IT-314 LAB-9

MUTATION TESTING

Maharshi Raval **202201003**

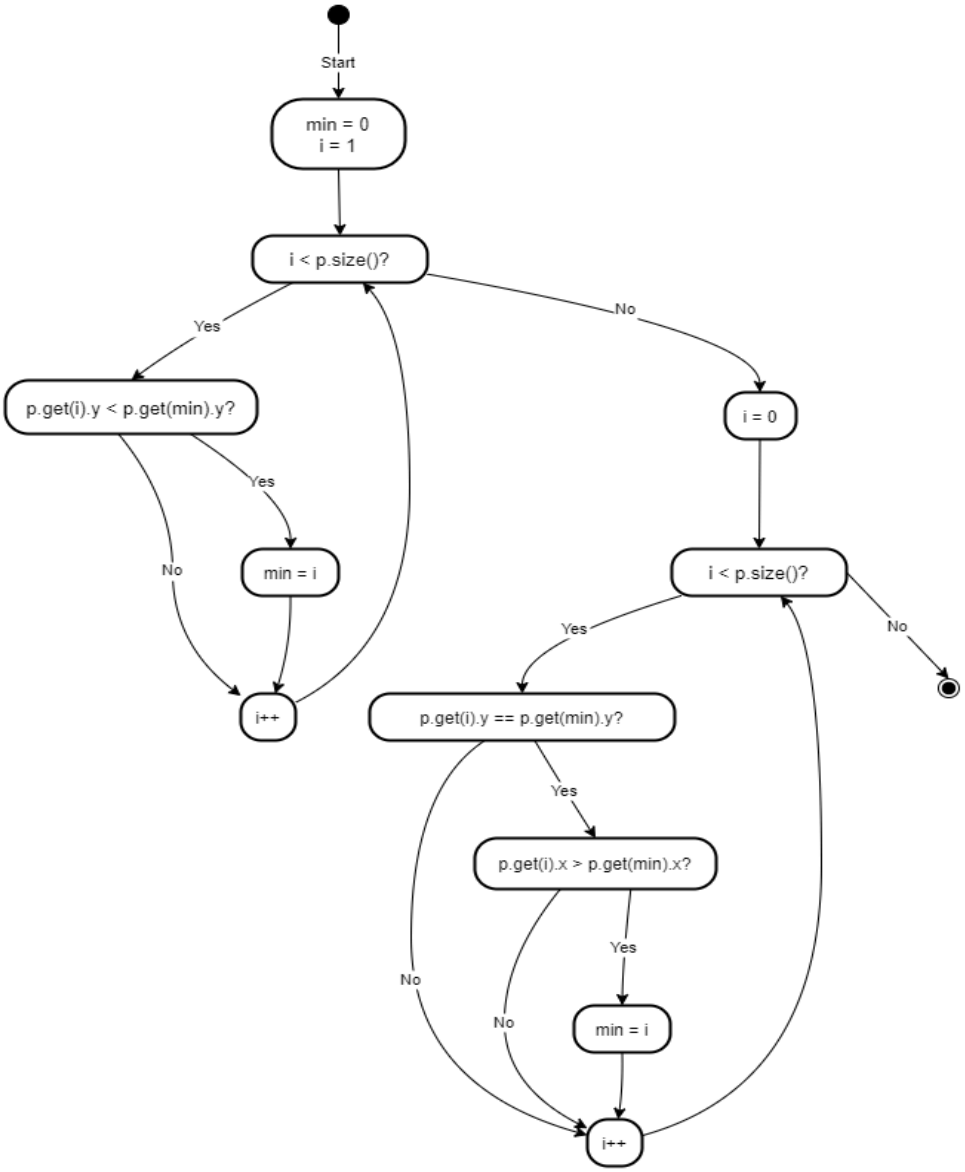
Code snippet under consideration:

```
Vector doGraham(Vector p) {
    int i, j, min, M;
    Point t;
    min = 0;

    // Search for minimum:
    for (i = 1; i < p.size(); ++i) {
        if (((Point) p.get(i)).y < ((Point) p.get(min)).y) {
            min = i;
        }
    }

    // Continue along the values with the same y component
    for (i = 0; i < p.size(); ++i) {
        if ((((Point) p.get(i)).y == ((Point) p.get(min)).y) &&
            (((Point) p.get(i)).x > ((Point) p.get(min)).x)) {
            min = i;
        }
    }
}
```

Task 1: Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG).



Control Flow Graph Factory Tool : No

Eclipse flow graph generator : No

Task 2: Construct test sets for your flow graph that are adequate for the following criteria:

a. **Statement Coverage:**

TC1: p = [(1, 7), (2, 3), (8, 1)]

- Covers cases where `p.get(i).y < p.get(min).y` is true.

TC2: p = [(1, 1), (2, 1), (3,1), (4,1)]

- covers cases where `p.get(i).y == p.get(min).y` and `p.get(i).x > p.get(min).x`

b. **Branch Coverage:**

TC3: p = [(1, 2), (3, 3), (5, 9), (10, 15)]

- covers the false branch for condition `p.get(i).y < p.get(min).y`

TC3: p = [(10, 15), (5, 9), (3, 3), (1, 2)]

- covers the true branch for condition `p.get(i).y < p.get(min).y`

TC4: p = [(1, 1), (2, 1), (3, 1)]

- cover both true and false branches of `p.get(i).y == p.get(min).y` and `p.get(i).x > p.get(min).x`.

c. **Basic Condition Coverage:**

TC5: p = [(1, 2), (3, 3)]

- Covers p.get(i).y < p.get(min).y as false.

TC6: p = [(1, 5), (1, 3)]

- Covers p.get(i).y < p.get(min).y as true.

TC7**:** p = [(1, 1), (2, 1)]

- Covers p.get(i).y == p.get(min).y as true and p.get(i).x > p.get(min).x as true.

TC8: p = [(2, 2), (1, 1)]

- Covers p.get(i).y == p.get(min).y as false and p.get(i).x > p.get(min).x as false.

Task 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.

```
[*] 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%)
```

1. Deletion Mutation: Remove `min = i` in the first `if` condition.

Mutation Code:

```
// Remove this line: min = i; if (((Point)
p.get(i)).y < ((Point) p.get(min)).y) {
    // min = i; <- this line is removed

}
```

2. Insertion Mutation: Add `min = 0` at the beginning of the second loop

Mutation Code:

```
   for(i = 0; i < p.size(); ++i) { min = 0; //
      Inserted line if (((Point) p.get(i)).y ==
      ((Point) p.get(min)).y &&
       ((Point) p.get(i)).x < ((Point) p.get(min)).x) {
       min = i;

   }

}
```

3. **Modification Mutation: Change `p[i].y < p[min].y` to `p[i].y > p[min].y` in the first `if` condition.**

Mutation Code:

```
   if (((Point) p.get(i)).y > ((Point) p.get(min)).y) { // Modified from
      < to > min = i;
}
```

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

```
import unittest
from point import Point, find_min_point
class TestFindMinPointPathCoverage(unittest.TestCase):

    def test_no_points(self):
        points = []
        with self.assertRaises(IndexError): #Expect an IndexError due to emptylist
            find_min_point(points)
    def test_single_point(self):
        points = [Point(0, 0)]
        result = find_min_point(points)
        self.assertEqual(result, points[0]) # Expect the point (0, 0)
    def test_two_points_unique_min(self):
        points = [Point(1, 2), Point(2, 3)]
        result = find_min_point(points)
        self.assertEqual(result, points[0]) # Expect the point (1, 2)


    def test_multiple_points_unique_min(self):
        points = [Point(1, 4), Point(2, 3), Point(0, 1)]
        result = find_min_point(points)
        self.assertEqual(result, points[2]) # Expect the point (0, 1)
    def test_multiple_points_same_y(self):
```

```
        points = [Point(1, 2), Point(3, 2), Point(2, 2)]
        result = find_min_point(points)
        self.assertEqual(result, points[1]) # Expect the point (3, 2)
    def test_multiple_points_minimum_y_ties(self):
        points = [Point(1, 2), Point(2, 2), Point(3, 1), Point(4, 1)]
        result = find_min_point(points)
        self.assertEqual(result, points[3]) # Expect the point (4, 1)
# Run the tests if this file is executed
if __name__ == "__main__":
        unittest.main()
```

Mutation Testing using mut.py tool:

```
----------------------------------------------------------------------------------
[0.12519 s] survived
[*] Mutation score [1.53947 s]: 75.0%
   - all: 8
   - killed: 6 (75.0%)
   - survived: 2 (25.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
```

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).

Yes