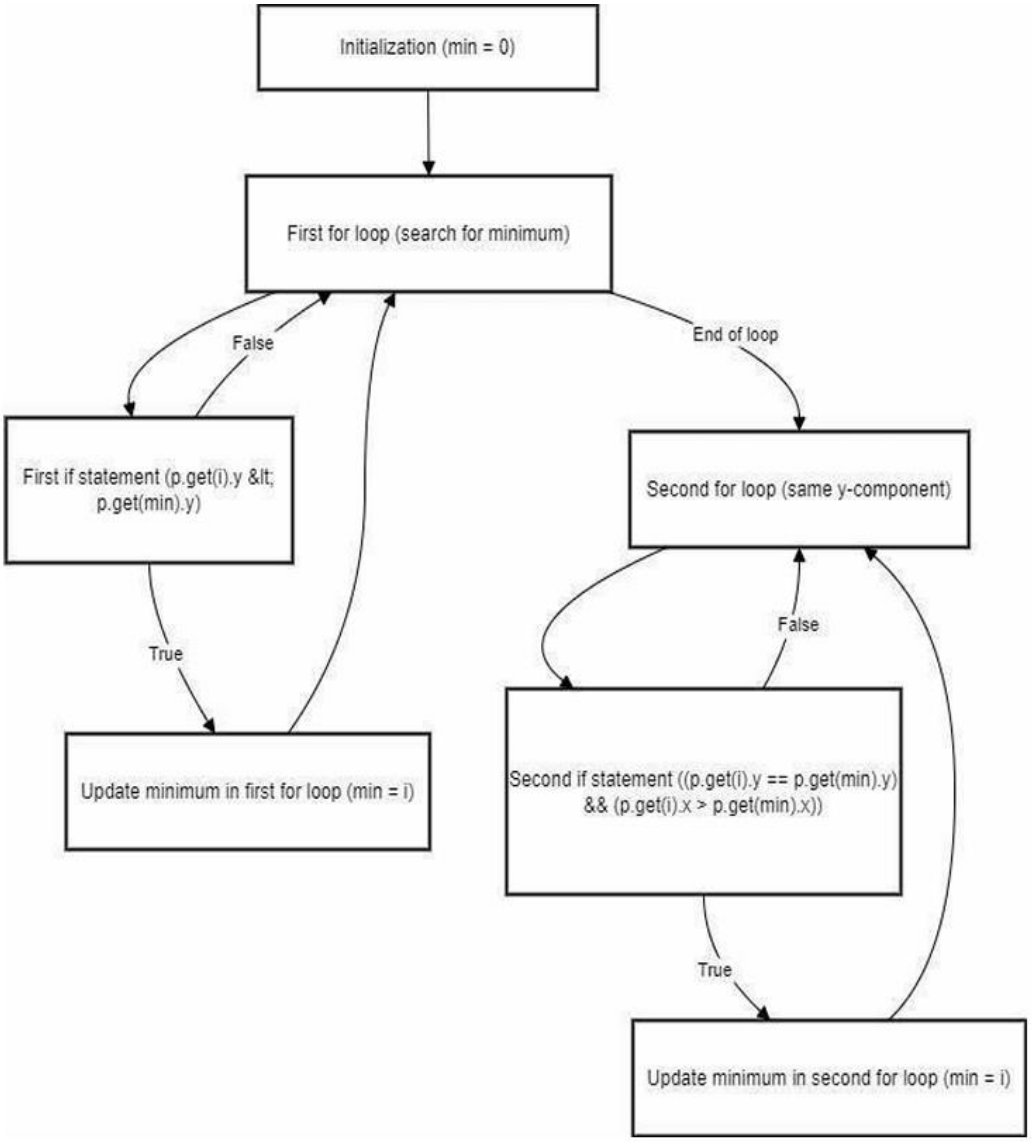
Software Engineering
Lab 9

Name: Ragan Patel
Student I'd: 202201152

1. Convert the code comprising the beginning of the do Graham 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 each line of code is executed at least once, meaning our test cases must cover all paths in the control flow graph (CFG).

- **Test Case 1:** A vector input with a single 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 have the same y-value but different x-values, such as [(1, 1), (2, 1), (3, 1)].

B. Branch Coverage:

Branch coverage ensures that each decision point, such as an "if" statement, has its true and false branches tested at least once. Our test cases should confirm both outcomes for each condition.

- **Test Case 1**: A vector with a single point, such as [(0, 0)], where the first loop exits immediately with no changes made.
- **Test Case 2:** A vector with two points, where the second point has a lower y-value, like [(1, 1), (2, 0)], ensuring that the minimum value is correctly updated.
- **Test Case 3:** A vector with points that have the same y-value but different x-values, such as [(1, 1), (3, 1), (2, 1)], covering conditions in both branches.
- **Test Case 4:** A vector where all points have the same y-value, like [(1, 1), (1, 1), (1, 1)], verifying that execution proceeds without updating the minimum value.

C. Basic Condition Coverage:

Basic condition coverage requires each condition in decision points to be tested as both true and false at least once, ensuring comprehensive evaluation.

- **Test Case 1:** A vector with a single point, such as **[(0, 0)]**, verifying that the condition p.get(i).y < p.get(min).y is evaluated as false.
- **Test Case 2:** A vector with two points, where the second point has a lower y-value, like **[(1, 1), (2, 0)]**, covering the condition as true.
- **Test Case 3:** A vector with points that have the same y-value but different x-values, such as **[(1, 1), (3, 1), (2, 1)]**, ensuring conditions in both branches are evaluated.
- **Test Case 4:** A vector where all points have identical x and y values, like **[(1, 1), (1, 1), (1, 1)]**, where conditions are evaluated as false.

Coverage Criteria	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	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: Without initializing min to zero, it could hold an arbitrary value, which may disrupt the search for the correct minimum point during the loops.
- Mutation Outcome: This could result in an incorrect initial value for min, leading to errors in selecting the lowest point.

2. Change Mutation

- Mutation: Modify the initial if condition by replacing < with <=, so it reads: if (((Point) p.get(i)).y <= ((Point) p.get(min)).y)
- Expected Effect: Changing < to <= would allow points with the same y-value to be selected, rather than only those with a strictly lower y-value. This could disrupt the function, causing it to overlook the absolute lowest y-point.
- Mutation Outcome: With this change, if multiple points have the same y-value, the code might incorrectly return a point with a lower x-value than intended.

3. Insertion Mutation

- **Mutation:** Insert an additional line, min = i;, at the end of the second loop.
- **Expected Effect:** This would set min to the last index in p, which is incorrect because min should only reference the actual minimum point.
- **Mutation Outcome:** This could cause the function to incorrectly designate the last point as the minimum, especially if the tests do not verify that min reflects the true minimum value at the end.

Test Cases for Path Coverage

To meet the path coverage criterion and ensure each loop is explored zero, one, or two times, we propose the following test cases:

- **Test Case 1: Zero Iterations**
    - **Input:** An empty vector p.
    - **Description:** Verifies that no iterations occur in either loop.
    - **Expected Output:** The function should handle this gracefully, either returning an empty result or a specific value indicating no points.
- **Test Case 2: One Iteration (First Loop)**
    - **Input:** A vector with a single point p = [(3, 7)].
    - **Description:** Tests that the first loop runs exactly once, identifying the only point as the minimum.
    - **Expected Output:** The function should return the only point in p.
- **Test Case 3: One Iteration (Second Loop)**
    - **Input:** A vector with two points having the same y-coordinate but different x-coordinates, such as [(2, 2), (3, 2)].
    - **Description:** Ensures that the first loop finds the minimum point, while the second loop runs once to compare 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, including at least two with the same y-coordinate, e.g., [(3, 1), (2, 2), (7, 1)].
    - **Description:** Ensures that the first loop finds the minimum y-coordinate point (3, 1) and that it continues to the second loop.
    - **Expected Output:** The function should return (7, 1), as it has the maximum x-coordinate among points with the minimum y-coordinate.
- **Test Case 5: Two Iterations (Second Loop)**
    - **Input:** A vector with points where multiple points have the same minimum y-coordinate, e.g., [(1, 1), (6, 1), (3, 2)].
    - **Description:** Ensures the first loop finds (1, 1) as the initial minimum, while the second loop runs twice to evaluate other points with y = 1.
    - **Expected Output:** The function should return (6, 1), as it has the highest x-coordinate among points with the minimum y-coordinate.

Test Case	Input Vector	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 flowgraph, 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 for 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 is not detected by your test set derived in Step 2. Write/identify a mutation code for each of the three operations separately, i.e., by deleting the code, by inserting the code, and 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)]
Test Case 9	[(4,4), (4,3), (4,5), (4,9)]	Points with the same x-coordinate; checks for max y.	[(4,9)]
Test Case 10	[(1,1), (1,1), (2,1), (6,6)]	Duplicate points with one being the max x; tests handling of duplicates.	[(6,6)]