;` at the start of the method.
   - Expected Effect: Without setting `min` to zero initially, it could hold any random value, potentially leading to errors in locating the correct minimum point in both loops.
   - Mutation Outcome: This omission may lead to an incorrect starting point for `min`, resulting in incorrect selection of the lowest point.

2. Change Mutation
   - Mutation: Modify the first `if` condition, changing `<` to `<=`, making it: `if (((Point) p.get(i)).y <= ((Point) p.get(min)).y)`
   - Expected Effect: By using `<=` instead of `<`, points with equal y-values could also be selected, rather than just those with strictly lower y-values. This could affect the function's accuracy in finding the absolute lowest y-value.
   - Mutation Outcome: With this change, if points have the same y-value, the function may incorrectly select a point with a lower x-coordinate instead of the intended one.

3. Insertion Mutation
   - Mutation: Add an extra line `min = i;` at the end of the second loop.
   - Expected Effect: This line would set `min` to the last index in `p`, which is incorrect since `min` should only indicate the actual minimum point.
   - Mutation Outcome: This mutation may lead the function to incorrectly treat the last point as the minimum, especially if tests do not confirm the final `min` value is accurate.

Test Cases for Path Coverage

To ensure that all paths through the loops are tested for zero, one, or two iterations, the following test cases are designed:

1. Zero Iterations
   - Input: An empty vector `p`.
   - Description: Ensures that both loops are not executed at all.
   - Expected Output: The function should handle this gracefully, ideally by returning an empty result or a specific value indicating no points.

2. One Iteration (First Loop)
   - Input: A vector with a single point, `[(3, 7)]`.
   - Description: Ensures the first loop runs exactly once, setting the minimum point as the only point in the vector.
   - Expected Output: The function should return the only point in `p`.

3. One Iteration (Second Loop)
   - Input: A vector with two points sharing the same y-coordinate but different x-coordinates, such as `[(2, 2), (3, 2)]`.
   - Description: Ensures that the first loop identifies the minimum point, and the second loop executes once to compare x-coordinates.
   - Expected Output: The function should return the point with the highest x-coordinate, `(3, 2)`.

4. Two Iterations (First Loop)
   - Input: A vector with multiple points, at least two of which have the same y-coordinate, such as `[(3, 1), (2, 2), (7, 1)]`.
   - Description: Ensures that the first loop identifies the minimum y-coordinate (first iteration for `(3,1)`) and moves on to the second loop.
   - Expected Output: Should return `(7, 1)`, as it has the maximum x-coordinate among points with the same y-value.

5. Two Iterations (Second Loop)
   - Input: A vector where more than one point has the same minimum y-coordinate, such as `[(1, 1), (6, 1), (3, 2)]`.
   - Description: Ensures that the first loop finds `(1, 1)`, and the second loop runs twice to compare points with `y = 1`.
   - Expected Output: Should return `(6, 1)` since it has the highest x-coordinate among points with the same minimum y.

Lab Execution:-

Q.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.

Control Flow Graph Factory :- YES

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

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

Q.3 and **Q.4** Same as Part I