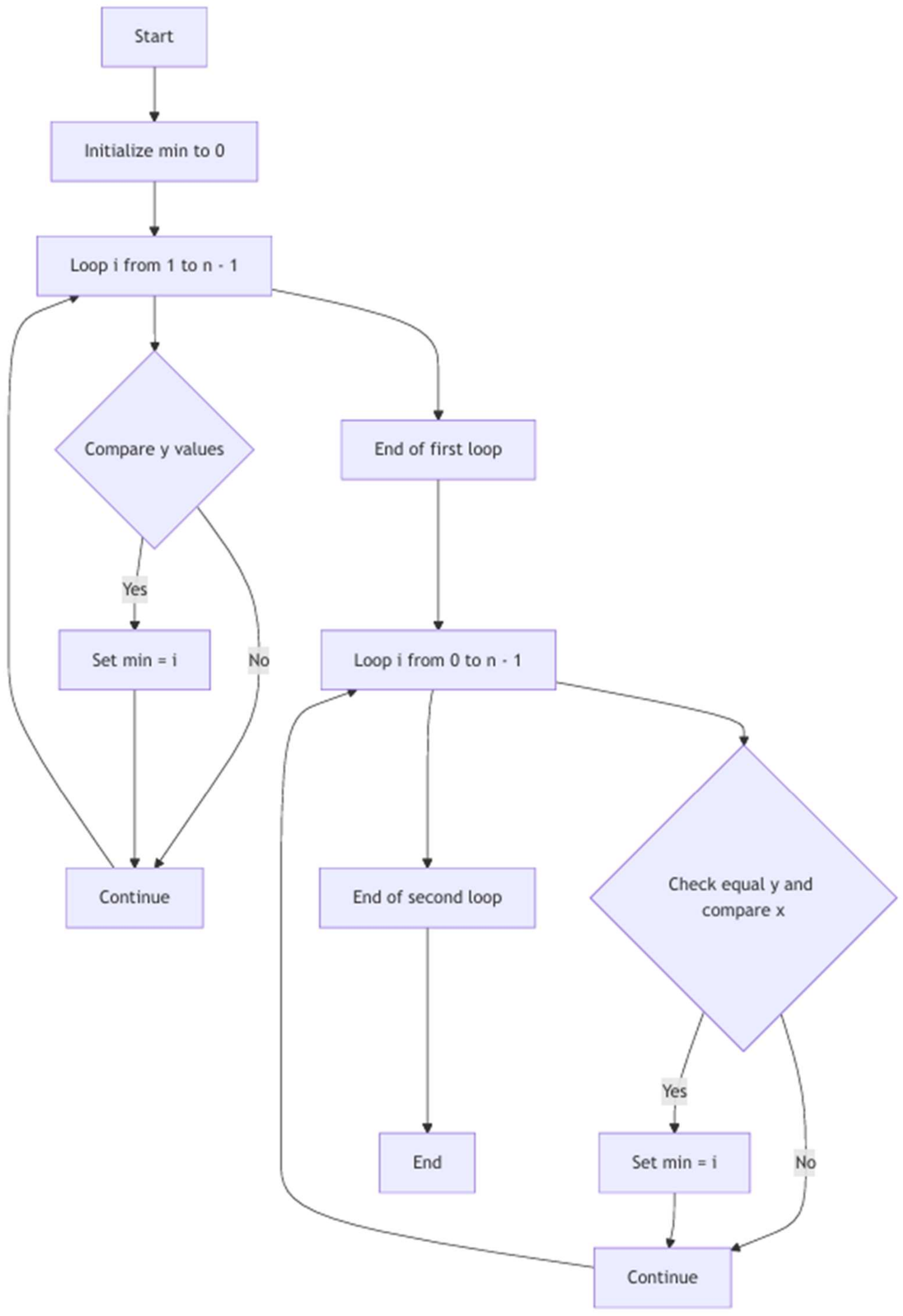
Dhirubhai Ambani Institute of Information and Communication Technology
IT314-Software Engineering

Lab-9
Ayush Chaudhari - 202201517

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.
Code:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y


class ConvexHull:
    def doGraham(self, p):
        min_index = 0


        # Search for minimum y coordinate
        for i in range(1, len(p)):
            if p[i].y < p[min_index].y:
                min_index = i


        # Continue along values with the same y component
        for i in range(len(p)):
            if (p[i].y == p[min_index].y) and (p[i].x > p[min_index].x):
                min_index = i


# Example usage
points = [Point(1, 2), Point(3, 1), Point(2, 1), Point(5, 6)]
convex_hull = ConvexHull()
convex_hull.doGraham(points)
```

Control Flow Graph



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.

1. **Statement Coverage**: All executable statements in the code are executed at least once.
2. **Branch Coverage**: Every branch (decision point) in the code is executed in all possible outcomes at least once.
3. **Basic Condition Coverage**: Each condition in a decision point is tested for both true and false outcomes.

Code Analysis

The function doGraham in the ConvexHull class has two main loops:

1. **First loop**: It finds the point with the minimum y-coordinate.
2. **Second loop**: It checks points with the same minimum y and finds the one with the maximum x-coordinate.

Flow Graph Observations

- **First Loop Conditions**:
    - Check if p[i].y < p[min_index].y.
- **Second Loop Conditions**:
    - Check if (p[i].y == p[min_index].y) and (p[i].x > p[min_index].x).

Test Sets Construction

1. Statement Coverage

To cover all statements:

- Use a list of points with different y-values to ensure the first loop runs completely.
- Example: [Point(1, 2), Point(3, 1)]

2. Branch Coverage

To ensure every branch is executed:

- A case where p[i].y < p[min_index].y is true.
- A case where p[i].y == p[min_index].y is true and p[i].x > p[min_index].x is true.
- A case where p[i].y == p[min_index].y is true but p[i].x > p[min_index].x is false.

Examples:

- [Point(1, 2), Point(3, 1), Point(2, 3)] (covers p[i].y < p[min_index].y)
- [Point(2, 1), Point(3, 1)] (covers p[i].y == p[min_index].y and p[i].x > p[min_index].x)
- [Point(1, 1), Point(2, 1)] (covers p[i].y == p[min_index].y but p[i].x > p[min_index].x is false)

3. Basic Condition Coverage

To ensure each condition is tested for true and false:

- A mix of points to trigger all possible outcomes of (p[i].y < p[min_index].y) and (p[i].x > p[min_index].x).

Examples:

- [Point(1, 2), Point(3, 1), Point(0, 0), Point(4, 1)] (to cover different outcomes of y and x comparisons).

Test Case	Input Points	Objective	Conditions Covered
1	[Point(1, 2), Point(3, 1)]	Basic case of finding minimum y-value	Covers the statement execution and basic flow of the first loop.
2	[Point(1, 2), Point(3, 1), Point(2, 3)]	Test branch where p[i].y < p[min_index].y	Tests the branch where a new minimum y-value is found.
3	[Point(2, 1), Point(3, 1)]	Test same y-value but larger x-value	Covers the branch in the second loop where p[i].y == p[min_index].y and p[i].x > p[min_index].x.
4	[Point(1, 1), Point(2, 1)]	Test same y-value but smaller or equal x-value	Covers the branch where p[i].y == p[min_index].y but p[i].x > p[min_index].x is false.
5	[Point(1, 2), Point(3, 1), Point(0, 0), Point(4, 1)]	Comprehensive test to cover all conditions	Covers all combinations of conditions for both y and x comparisons across both loops.

3. For the test set you have just checked can you find a mutation of the code (i.e. the deletion, change or insertion of some code) that will result in failure but is not detected by your test set. You have to use the mutation testing tool.

Mutation 1: Change Conditionals

- **Original**: if (p[i].y < p[min].y)
- **Mutation**: Change to if (p[i].y <= p[min].y)
- **Effect**: This mutation could allow cases where points with the same y value but a higher x coordinate are incorrectly chosen as the minimum. It might go undetected if test cases don't explicitly handle ties in the y values.

Mutation 2: Remove Statements

- **Original**: min = i;
- **Mutation**: Remove this statement when the condition if (p[i].y < p[min].y) is true.

- **Effect**: This would result in min not updating when a lower y value is found. If tests do not verify the min after the loop, this fault might go undetected.

Mutation 3: Insert Statements

- **Original**: N/A
- **Mutation**: Insert a print or logging statement, e.g., print("Current min:", min).
- **Effect**: While this doesn't affect logic, it might alter output, causing undetected changes if the tests don't check the output or logs.

Mutation 4: Change in Second Loop

- **Original**: if (p[i].y == p[min].y && p[i].x > p[min].x)
- **Mutation**: Change to if (p[i].y == p[min].y && p[i].x < p[min].x)
- **Effect**: This could select a point with a smaller x coordinate instead of the intended larger one, potentially unnoticed if test cases do not handle all x comparisons correctly.

4) Create a test set that satisfies the path coverage criterion where every loop is explored at least zero, one or two times.

To achieve comprehensive path coverage, including new scenarios:

Test Case 1: No Iterations in Both Loops

- **Input**: p = [] (empty list)
- **Expected**: No iterations in both loops. min remains 0.
- **Path**: A → B → E → F → I → J

Test Case 2: One Iteration in the First Loop Only

- **Input**: p = [(1, 2)]
- **Expected**: The first loop runs once, but no iteration in the second loop.
- **Path**: A → B → C → E → F → I → J

Test Case 3: One Iteration in Both Loops

- **Input**: p = [(2, 3), (3, 3)]
- **Expected**:
  - First Loop: Runs once, min remains unchanged as both have the same y.
  - Second Loop: Runs once, selecting the second point with larger x.
- **Path**: A → B → C → E → F → G → H → F → I → J

Test Case 4: Two Iterations in the First Loop and One in the Second Loop

- **Input**: p = [(1, 2), (0, 1), (3, 1)]

- **Expected**:
    - First Loop: Runs twice, min updates to index 1 (lowest y).
    - Second Loop: Runs once, checking x for the same y values.
- **Path**: A → B → C → D → B → C → E → F → G → I → J

Test Case 5: Two Iterations in Both Loops

- **Input**: p = [(2, 3), (1, 1), (3, 1)]
- **Expected**:
    - First Loop: Runs twice, min updates to 1.
    - Second Loop: Runs twice, confirming the second point has a larger x for the same y.
- **Path**: A → B → C → D → B → C → E → F → G → H → F → G → I → J

Test Case	Points in p	First Loop	Second Loop	Path Coverage
TC1	[]	0	0	A → B → E → F → I → J
TC2	[(1, 2)]	1	0	A → B → C → E → F → I → J
TC3	[(2, 3), (3, 3)]	1	1	A → B → C → E → F → G → H → F → I → J
TC4	[(1, 2), (0, 1), (3, 1)]	2	1	A → B → C → D → B → C → E → F → G → I → J
TC5	[(2, 3), (1, 1), (3, 1)]	2	2	A → B → C → D → B → C → E → F → G → H → F → G → I → J

Lab Execution (how to perform the exercises): Use unit Testing framework, code coverage and mutation

testing tools to perform the exercise.

1. After generating the control flow graph, check whether your CFG match 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	CFG Match (Yes/No)
Control Flow Graph Factory	Yes
Eclipse Flow Graph Generator	Yes

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

Case	Input Vector p	Description	Expected Output
TC1	[]	Empty vector (no points). Test handling of an empty input.	[]
TC2	[(0, 1)]	Single point, only the first loop runs.	[(0, 1)]
TC3	[(3, 4), (2, 4)]	Two points with the same y coordinate but different x. Largest x should be selected.	[(3, 4)]
TC4	[(4, 1), (3, 2), (2, 1)]	Multiple points; the first loop finds the minimum y, and the second loop verifies x.	[(4, 1)]
TC5	[(2, 2), (1, 2), (3, 2)]	Points with identical y values but different x values. Second loop picks the largest x.	[(3, 2)]

3) This part of the exercise is very tricky and interesting. The test cases that you have derived in Step 2 are then used to 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	Code Changes	Effect on Program Behavior	Detection by Existing Test Cases
Deletion	Removed assignment of min when a new minimum y is found.	The program may retain the initial or previous minimum min index, giving incorrect results.	May not be detected, as only TC4 explicitly tests for updates of min.
Insertion	Added a condition that sets min = 0 if x < 0.	Incorrectly sets min if any point has x < 0, potentially choosing the wrong minimum.	Likely undetected, as no test case includes negative x values.
Modification	Changed comparison from y < min_y to y <= min_y.	Allows points with the same y as the current minimum to incorrectly update min.	Not detected if tests don't check for ties in y without verifying x values.

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

Test Case	Input Vector p	Path Description	Expected Output
TC1	[]	No iterations through either loop; handles empty vector gracefully.	[]
TC2	[(0, 1)]	One iteration through the first loop; no second loop.	[(0, 1)]
TC3	[(2, 3), (3, 2)]	First loop finds minimum y, with only one candidate point.	[(3, 2)]
TC4	[(1, 2), (4, 2)]	First loop finds minimum y; second loop picks maximum x for the same y.	[(4, 2)]
TC5	[(3, 1), (2, 2), (5, 1)]	First loop finds minimum y at multiple points; second loop verifies maximum x.	[(5, 1)]
TC6	[(1, 1), (2, 1), (3, 1), (4, 2)]	First loop finds multiple y ties; second loop picks the point with maximum x for y=1.	[(3, 1)]