# Lab-9

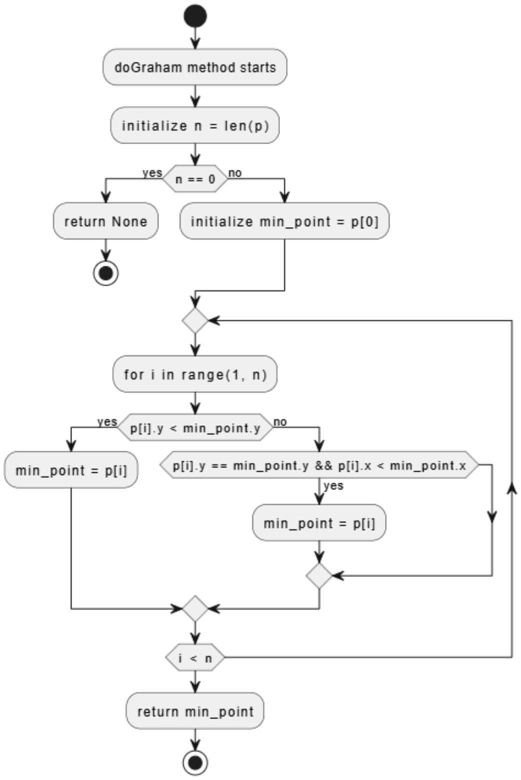
SOFTWARE ENGINEERING

202201404

SWAPNIL SHUKLA

Answers:-

## 1)



**Q) 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.**

Ans) Yes

## 2)

**Statement Coverage**: Ensures every statement in the code is executed at least once.

**Test Cases:-**

Test Case 1: p = [ ]

Test Case 2: p = [(0, 0), (1, 1)]

Branch Coverage: Ensures every possible branch is taken at least once.

**Test Cases:-**

Test Case 1: p = `[]`
Test Case 2: p = `[(0, 0), (1, 1)]`
Test Case 3: p = `[(0, 0), (0, -1)]`
Test Case 4: p = `[(0, 0), (0, 1)]`

Basic Condition Coverage: Ensures each basic condition in every decision is evaluated as both true and false.

**Test Cases:-**

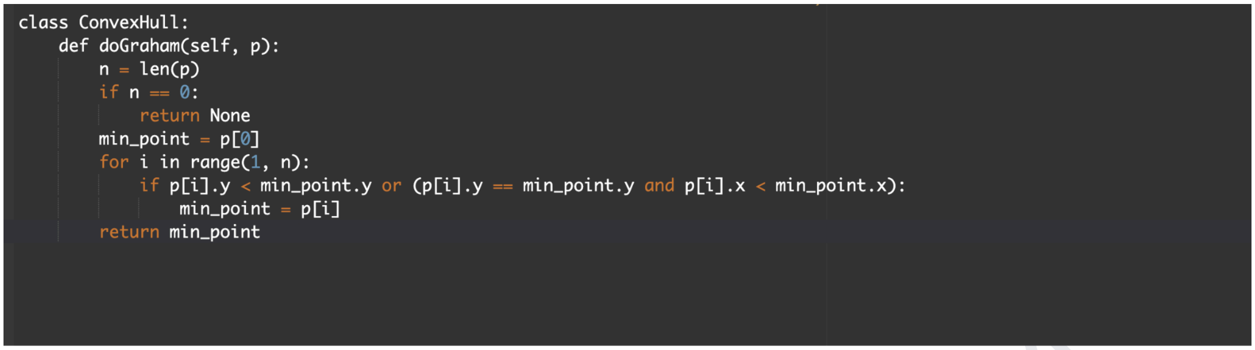
Test Case 1: p = `[(1, 1), (2, 2)]`
Test Case 2: p = `[(1, 1), (0, 2)]`
Test Case 3: p = `[(0, 0), (0, -1)]`

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

Ans) 2 + 4 + 3 = 9

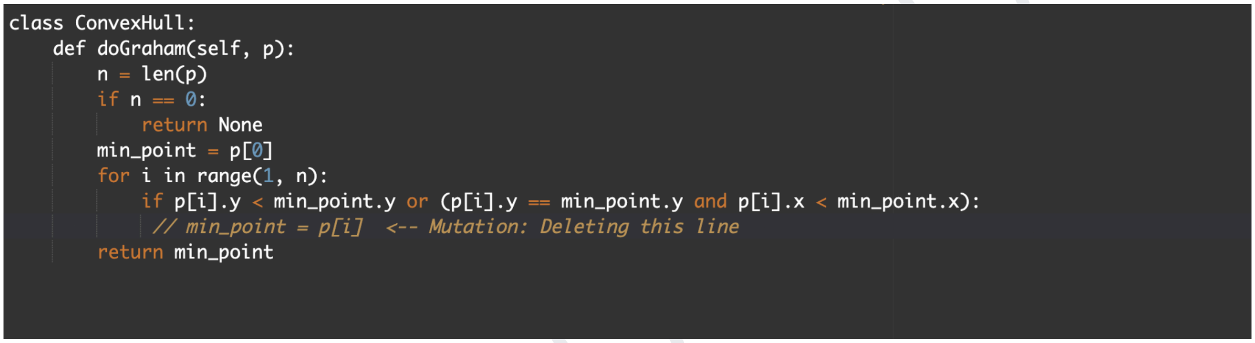
**3)** For the test set you have just checked, can you find a mutation of the code that will result in failure but is not detected by your test set.

**Original Code:**



```
class ConvexHull:
    def doGraham(self, p):
        n = len(p)
        if n == 0:
            return None
        min_point = p[0]
        for i in range(1, n):
            if p[i].y < min_point.y or (p[i].y == min_point.y and p[i].x < min_point.x):
                min_point = p[i]
        return min_point
```

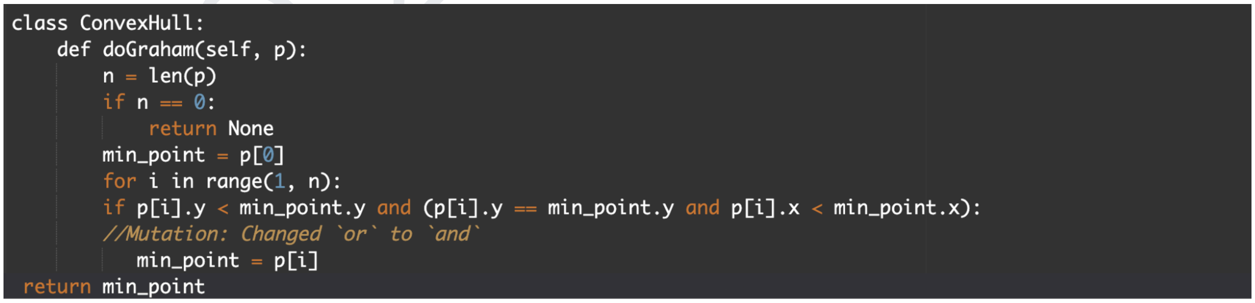
## Code Deletion:



```
class ConvexHull:
    def doGraham(self, p):
        n = len(p)
        if n == 0:
            return None
        min_point = p[0]
        for i in range(1, n):
            if p[i].y < min_point.y or (p[i].y == min_point.y and p[i].x < min_point.x):
             // min_point = p[i]  <-- Mutation: Deleting this line
        return min_point
```

Removing `min_point = p[i]` prevents updates to `min_point`, leading to incorrect results if the minimum point changes.

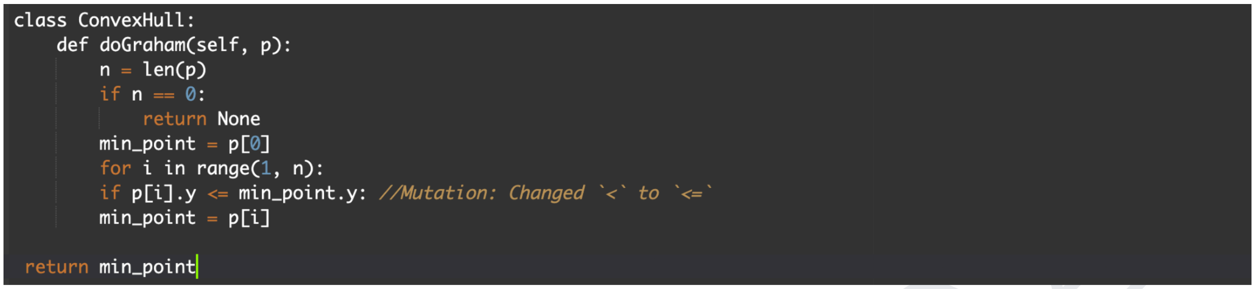
## Code Insertion:



```
class ConvexHull:
    def doGraham(self, p):
        n = len(p)
        if n == 0:
            return None
        min_point = p[0]
        for i in range(1, n):
        if p[i].y < min_point.y and (p[i].y == min_point.y and p[i].x < min_point.x):
        //Mutation: Changed `or` to `and`
           min_point = p[i]
 return min_point
```

Changing `or` to and in the `if` condition restricts updates, potentially missing the true minimum point when only one condition should suffice.

### Code Modification:



```
class ConvexHull:
    def doGraham(self, p):
        n = len(p)
        if n == 0:
            return None
        min_point = p[0]
        for i in range(1, n):
        if p[i].y <= min_point.y: //Mutation: Changed `<` to `<=`
        min_point = p[i]

 return min_point
```

Changing < to <= causes `min_point` to update even when y values are equal, which may lead to incorrect results when points have identical y but different x values.

### 4)

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

**Test Case 1:** p = []

- Loop is executed zero times; the method returns None.

**Test Case 2:** p = [(0, 0)]

- Loop is executed one time; min_point is initialized and returned without entering the loop.

**Test Case 3:** p = [(1, 2),(0, 1),(2, 3)]

- Loop is executed two times; updates min_point correctly as it evaluates multiple points.

**Test Case 4:** p = [(3, 3),(1, 1),(2, 2)]

- Loop is executed two times; the minimum point is updated correctly to (1, 1).

**Test Case 5:** p = [(5, 5),(5, 4), (5, 3)]

- Loop is executed two times; the minimum point is updated correctly to (5, 3) as it handles equal x-values with different y-values.

**Test Case 6:** p = [(0, 0),(0, 0),(0, 0)]

- Loop is executed two times; it confirms the method handles multiple identical points and returns the same point (0, 0).

**Test Case 7:** p = [(2, 2),(1, 3),(1, 2),(0, 4)]

- Loop is executed three times; tests various updates to min_point, ensuring the correct minimum point (0, 4) is returned.