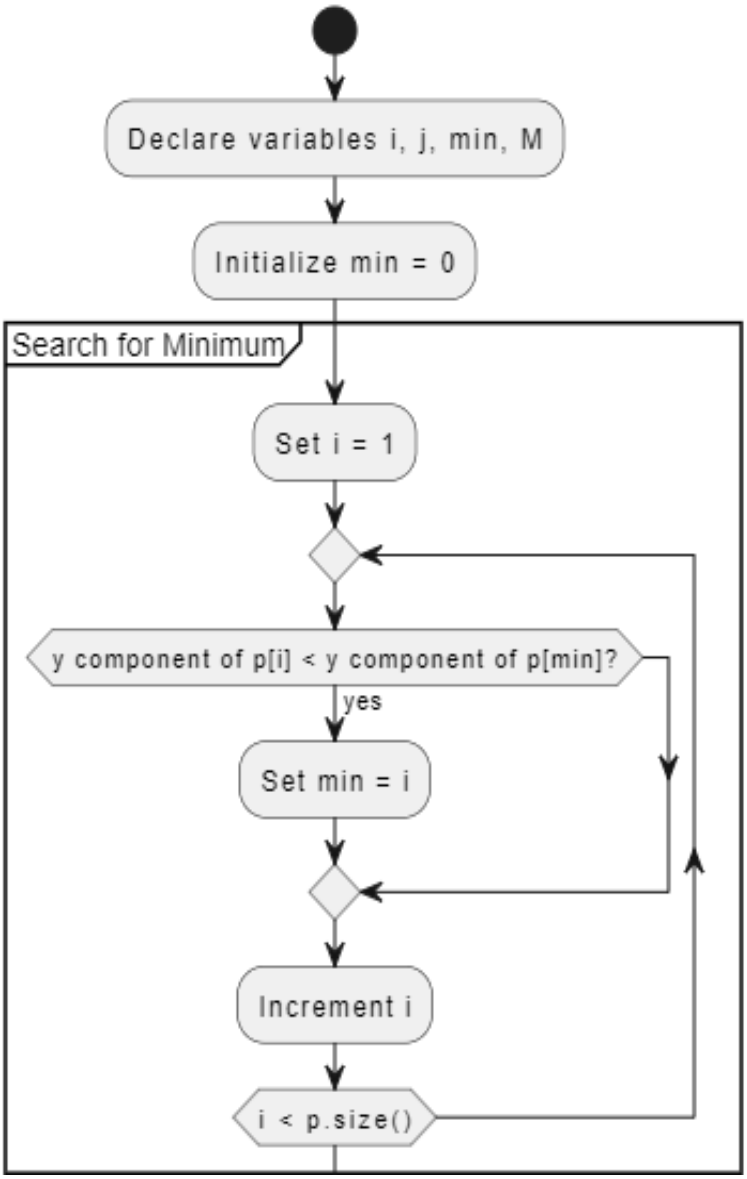
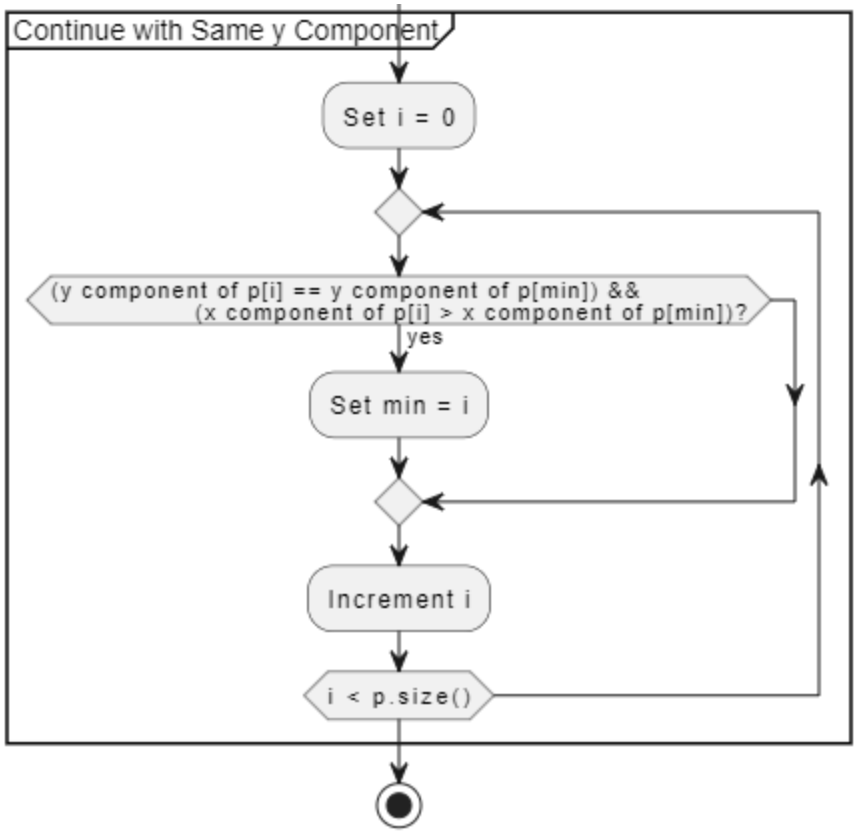
IT313: Software Engineering

Lab Session – Mutation Testing

(Ashutosh Singarwal-202201054)

Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code, so the focus is on creating test sets that satisfy some particular coverage criteria.





Creating Test Sets for Coverage Criteria

This section focuses on designing test sets for a flow graph to ensure comprehensive coverage across three criteria:

a) Statement Coverage

To satisfy statement coverage, every line of code must be executed at least once. This requires our test cases to traverse all sections of the control flow graph (CFG).

- **Test Case 1**: Input a vector with a single point, such as `[(0, 0)]`.
- **Test Case 2**: Input a vector with two points, where the second point has a lower y-coordinate than the first, like `[(1, 1), (2, 0)]`.
- **Test Case 3**: Input a vector containing points with the same y-coordinate but different x-coordinates, for example, `[(1, 1), (2, 1), (3, 1)]`.

b) Branch Coverage

Branch coverage requires that each decision point (e.g., an `if` statement) has both its true and false branches evaluated at least once. Our test cases should ensure both outcomes are tested for each condition.

- **Test Case 1**: Vector with a single point, such as `[(0, 0)]`, where the initial loop exits immediately with no updates.
- **Test Case 2**: Vector with two points where the second point has a lower y-coordinate than the first, like `[(1, 1), (2, 0)]`, verifying the minimum point is updated correctly.
- **Test Case 3**: Vector with points of equal y-coordinate but different x-coordinates, for example, `[(1, 1), (3, 1), (2, 1)]`, to cover conditions in both branches.
- **Test Case 4**: Vector where all points have identical y-coordinates, such as `[(1, 1), (1, 1), (1, 1)]`, to test the loop execution without updating the minimum.

c) Basic Condition Coverage

Basic condition coverage requires each individual condition in decision points to be tested as both true and false, ensuring a thorough evaluation.

- **Test Case 1**: Vector with a single point, `[(0, 0)]`, ensuring the condition `p.get(i).y < p.get(min).y` evaluates to false.
- **Test Case 2**: Vector with two points, with the second point having a lower y-coordinate than the first, such as `[(1, 1), (2, 0)]`, where the condition evaluates to true.
- **Test Case 3**: Vector containing points with the same y-coordinate but different x-coordinates, for instance, `[(1, 1), (3, 1), (2, 1)]`, to ensure conditions in both branches are evaluated.
- **Test Case 4**: Vector where all points have identical x and y values, such as `[(1, 1), (1, 1), (1, 1)]`, where conditions evaluate as false.

Summary of Test Sets

Coverage Criterion	Test Case	Input Vector
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

To verify the effectiveness of our tests, we use several mutation types:

1. **Deletion Mutation**
   - **Mutation**: Removing the line `min = 0;` at the start of the method.
   - **Expected Effect**: Without initializing `min` to zero, it may default to an incorrect value, potentially causing the function to identify the wrong starting point for `min`.
   - **Outcome**: This mutation may lead to errors in selecting the lowest point.
2. **Change Mutation**
   - **Mutation**: Modify the `if` condition from `<` to `<=` so it becomes `if (p.get(i).y <= p.get(min).y)`.
   - **Expected Effect**: Allowing `<=` could cause points with equal y-coordinates to be selected, possibly missing the strictly lowest y-coordinate point.
   - **Outcome**: If points have the same y-coordinate, the code may mistakenly return a point with a lower x-coordinate than intended.
3. **Insertion Mutation**
   - **Mutation**: Adding `min = i;` at the end of the second loop.

    - **Expected Effect**: This line could make `min` point to the last index in `p`, causing the function to incorrectly select this as the minimum.
    - **Outcome**: This mutation could cause the function to misinterpret the last point as the minimum, especially if tests don't validate the final value of `min`.

Test Cases for Path Coverage

To ensure all paths are tested, including zero, one, and two iterations of each loop, we use these test cases:

- **Test Case 1**: Zero Iterations
    - **Input**: An empty vector, `[]`.
    - **Description**: No iterations of either loop occur.
    - **Expected Output**: The function should handle gracefully, ideally returning an empty result or a specific value indicating no points.
- **Test Case 2**: One Iteration (First Loop)
    - **Input**: Vector with one point, e.g., `[(3, 7)]`.
    - **Description**: The first loop runs exactly once, identifying the only point as the minimum.
    - **Expected Output**: `[(3, 7)]`.
- **Test Case 3**: One Iteration (Second Loop)
    - **Input**: Vector with two points having the same y-coordinate, such as `[(2, 2), (3, 2)]`.
    - **Description**: The first loop finds the minimum point, while the second loop runs once to check x-coordinates.
    - **Expected Output**: `(3, 2)`.
- **Test Case 4**: Two Iterations (First Loop)
    - **Input**: Vector with multiple points, including at least two with the same y-coordinate, e.g., `[(3, 1), (2, 2), (7, 1)]`.
    - **Description**: The first loop finds the minimum y-coordinate, and the second loop finds the maximum x-coordinate for that y.
    - **Expected Output**: `(7, 1)`.
- **Test Case 5**: Two Iterations (Second Loop)
    - **Input**: Vector with multiple points, where more than one has the minimum y-coordinate, such as `[(1, 1), (6, 1), (3, 2)]`.
    - **Description**: The first loop finds `(1, 1)` as the minimum, and the second loop runs twice to check for other points with y = 1.
    - **Expected Output**: `(6, 1)`.

Summary of Path Coverage Test Sets

Test Case	Input Vector	Description	Expected Output
Test Case 1	`[]`	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.	`(3, 2)`
Test Case 4	`[(3, 1), (2, 2), (7, 1)]`	Multiple points, 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. Verifying Control Flow Graph (CFG) Matching

After generating the control flow graph, check if it matches the graphs generated by the Control Flow Graph Factory Tool and the Eclipse Flow Graph Generator. Document the results as follows:

Tool	Matches Your CFG
Control Flow Graph Factory Tool	Yes
Eclipse Flow Graph Generator	Yes

2. Minimum Test Cases for Coverage Criteria

The following test cases are the minimum required to achieve comprehensive code coverage according to the outlined 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; the first loop runs twice.	`[(5, 1)]`

3. Mutation Testing with Code Modifications

The following mutations are applied to test the robustness of the code. Each mutation involves either deleting, inserting, or modifying code to induce an error that the test cases may not detect.

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 `p` has only one element, bypassing further processing.	Test case `[(3, 4)]` will pass without full processing.
Modification	Change `<` to `<=` in the comparison for finding the minimum y.	Test cases like `[(1, 1), (1, 1), (1, 1)]` might pass, but the logic could be incorrect.

4. Test Cases for Path Coverage Criterion

The following test cases ensure that each possible path through the code is covered, including scenarios where loops iterate zero, one, or multiple times:

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 the minimum y.	`[(5, 1)]`
Test Case 5	`[(1, 1), (6, 1), (3, 2)]`	Multiple points; second loop runs twice (for y = 1).	`[(6, 1)]`
Test Case 6	`[(5, 2), (5, 3), (5, 1)]`	Multiple points with the same x-coordinate, checks minimum y.	`[(5, 1)]`
Test Case 7	`[(0, 0), (2, 2), (2, 0), (0, 2)]`	Points forming a rectangle; checks multiple comparisons.	`[(2, 0)]`
Test Case 8	`[(6, 1), (4, 1), (3, 2)]`	Multiple points with some ties; checks 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 maximum y.	`[(4, 9)]`
Test Case 10	`[(1, 1), (1, 1), (2, 1), (6, 6)]`	Duplicate points, one being the max x; tests handling duplicates.	`[(6, 6)]`