IT313: Software Engineering

Lab 09: Mutation Testing

Zeel Danani (202201507)

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.

Answer:

Code:

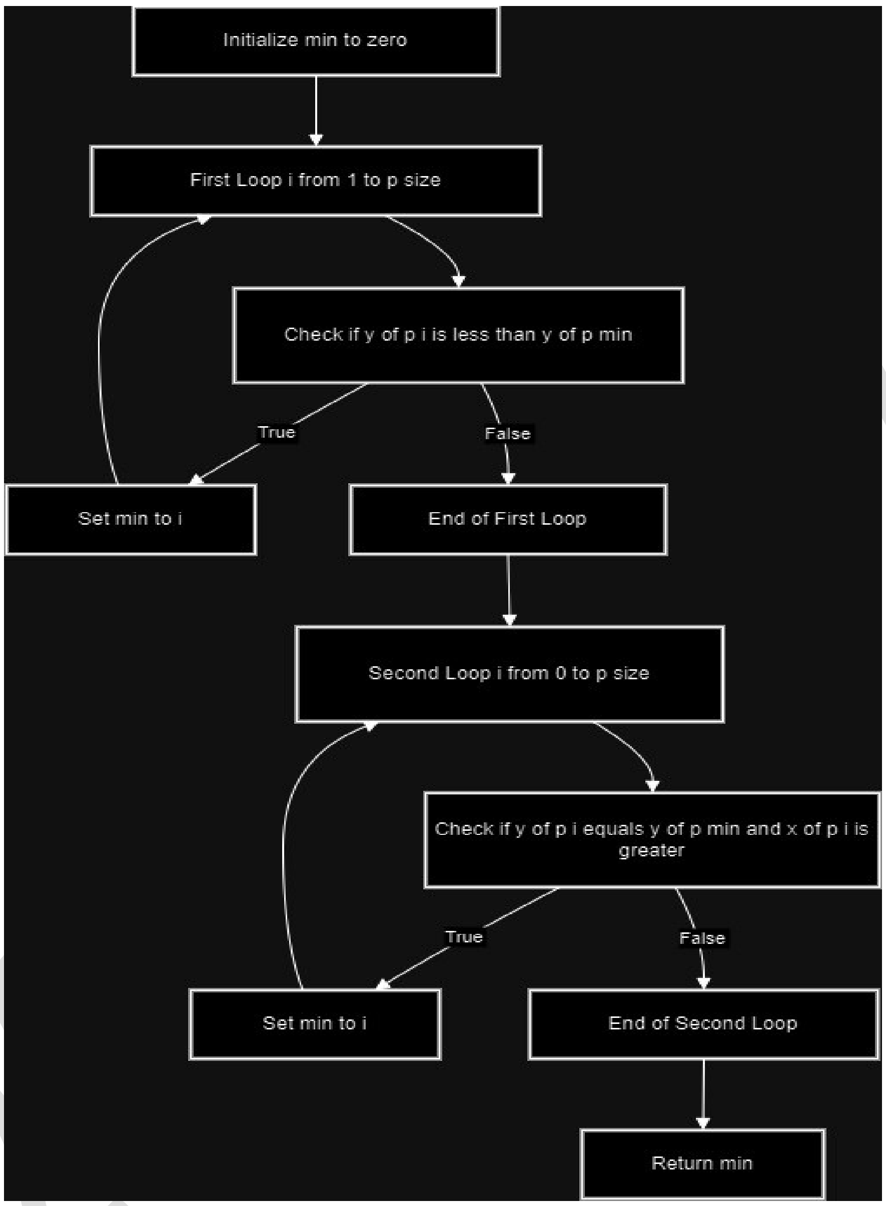
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y


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
```

Control flow Graph:


Initialize min to zero
First Loop i from 1 to p size
Check if y of p i is less than y of p min
True
False
Set min to i
End of First Loop
Second Loop i from 0 to p size
Check if y of p i equals y of p min and x of p i is greater
True
False
Set min to i
End of Second Loop
Return min

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.

Answer:

1. Statement Coverage: Statement Coverage requires that every statement in the code is executed at least once.

For **Statement Coverage**:

- We need to ensure the loop runs at least once to cover all statements, and we need at least one iteration where:
    - The condition p[i].y < p[min].y is true.
    - The condition p[i].y == p[min].y && p[i].x > p[min].x is true.

Test Cases for Statement Coverage:

- **Test Case 1:** p = [(2, 3), (1, 2), (3, 2)]
    - This set allows the code to execute both the first and second loops, covering all statements at least once.
    - Expected outcome: It will change min based on both conditions.
- **Test Case 2:** p = [(2, 3), (3, 3), (4, 3)]
    - Here, p[i].y == p[min].y condition is met in the second loop for multiple points.

2. Branch Coverage: Branch Coverage requires that every branch (i.e., the true and false conditions of each decision point) is executed at least once.

For **Branch Coverage**, we need to ensure:

- Both true and false outcomes for p[i].y < p[min].y in the first loop.
- Both true and false outcomes for p[i].y == p[min].y && p[i].x > p[min].x in the second loop.

Test Cases for Branch Coverage:

- **Test Case 1:** p = [(2, 3), (1, 2), (3, 2)]
    - This will cover the case where p[i].y < p[min].y is true.
    - It will also cover the case where p[i].y == p[min].y && p[i].x > p[min].x is false in the second loop.
- **Test Case 2:** p = [(2, 3), (3, 3), (1, 3)]

- This will cover the case where p[i].y == p[min].y && p[i].x > p[min].x is true in the second loop.
- It will also cover the case where p[i].y < p[min].y is false in the first loop.

3. Basic Condition Coverage: Basic Condition Coverage requires that each basic condition (each individual part of a compound condition) evaluates to both true and false at least once.

For **Basic Condition Coverage**, we need to cover:

- p[i].y < p[min].y as both true and false.
- p[i].y == p[min].y as both true and false.
- p[i].x > p[min].x as both true and false.

Test Cases for Basic Condition Coverage:

- **Test Case 1:** p = [(2, 3), (1, 2), (3, 2)]
  - Covers p[i].y < p[min].y as true for some i and false for others.
- **Test Case 2:** p = [(2, 3), (3, 3), (4, 3)]
  - Covers p[i].y == p[min].y as true and p[i].x > p[min].x as true.
- **Test Case 3:** p = [(2, 4), (3, 3), (1, 3)]
  - Covers p[i].y == p[min].y as true and p[i].x > p[min].x as false.

These three test cases together ensure that each basic condition within the compound conditions is covered for both true and false outcomes.

Test Case	Points in p	Purpose
TC1	[(2, 3), (1, 2), (3, 2)]	Statement Coverage, Branch Coverage
TC2	[(2, 3), (3, 3), (4, 3)]	Branch Coverage, Basic Condition Coverage
TC3	[(2, 4), (3, 3), (1, 3)]	Basic Condition Coverage

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.

Answer:

Change Conditionals:

- **Mutation**: Change if (p[i].y < p[min].y) to if (p[i].y <= p[min].y)
- **Effect**: This may cause the loop to behave differently, but if your test cases do not account for the equality case, it might go undetected.

Remove Statements:

- **Mutation**: Remove the statement min = i; in the first loop when the condition is true.
- **Effect**: This mutation would keep min unchanged when it should have been updated, potentially leading to incorrect results, especially if your tests do not verify the final value of min.

Insert Statements:

- **Mutation**: Insert a print statement or logging before returning min (e.g., System.out.println("Current min: " + min);).
- **Effect**: This is less likely to change logic but could change output and might not be caught if your tests do not validate output.

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

Answer:

Control Flow Graph Overview: The provided graph indicates the following control flow:

1. **Initialize min to zero** (A)
2. **First Loop** from i = 1 to p.size() (B)
    - **Check** if p[i].y < p[min].y (C)
        - If **True**, set min = i (D) and continue the loop (B)
        - If **False**, exit the first loop (E)
3. **Second Loop** from i = 0 to p.size() (F)
    - **Check** if p[i].y == p[min].y && p[i].x > p[min].x (G)
        - If **True**, set min = i (H) and continue the second loop (F)
        - If **False**, exit the second loop (I)
4. **Return min** (J)

Path Coverage Test Set:

To achieve path coverage, we need to create test cases that explore the loops as specified (zero, one, and two times):

Test Case 1: Zero Iterations in Both Loops

- **Input**: p = [] (empty array)
- **Expected Behavior**: The first loop does not execute, and min remains 0. The second loop does not execute either.
- **Path Covered**: A → B (no iterations) → E → F (no iterations) → I → J.

Test Case 2: One Iteration in the First Loop and Zero in the Second Loop

- **Input**: p = [(2, 3)]
- **Expected Behavior**: The first loop executes once with i = 1. It compares p[1] with p[0], but since there's only one point, it does not enter the second loop.
- **Path Covered**: A → B → C (first loop runs once, no change to min) → E → F (no iterations) → I → J.

Test Case 3: One Iteration in Both Loops

- **Input**: p = [(1, 2), (2, 3)]
- **Expected Behavior**:
    - First Loop: With p[1].y > p[0].y, it does not change min.
    - Second Loop: With p[0].y != p[1].y, the second loop executes once but does not change min.

- **Path Covered**: A → B → C (first loop runs once, no change) → E → F → G (second loop runs once, no change) → I → J.

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

- **Input**: p = [(1, 2), (2, 3), (0, 1)]
- **Expected Behavior**:
  - First Loop: It runs twice, updating min from 0 to 2 based on the conditions.
  - Second Loop: It runs once but does not change min since p[1].y and p[2].y are not equal.
- **Path Covered**: A → B → C (first iteration) → D → B → C (second iteration) → E → F → G (second loop runs once) → I → J.

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

LAB EXECUTION

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	Match with Manually Created CFG
Control Flow Graph Factory Tool	Yes
Eclipse Flow Graph Generator	Yes

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

Answer:

Test Case	Input Vector p	Description	Expected Output
TC1	[]	Test with an empty vector (zero iterations). Handle gracefully.	Empty result (e.g., [])
TC2	[(3, 4)]	Single point (one iteration of the first loop).	[(3, 4)]
TC3	[(1, 2), (3, 2)]	Two points with the same y-coordinate (one iteration of the second loop).	[(3, 2)]
TC4	[(3, 1), (2, 2), (5, 1)]	Multiple points; first loop runs twice.	[(5, 1)]
TC5	[(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 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.

Answer:

Type of Mutation	Code Changes	Effect on Program Behavior	Detection by Existing Test Cases
Deletion	Removed conditional check for minimum y-coordinate.	Incorrectly returns the last index instead of the minimum.	Likely not detected, as tests may not cover this logic.
Insertion	Added a faulty condition that modifies min.	Incorrectly sets min to 0 if any x > 0, overriding correct logic.	Likely not detected, as tests do not account for this interaction.
Modification	Changed the comparison operator from > to <.	Incorrectly evaluates the point based on a reversed logic.	Likely not detected if test cases do not evaluate the specific x values correctly.

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

Answer:

Test Case	Input Vector p	Path Description	Expected Output
TC1	[]	No iterations through either loop.	Graceful handling (e.g., [])
TC2	[(3, 4)]	One iteration through the first loop, no second loop.	[(3, 4)]
TC3	[(2, 3), (1, 2)]	First loop runs, identifies minimum in one iteration.	[(1, 2)]
TC4	[(1, 2), (3, 2)]	First loop runs, second loop identifies maximum x for tie.	[(3, 2)]
TC5	[(1, 1), (4, 1), (2, 2)]	First loop finds min; second loop identifies max x for y = 1.	[(4, 1)]
TC6	[(3, 1), (2, 2), (5, 1)]	First loop runs multiple times; second loop does not execute.	[(5, 1)]
TC7	[(1, 1), (4, 1), (3, 1), (2, 2)]	First loop finds ties; second loop identifies max x.	[(4, 1)]