SOFTWARE ENGINEERING

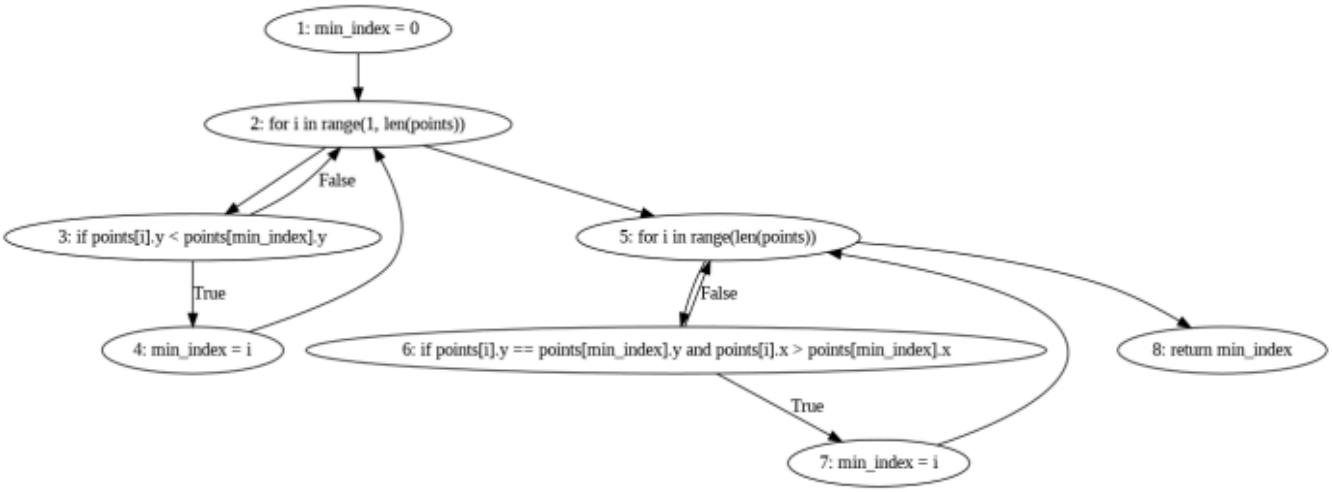
(IT-314)

Lab - 9

Khushi Vora - 202201332

Q1: 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:

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

def do_graham(points): min_index = 0


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

  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
```



Q2: 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:

Test Cases for Statement Coverage
We want to ensure every statement in the function is executed at least once.

1. Test Case 1: p = [{x: 1, y: 2}, {x: 2, y: 3}]
Explanation: This test case ensures the first for loop is executed since p has more than one point.

2. Test Case 2: p = [{x: 1, y: 2}, {x: 2, y: 2}, {x: 3, y: 2}]

Explanation: This test case ensures the second for loop is executed, as there are points with the same y-coordinate but different x-coordinates.

Test Cases for Branch Coverage
We need to create tests that cover both outcomes of each condition.
1. Test Case 1: p = [{x: 0, y: 0}, {x: 1, y: -1}, {x: 2, y: 1}]
Explanation: Ensures ((Point) p.get(i)).y < ((Point) p.get(min)).y is true.
2. Test Case 2: p = [{x: 0, y: 0}, {x: 1, y: 2}, {x: 2, y: 3}]
Explanation: Ensures ((Point) p.get(i)).y < ((Point) p.get(min)).y is false.
3. Test Case 3: p = [{x: 1, y: 2}, {x: 2, y: 2}, {x: 3, y: 2}]
Explanation: Ensures ((Point) p.get(i)).y == ((Point) p.get(min)).y is true, and ((Point) p.get(i)).x > ((Point) p.get(min)).x is also true.
4. Test Case 4: p = [{x: 1, y: 2}, {x: 2, y: 3}, {x: 3, y: 4}]
Explanation: Ensures ((Point) p.get(i)).y == ((Point) p.get(min)).y is false

Test Cases for Basic Condition Coverage
We must ensure that every condition is tested for both true and false outcomes.
1. Test Case 1: p = [{x: 0, y: 0}, {x: -1, y: -1}, {x: 1, y: 1}]
Condition ((Point) p.get(i)).y < ((Point) p.get(min)).y) is both true and false in different iterations.
2. Test Case 2: p = [{x: 0, y: 0}, {x: 1, y: 0}, {x: 2, y: 0}]
Condition ((Point) p.get(i)).y == ((Point) p.get(min)).y) is true, and ((Point) p.get(i)).x > ((Point) p.get(min)).x) is also tested as true and false.
3. Test Case 3: p = [{x: 0, y: 0}, {x: -1, y: 0}, {x: 1, y: 0}]
Condition ((Point) p.get(i)).x > ((Point) p.get(min)).x) is both true and false.

Q3: 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:

```
[*] Start mutation process:
   - targets: point
   - tests: test_points
[*] 4 tests passed:
   - test_points [0.36220 s]
[*] Start mutants generation and execution:
   - [#   1] COI point:
--------------------------------------------------------------------------------
   6:
   7: def find_min_point(points):
   8:     min_index = 0
   9:     for i in range(1, len(points)):
- 10:         if points[i].y < points[min_index].y:
+ 10:         if not (points[i].y < points[min_index].y):
  11:             min_index = i
  12:     for i in range(len(points)):
  13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
  14:             min_index = i
--------------------------------------------------------------------------------
[0.23355 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_ties
   - [#   2] COI point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
--------------------------------------------------------------------------------
[0.23355 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_ties
   - [#   2] COI point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if not ((points[i].y == points[min_index].y and points[i].x > points[min_index].x))
  14:             min_index = i
  15:     return points[min_index]
--------------------------------------------------------------------------------
[0.27441 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_same_y
   - [#   3] LCR point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y or points[i].x > points[min_index].x):
  14:             min_index = i
  15:     return points[min_index]
```

```
--------------------------------------------------------------------------------
[0.18323 s] survived
   - [#   6] ROR point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y != points[min_index].y and points[i].x > points[min_index].x):
  14:             min_index = i
  15:     return points[min_index]
--------------------------------------------------------------------------------
[0.18059 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_same_y
   - [#   7] ROR point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y and points[i].x < points[min_index].x):
  14:             min_index = i
  15:     return points[min_index]
--------------------------------------------------------------------------------
```

```
[0.13933 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_same_y
   - [#   8] ROR point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y and points[i].x >= points[min_index].x):
  14:             min_index = i
  15:     return points[min_index]
--------------------------------------------------------------------------------
[0.11494 s] survived
[*] Mutation score [2.22089 s]: 75.0%
   - all: 8
   - killed: 6 (75.0%)
   - survived: 2 (25.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
```

Q4: Create a test set that satisfies the path coverage criterion where every loop is explored at least zero one or two times.

Answer:

Test Case 1: Loop Explored Zero Times
● Input: An empty vector p.

Vector<Point> p = new Vector<Point>();
● Expected Result: The method should return immediately without any processing. This covers the condition where the vector size is zero, leading to the exit condition of the method.

Test Case 2: Loop Explored Once
● Input: A vector with one point.
Vector<Point> p = new Vector<Point>();
p.add(new Point(0, 0));
● Expected Result: The method should run the second loop once and handle the point. There will be no effective change to the vector since it contains only one point. This test case covers the scenario where the loop iterates once.

Test Case 3: Loop Explored Twice
● Input: A vector with two points where the first point has a higher y-coordinate than the second.

Vector<Point> p = new Vector<Point>();
p.add(new Point(1, 1));
p.add(new Point(0, 0));
● Expected Result: The method should enter the first loop and compare the two points, finding that the second point has a lower y-coordinate. Consequently, min should be updated to 1, and a swap should occur, moving the second point to the front of the vector.

Test Case 4: Loop Explored More Than Twice

● Input: A vector with multiple points.

Vector<Point> p = new Vector<Point>();
p.add(new Point(2, 2));
p.add(new Point(1, 0));
p.add(new Point(0, 3));

● Expected Result: The method should iterate through the vector, comparing the points. It should identify the point with the lowest y-coordinate and make the necessary swaps to bring it to the front. This case ensures that the loops are executed multiple times, covering more complex scenarios.