`IT-314

Software Lab

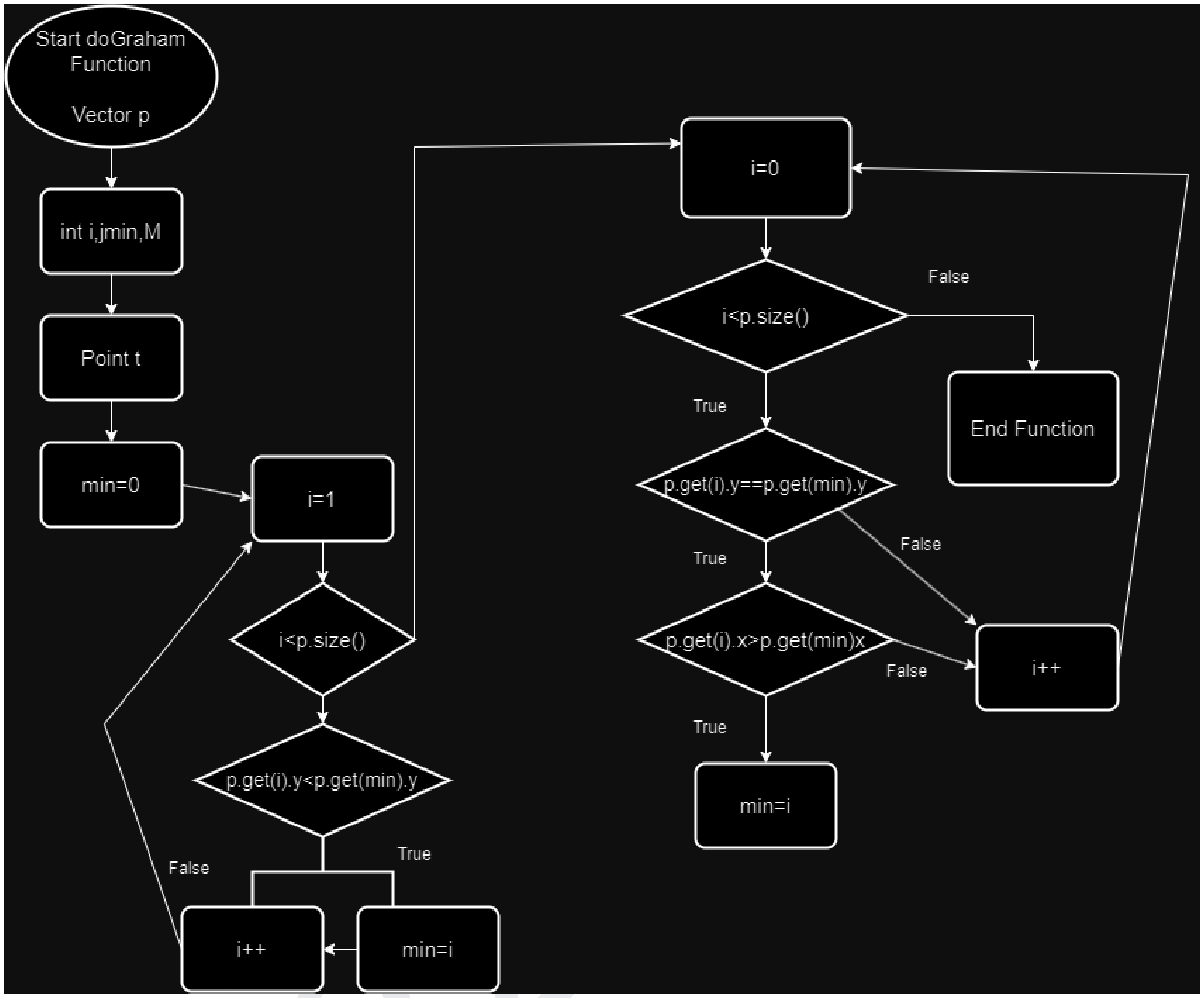
Mutation Testing and Control Flow Graph

Rakshit Pandhi

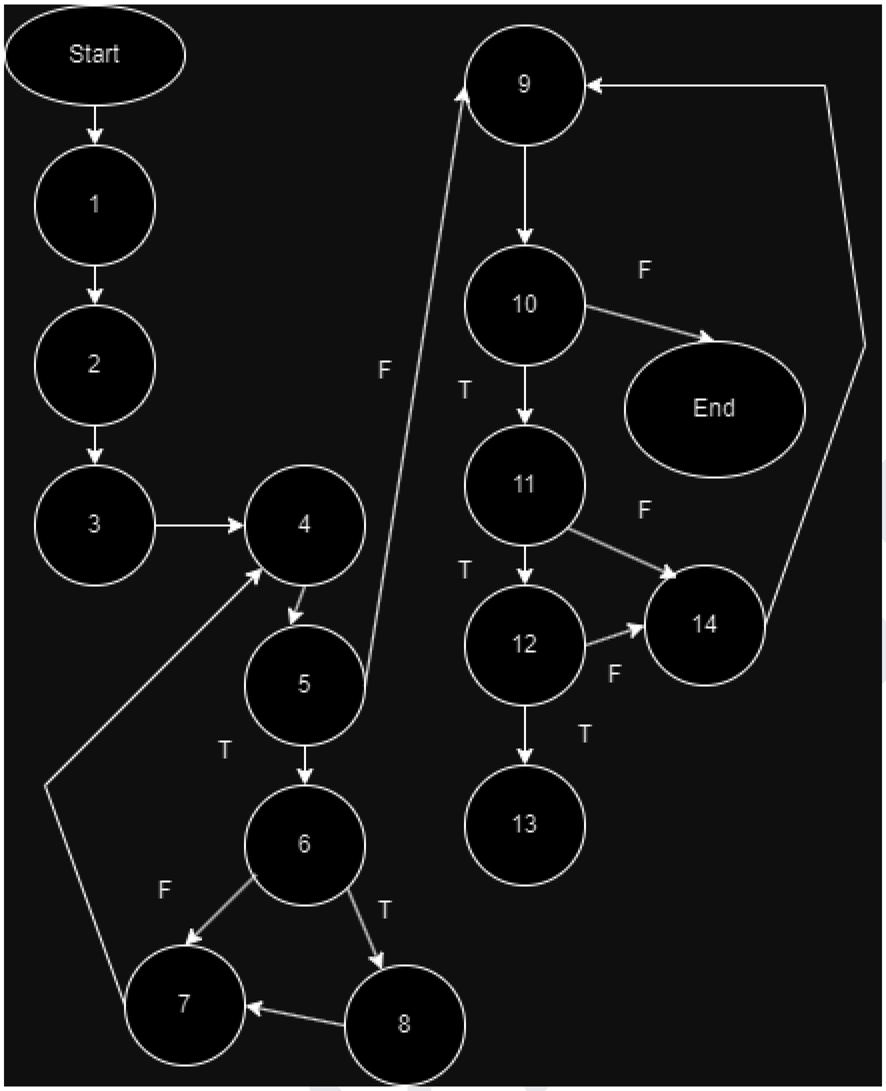
202201426

Question 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).



Start doGraham Function
Vector p
int i,jmin,M
Point t
min=0
i=1
i<p.size()
p.get(i).y<p.get(min).y
True
False
i++
min=i
i=0
i<p.size()
False
End Function
True
p.get(i).y==p.get(min).y
False
True
p.get(i).x>p.get(min)x
False
i++
True
min=i



Question 2

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

Statement Coverage

Test Case 1: Single Point

- Input: points = [Point(0, 0)]
- Expected Result: min_index = 0

- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (False) → End Function

Test Case 2: Two Points, Distinct Y-coordinates

- Input: points = [Point(0, 0), Point(1, 2)]
- Expected Result: min_index = 0
- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (True) → p.get(i).y != p.get(min).y (False) → i++ → i < p.size() (False) → End Function

---

Branch Coverage

Test Case 1: Two Points, Same Y-coordinates, Different X-coordinates

- Input: points = [Point(0, 0), Point(2, 0)]
- Expected Result: min_index = 0
- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (True) → p.get(i).y == p.get(min).y (True) → p.get(i).x > p.get(min).x (True) → min = i → i++ → i < p.size() (False) → End Function

Test Case 2: Two Points, Different Y-coordinates

- Input: points = [Point(0, 0), Point(1, 1)]
- Expected Result: min_index = 0

- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (True) → p.get(i).y != p.get(min).y (False) → i++ → i < p.size() (False) → End Function

Test Case 3: Single Point

- Input: points = [Point(0, 0)]
- Expected Result: min_index = 0
- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (False) → End Function

---

Basic Condition Coverage

Test Case 1: Two Points, Same Y-coordinates, Larger X-coordinate

- Input: points = [Point(0, 0), Point(2, 0)]
- Expected Result: min_index = 1
- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (True) → p.get(i).y == p.get(min).y (True) → p.get(i).x > p.get(min).x (True) → min = i → i++ → i < p.size() (False) → End Function

Test Case 2: Two Points, Same Y-coordinates, Smaller X-coordinate

- Input: points = [Point(2, 0), Point(0, 0)]
- Expected Result: min_index = 1

- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (True) → p.get(i).y == p.get(min).y (True) → p.get(i).x > p.get(min).x (False) → i++ → i < p.size() (False) → End Function

Test Case 3: Two Points, Different Y-coordinates

- Input: points = [Point(0, 0), Point(1, 1)]
- Expected Result: min_index = 0
- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (True) → p.get(i).y != p.get(min).y (False) → i++ → i < p.size() (False) → End Function

Test Case 4: Single Point

- Input: points = [Point(0, 0)]
- Expected Result: min_index = 0
- Path Covered: Start → int i, j, min, M → Point t → min=0 → i=1 → i < p.size() (False) → End Function

---

Statement Coverage

Test Case 1: Simple Path Without Loop

- Path: Start → 1 → 2 → 3 → 4 (F) → 9 → 10 (F) → End

Test Case 2: Path with True Conditions and Loops

- Path: Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (F) → 8 → 6 → 7 (T) → 9 → 10 (T) → 11 (T) → 12 (F) → 14 → End

---

Branch Coverage

Test Case 1: Branch with Early Exit

- Path: Start → 1 → 2 → 3 → 4 (F) → 9 → 10 (F) → End

Test Case 2: Path with True Conditions and Loops

- Path: Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (F) → 8 → 6 → 7 (T) → 9 → 10 (T) → 11 (T) → 12 (F) → 14 → End

Test Case 3: Path with Mixed Conditions

- Path: Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (T) → 9 → 10 (T) → 11 (F) → End

---

Basic Condition Coverage

Test Case 1: All Conditions False

- Path: Start → 1 → 2 → 3 → 4 (F) → 9 → 10 (F) → End

Test Case 2: Some Conditions True, Loop Back

- Path: Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (F) → 8 → 6 → 7 (T) → 9 → 10 (T) → 11 (T) → 12 (F) → 14 → End

Test Case 3: Early Loop Exit

- Path: Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (F) → End

Test Case 4: Path with Full Condition Coverage

- Path: Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (T) → 9 → 10 (T) → 11 (F) → End

Question 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.

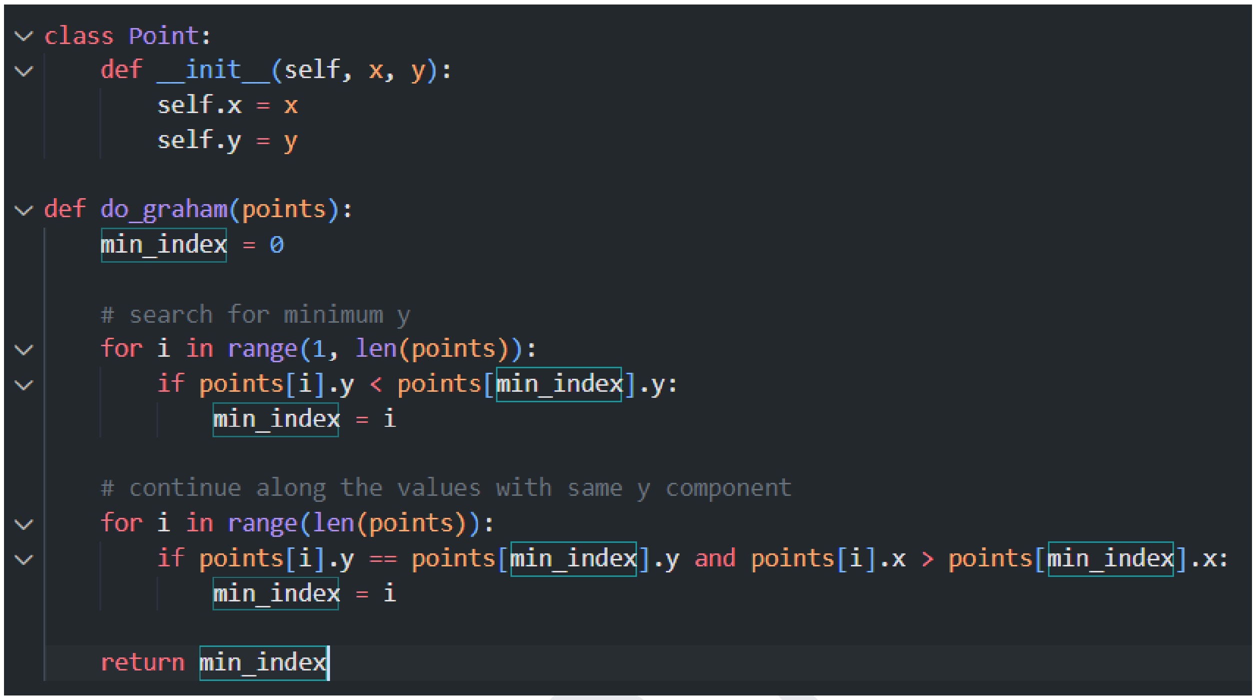
Here, you need to insert/delete/modify a piece of code that will result in failure but it is not detected

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,

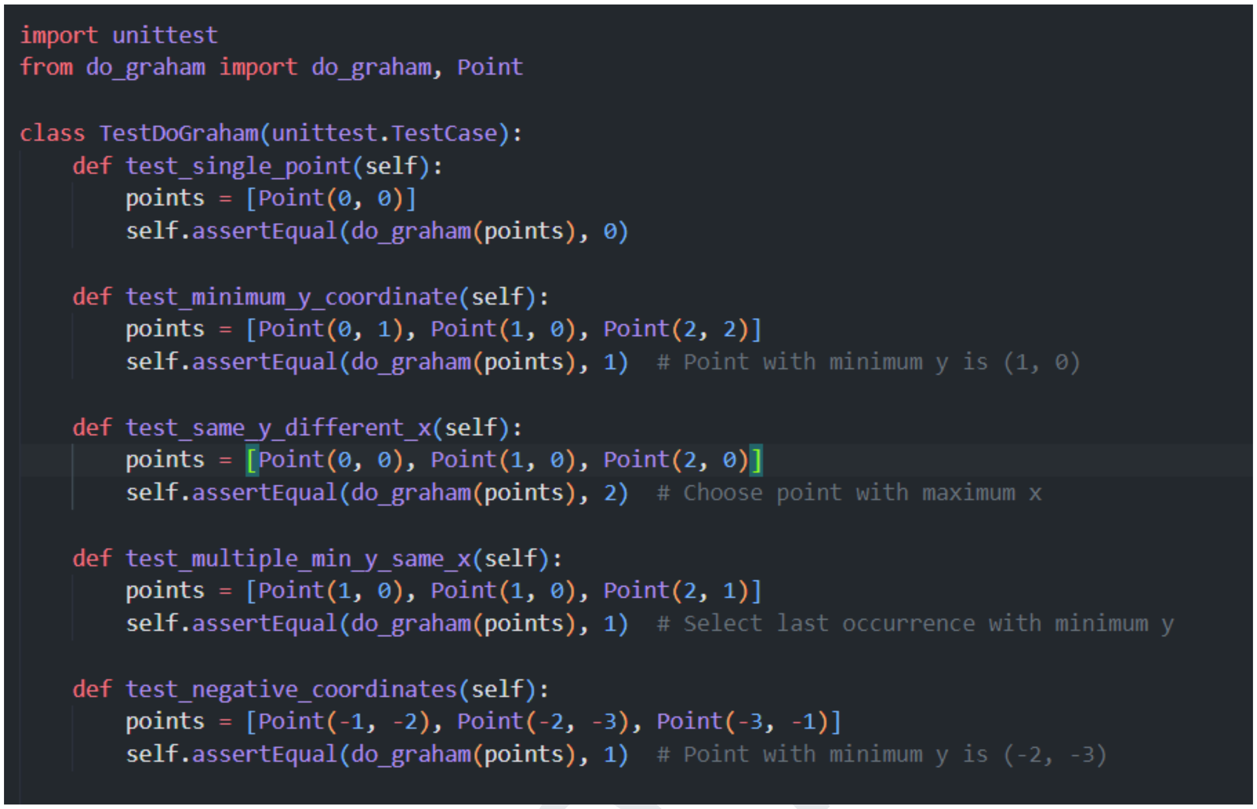
by inserting the code, by modifying the code.

Original Code:



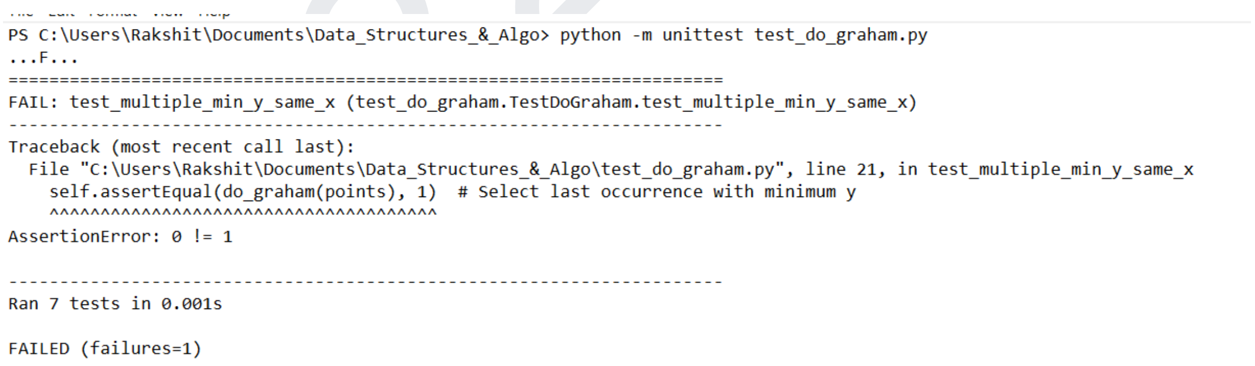
```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def do_graham(points):
    min_index = 0

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

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

    return min_index
```

Test:



```python
import unittest
from do_graham import do_graham, Point

class TestDoGraham(unittest.TestCase):
    def test_single_point(self):
        points = [Point(0, 0)]
        self.assertEqual(do_graham(points), 0)

    def test_minimum_y_coordinate(self):
        points = [Point(0, 1), Point(1, 0), Point(2, 2)]
        self.assertEqual(do_graham(points), 1)  # Point with minimum y is (1, 0)

    def test_same_y_different_x(self):
        points = [Point(0, 0), Point(1, 0), Point(2, 0)]
        self.assertEqual(do_graham(points), 2)  # Choose point with maximum x

    def test_multiple_min_y_same_x(self):
        points = [Point(1, 0), Point(1, 0), Point(2, 1)]
        self.assertEqual(do_graham(points), 1)  # Select last occurrence with minimum y

    def test_negative_coordinates(self):
        points = [Point(-1, -2), Point(-2, -3), Point(-3, -1)]
        self.assertEqual(do_graham(points), 1)  # Point with minimum y is (-2, -3)
```

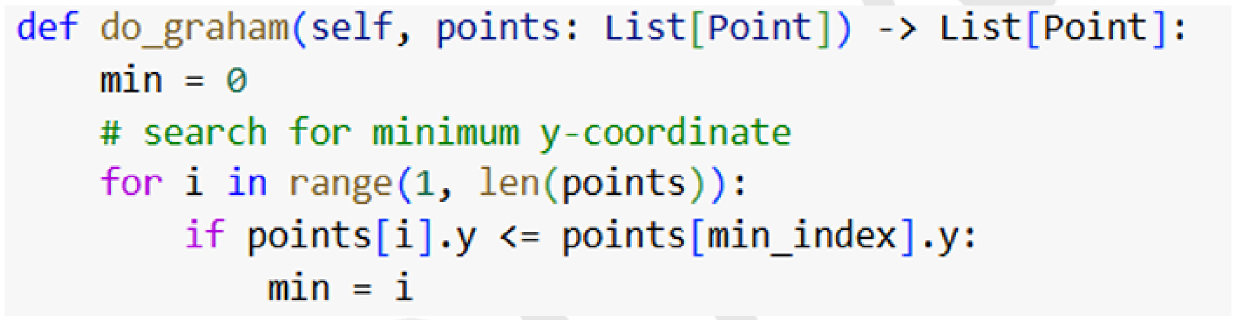
Mutation:



```
PS C:\Users\Rakshit\Documents\Data_Structures_&_Algo> python -m unittest test_do_graham.py
...F...
======================================================================
FAIL: test_multiple_min_y_same_x (test_do_graham.TestDoGraham.test_multiple_min_y_same_x)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Rakshit\Documents\Data_Structures_&_Algo\test_do_graham.py", line 21, in test_multiple_min_y_same_x
    self.assertEqual(do_graham(points), 1)  # Select last occurrence with minimum y
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 0 != 1

----------------------------------------------------------------------
Ran 7 tests in 0.001s

FAILED (failures=1)
```

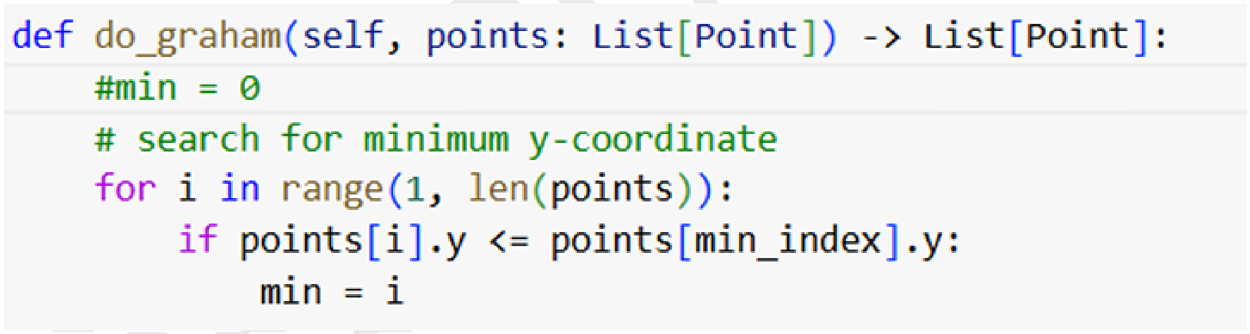
Changes which can be done are like changing the comparison sign, insert redundant checks, deleting some line and checking the output,etc.

Deletion Mutation:
Remove the line min = 0;
Before:



```
def do_graham(self, points: List[Point]) -> List[Point]:
    min = 0
    # search for minimum y-coordinate
    for i in range(1, len(points)):
        if points[i].y <= points[min_index].y:
            min = i
```

After:



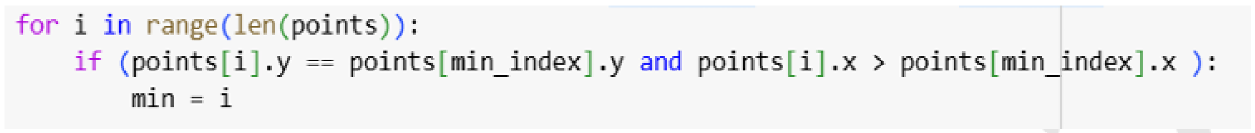
```
def do_graham(self, points: List[Point]) -> List[Point]:
    #min = 0
    # search for minimum y-coordinate
    for i in range(1, len(points)):
        if points[i].y <= points[min_index].y:
            min = i
```

Result: Without initializing min to 0, the code will try to access p.get(min) without a valid starting index, potentially causing a runtime error (e.g., NullPointerException or ArrayIndexOutOfBoundsException) if min is uninitialized or holds an invalid value. However, this error may not be caught by the minimal test set if all test cases contain multiple points, as min might be set during the first iteration of the first loop.

Test Case to Detect Deletion Mutation:
• Input: p = [(0, 0)]

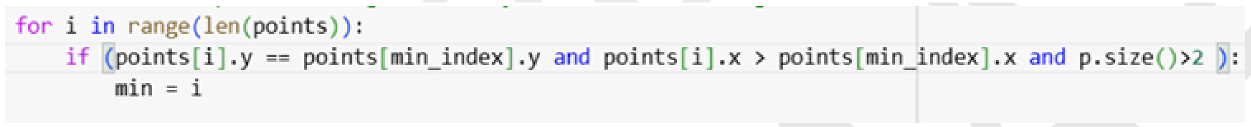
Insertion Mutation: Insert a condition p.size() > 2 in the second loop so that it only executes if there are more than two points in p:

Before:



```
for i in range(len(points)):
    if (points[i].y == points[min_index].y and points[i].x > points[min_index].x ):
        min = i
```

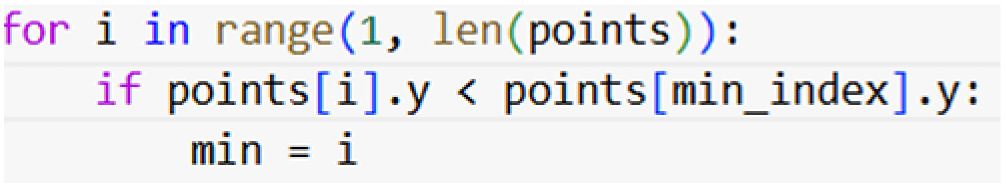
After:



```
for i in range(len(points)):
    if (points[i].y == points[min_index].y and points[i].x > points[min_index].x and p.size()>2 ):
        min = i
```

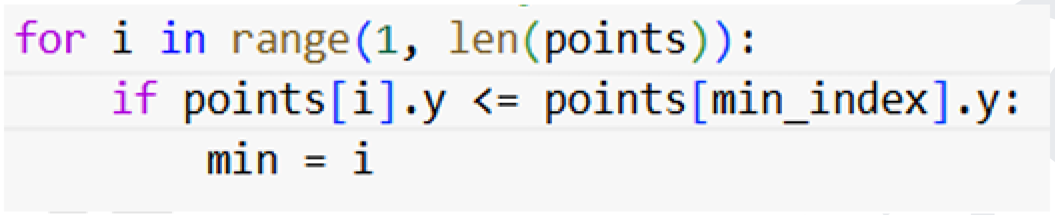
Result: Adding this condition will make the second loop ineffective if there are fewer than three points. This could cause the method to return an incorrect min value if there are only two points with the same y-coordinate but different x-coordinates. Test Case to Detect Insertion Mutation: • Input: p = [(1, 1), (2, 1)]

Modification Mutation: Change the < operator to <= in the first loop: Before:



```
for i in range(1, len(points)):
    if points[i].y < points[min_index].y:
        min = i
```

After:



```
for i in range(1, len(points)):
    if points[i].y <= points[min_index].y:
        min = i
```

Result: This modification would alter the logic for selecting the minimum y-coordinate. Instead of keeping the first encountered point with the minimum y-value, it would keep the last one, potentially affecting the result if there are multiple points with the same minimum y-value.

Test Case to Detect Modification Mutation: Input: p = [(1, 1), (3, 1), (2, 2)] Expected behavior: The point (1, 1) should be chosen as the minimum, not (3, 1).

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

For the **doGraham** function, let's outline each possible path from start to finish. The paths will be based on the following major conditions:

1. i < p.size() check.
2. p.get(i).y == p.get(min).y check.
3. p.get(i).x > p.get(min).x check.

Paths in doGraham Function Flow

1. **Path 1:** Start → min=0 → i=1 → i < p.size() (False) → End Function.
   - **Description:** No points beyond the first; loop doesn't start.
2. **Path 2:** Start → min=0 → i=1 → i < p.size() (True) → p.get(i).y == p.get(min).y (False) → i++ → i < p.size() (False) → End Function.
   - **Description:** Two points, with different y coordinates.
3. **Path 3:** Start → min=0 → i=1 → i < p.size() (True) → p.get(i).y == p.get(min).y (True) → p.get(i).x > p.get(min).x (False) → i++ → i < p.size() (False) → End Function.
   - **Description:** Two points with the same y coordinate but the second point has a smaller x.

4. **Path 4:** Start → min=0 → i=1 → i < p.size() (True) → p.get(i).y == p.get(min).y (True) → p.get(i).x > p.get(min).x (True) → min = i → i++ → i < p.size() (False) → End Function.
    - **Description:** Two points with the same y coordinate, and the second point has a larger x.
5. **Path 5:** Start → min=0 → i=1 → i < p.size() (True) → p.get(i).y != p.get(min).y (False) → i++ → i < p.size() (True) → ... (continue the process for multiple points) until i < p.size() (False) → End Function.
    - **Description:** Multiple points with the last point having the minimum y and smaller x value.

Test Cases for doGraham Function Flow Paths

Here are test cases corresponding to each path:

- **Test Case for Path 1:**
    - **Input:** points = [Point(0, 0)]
    - **Expected Result:** min_index = 0
- **Test Case for Path 2:**
    - **Input:** points = [Point(0, 0), Point(1, 2)]
    - **Expected Result:** min_index = 0

- **Test Case for Path 3:**
  - **Input:** points = [Point(2, 0), Point(1, 0)]
  - **Expected Result:** min_index = 1
- **Test Case for Path 4:**
  - **Input:** points = [Point(0, 0), Point(2, 0)]
  - **Expected Result:** min_index = 1
- **Test Case for Path 5:**
  - **Input:** points = [Point(0, 2), Point(1, 1), Point(2, 0)]
  - **Expected Result:** min_index = 2

Paths in Second Flow Graph

1. **Path 1:** Start → 1 → 2 → 3 → 4 (False) → 9 → 10 (False) → End
2. **Path 2:** Start → 1 → 2 → 3 → 4 (True) → 5 → 6 (False) → End
3. **Path 3:** Start → 1 → 2 → 3 → 4 (True) → 5 → 6 (True) → 7 (False) → 8 → 6 → 7 (True) → 9 → 10 (False) → End
4. **Path 4:** Start → 1 → 2 → 3 → 4 (True) → 5 → 6 (True) → 7 (True) → 9 → 10 (True) → 11 (False) → End

5. **Path 5:** Start → 1 → 2 → 3 → 4 (True) → 5 → 6 (True) → 7 (True) → 9 → 10 (True) → 11 (True) → 12 (False) → 14 → End
6. **Path 6:** Start → 1 → 2 → 3 → 4 (True) → 5 → 6 (True) → 7 (True) → 9 → 10 (True) → 11 (True) → 12 (True) → 13 → End

---

Test Cases for Second Flow Graph Paths

- **Test Case for Path 1:**
  - **Conditions:** No branching at node 4, end directly after node 10.
  - **Path Covered:** Start → 1 → 2 → 3 → 4 (F) → 9 → 10 (F) → End
- **Test Case for Path 2:**
  - **Conditions:** True at node 4, end directly after node 6.
  - **Path Covered:** Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (F) → End
- **Test Case for Path 3:**
  - **Conditions:** Loop through 6 to 8, followed by exiting at node 10.
  - **Path Covered:** Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (F) → 8 → 6 → 7 (T) → 9 → 10 (F) → End

- **Test Case for Path 4:**
    - **Conditions:** Exit after node 11 on a False condition.
    - **Path Covered:** Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (T) → 9 → 10 (T) → 11 (F) → End
- **Test Case for Path 5:**
    - **Conditions:** Branches through nodes 12 to 14, ending after 14.
    - **Path Covered:** Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (T) → 9 → 10 (T) → 11 (T) → 12 (F) → 14 → End
- **Test Case for Path 6:**
    - **Conditions:** All conditions true, complete traversal through node 13.
    - **Path Covered:** Start → 1 → 2 → 3 → 4 (T) → 5 → 6 (T) → 7 (T) → 9 → 10 (T) → 11 (T) → 12 (T) → 13 → End