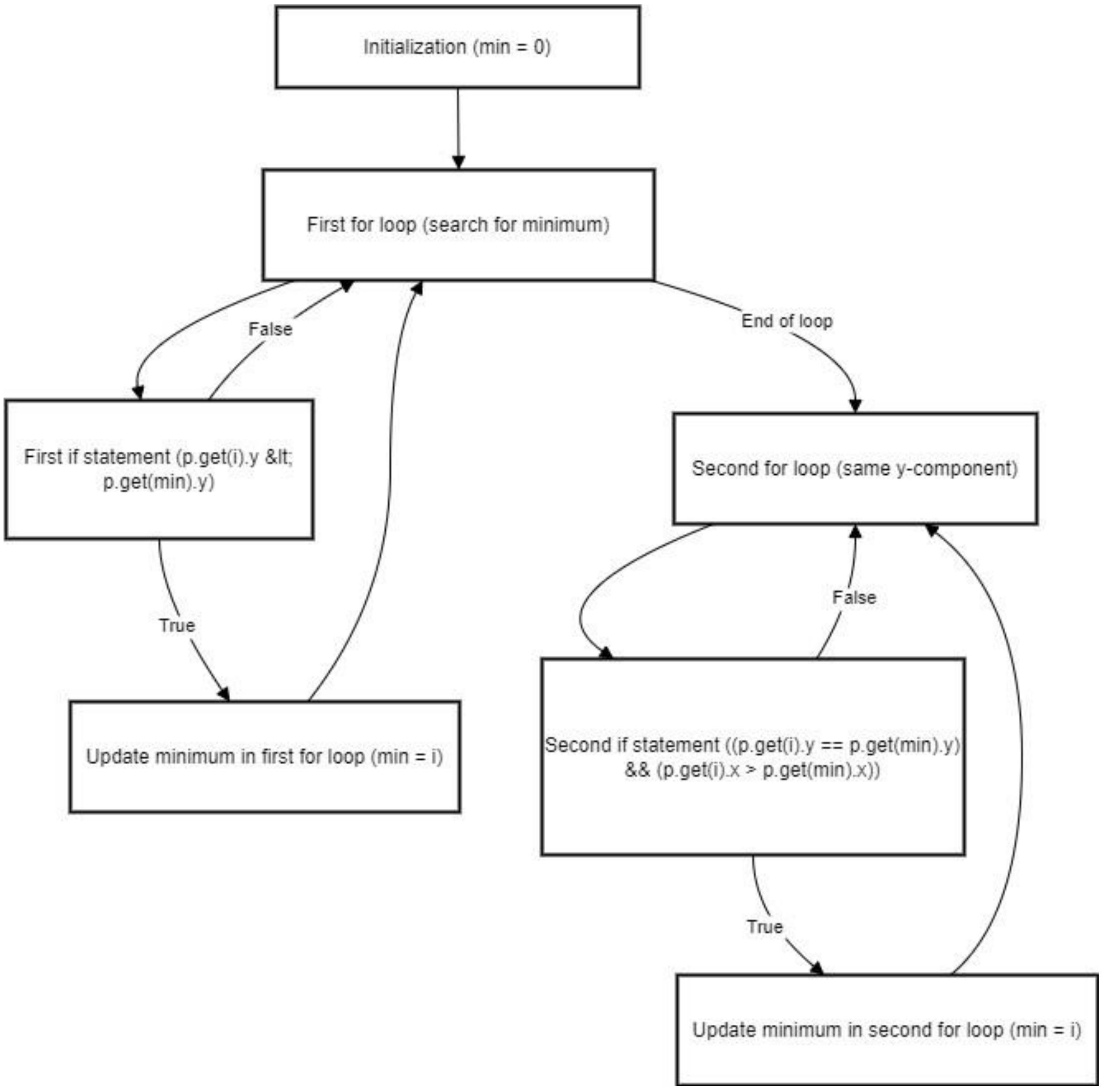
Software Engineering
Lab : 9

Name : Stuti Pandya
Id : 202201439

Ques 1 : Convert the code comprising the beginning of the doGraham method into a controlflow graph (CFG). You are free to write the code in any programming language.


Initialization (min = 0)
First for loop (search for minimum)
False
End of loop
First if statement (p.get(i).y &lt; p.get(min).y)
Second for loop (same y-component)
True
False
Update minimum in first for loop (min = i)
Second if statement ((p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x))
True
Update minimum in second for loop (min = i)

Ques 2 : Construct test sets for your flow graph that are adequate for the followingcriteria:

a. Statement Coverage.
b. Branch Coverage.
c. Basic Condition Coverage.

1. 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)]`)

2. 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.

3. 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:

- **Test Case 1:** Input vector containing only one point, such as [(1, 1)].
  - This case tests the scenario where the first condition p.get(i).y < p.get(min).y is not applicable since there are no other points for comparison.
- **Test Case 2:** Input vector with two points, where the second point has a lower y-coordinate than the first, such as [(1, 1), (2, 0)].
  - This case tests the scenario where the first condition is true (second point has a lower y value) and the second condition is false.
- **Test Case 3:** Input vector containing points with identical y-coordinates but differing x-coordinates, such as [(1, 1), (3, 1), (2, 1)].
  - This case tests the scenario where the second condition is true (selecting based on the x-coordinate when y-coordinates are the same).
- **Test Case 4:** Input vector with points that have identical y and x values, such as [(1, 1), (1, 1), (1, 1)].
  - This case tests the scenario where both conditions are false, as all points are identical in both y and x values.

Summary of Test Sets

Coverage Criterion	Test Case	Input
Statement Coverage	1	[(1, 1)]
	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:**
     - Without initializing min to 0, it may hold an undefined or random value,

potentially affecting the selection of the minimum point in the subsequent comparisons.

- **Mutation Outcome:** This mutation could cause the method to start with an incorrect index for min, potentially leading to an inaccurate minimum point selection.

2. **Condition Change Mutation**
   - **Mutation:** Modify the condition in the first if statement from < to <=, changing it to: if (((Point) p.get(i)).y <= ((Point) p.get(min)).y).
   - **Expected Effect:**
     - By allowing <= instead of <, the method may select points with the same y value rather than exclusively finding the point with the lowest y. This could interfere with the intended selection if multiple points have equal y values.
   - **Mutation Outcome:** The code may return a point with a smaller x value than intended if multiple points share the same y-coordinate, possibly affecting the final selection.
3. **Insertion Mutation**
   - **Mutation:** Add an extra min = i; statement at the end of the second for loop.
   - **Expected Effect:**
     - This insertion would set min to the last index in the list after the loop, which is incorrect because min should only represent the index of the minimum point found.
   - **Mutation Outcome:** The method could mistakenly select the last point as the minimum, especially in cases where the test does not verify that min accurately points to the lowest y-coordinate.

Test Case 1: Zero Iterations in Loops

- **Input:** An empty vector p.
- **Purpose:** Verifies that the function handles cases where there are no points, resulting in zero iterations for both loops.
- **Expected Output:** The function should either return an empty result or a specific indicator that no points were provided.

Test Case 2: Single Iteration in First Loop Only

- **Input:** A vector containing a single point, such as [(3, 4)].
- **Purpose:** Tests the case where the first loop executes exactly once because there is only one point, which is inherently the minimum.
- **Expected Output:** The function should return the sole point in p, which is (3, 4).

Test Case 3: Single Iteration in the Second Loop Only

- **Input:** A vector with two points that have identical y-coordinates but different x-coordinates, such as [(1, 2), (3, 2)].
- **Purpose:** Ensures the first loop identifies the minimum y-coordinate, while the second loop runs once to compare x-coordinates for points with the same y-value.
- **Expected Output:** The function should return the point with the highest x-coordinate, which is (3, 2).

Test Case 4: Two Iterations in the First Loop

- **Input:** A vector with multiple points, where at least two points share the same y-coordinate, such as [(3, 1), (2, 2), (5, 1)].
- **Purpose:** Confirms that the first loop can iterate twice to find the minimum y-coordinate while progressing through multiple points.
- **Expected Output:** The function should select (5, 1) as it has the highest x-coordinate among points with the minimum y-coordinate.

Test Case 5: Two Iterations in the Second Loop

- **Input:** A vector containing points where multiple points have the same minimum y-coordinate, such as [(1, 1), (4, 1), (3, 2)].
- **Purpose:** Tests that the first loop finds the initial minimum point and the second loop iterates twice to evaluate other points with the same y-coordinate.
- **Expected Output:** The function should return (4, 1), the point with the highest x-coordinate among points with the minimum y-val

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 pis 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, 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)]