IT313: Software Engineering

Lab 9

202201242

Dev Davda

Given code:

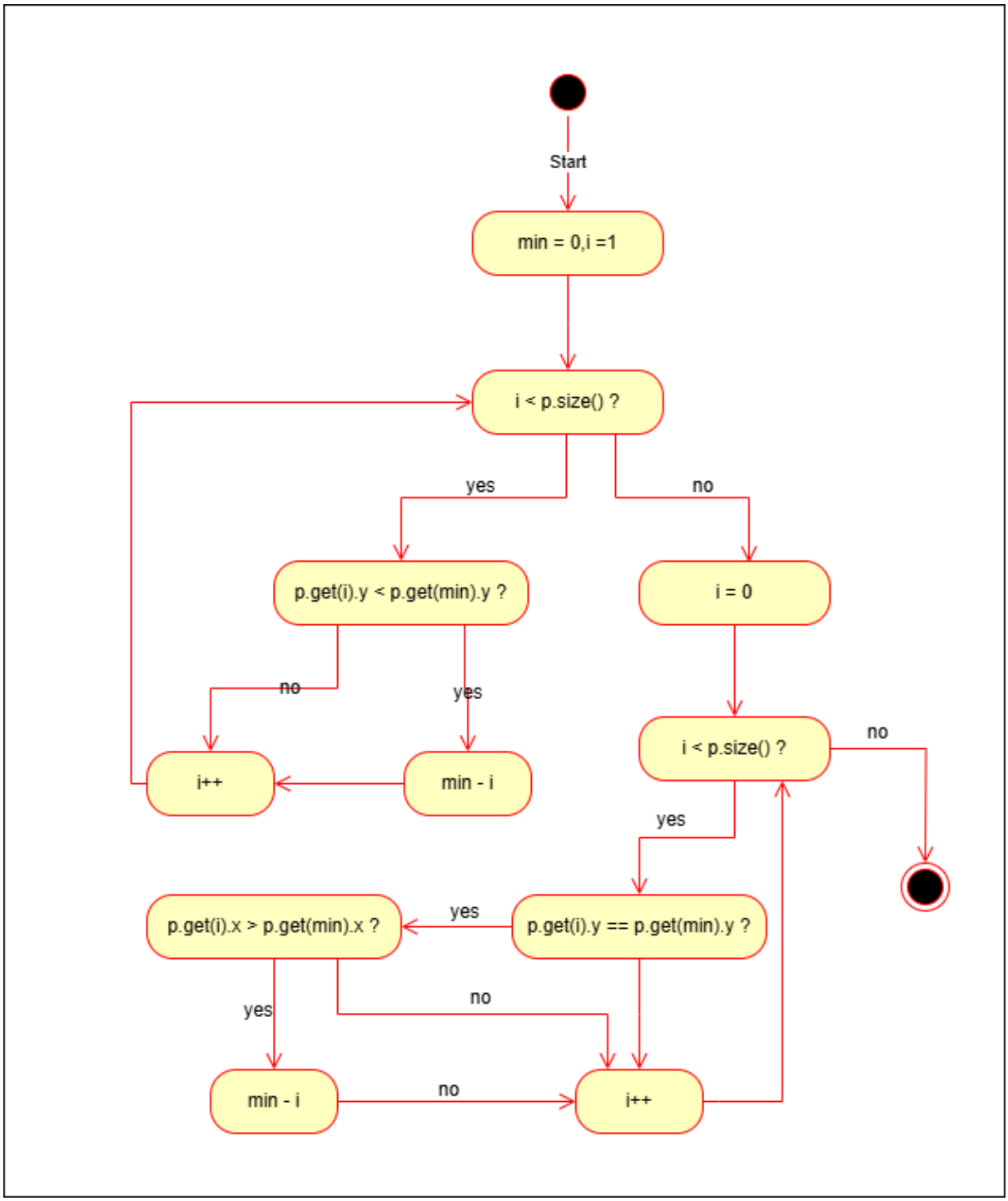
```
Vector doGraham(Vector p) {
        int i,j,min,M;

        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 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;
            }
        }
```

1] Control Flow Graph (CFG)



2. Construct Test Sets for Coverage Criteria

a. Statement Coverage

- **Test Case 1:** p has at least one point with a unique minimum y value and another point with a higher y value.
    - Example: p = [(1, 2), (3, 4)]

b. Branch Coverage

- **Test Case 1:** p has points with different y values to cover y(i) < y(min).
  - Example: p = [(1, 3), (2, 2)]
- **Test Case 2:** p has points with the same y values, where some x values are larger to cover y(i) == y(min) && x(i) > x(min).
  - Example: p = [(1, 3), (3, 3)]

c. Basic Condition Coverage

- **Test Case 1:** p has at least one point with a lower y value and another point with a higher y value.
  - Example: p = [(1, 2), (3, 4)]
- **Test Case 2:** p has points with the same y value and different x values, where the second point has a larger x.
  - Example: p = [(1, 2), (3, 2)]
- **Test Case 3:** p has points with the same y value and identical x values to check the false branch of x(i) > x(min).
  - Example: p = [(2, 2), (2, 2)]

3. Mutation Testing

Possible Mutation:

1] Change the Comparison Operator for x(i) > x(min)

Original:

```
if (((Point) p.get(i)).y == ((Point) p.get(min)).y && ((Point) p.get(i)).x > ((Point) p.get(min)).x)
```

Mutated:

```
if (((Point) p.get(i)).y == ((Point) p.get(min)).y && ((Point) p.get(i)).x >= ((Point) p.get(min)).x)
```

This mutation changes the > comparison to >=. With this mutation, the function may still pass the tests where there are no duplicate points with the same x and y values but it could fail in cases where there are identical points. This would not be detected by our current test set, as we have no cases where two points with the same x and y values are tested.

- **Test Case to Detect the Mutation:** [(1, 2), (1, 2)]

2] Remove the Update of min = i;

removing the line min = i; in either the first or the second loop.

Original:

min = i;

Mutated (removed min = i;):

// min = i; (removed)

If we remove this line from either the first or second loop, the code will no longer update the min variable as intended. However, this mutation might pass most of our current tests since they only check specific scenarios and do not involve extensive variations in y and x values across multiple points.

- **Test Case to Detect the Mutation:** p = [(5, 5), (1, 2), (3, 4)].

3] Change the Comparison y(i) < y(min) to y(i) > y(min)

This mutation would change the code to incorrectly identify the maximum y point instead of the minimum.

Original:

if (((Point) p.get(i)).y < ((Point) p.get(min)).y)

Mutated:

if (((Point) p.get(i)).y > ((Point) p.get(min)).y)

This mutation would cause the code to fail, as it would choose the point with the maximum y value rather than the minimum. However, this error might go undetected in cases where we only have points with increasing y values or similar conditions.

- **Test Case to Detect the Mutation:** p = [(1, 1), (3, 2), (2, 3)].

4. Test Set for Path Coverage

For **path coverage**, we need to create a set of tests where each loop is executed zero, one, and two times.

Here is the python code:

```
import unittest

from point import Point, find_min_point


class TestFindMinPointPathCoverage(unittest.TestCase):
```

```
    def test_no_points(self):
        # Zero iterations: Test with an empty list (loops are not
entered)
        points = []
        with self.assertRaises(IndexError):  # Expect an IndexError
due to empty list
            find_min_point(points)


    def test_single_point(self):
        # One iteration: Test with a single point (each loop runs
exactly once)
        points = [Point(0, 0)]
        result = find_min_point(points)
        self.assertEqual(result, points[0])  # Expect the point (0,
0)


    def test_two_points_different_y(self):
        # Two iterations: Test with two points having different y
values
        points = [Point(1, 2), Point(2, 3)]
        result = find_min_point(points)
        self.assertEqual(result, points[0])  # Expect the point (1,
2)


    def test_three_points_unique_min_y(self):
        # Two iterations: Test with three points having a unique
minimum y value
        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):
        # Two iterations with the same y value and varying x values
        points = [Point(1, 2), Point(3, 2), Point(2, 2)]
```

```
        result = find_min_point(points)
        self.assertEqual(result, points[1])  # Expect the rightmost
point (3, 2)


    def test_multiple_points_minimum_y_ties(self):
        # Two iterations with minimum y ties, choosing the rightmost
among them
        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 rightmost
minimum point (4, 1)


if __name__ == "__main__":
    unittest.main()
```