LAB 9 - Software Engineering

202201512_Chirag Patel

Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code, so the focus is on creating test sets that satisfy some particular coverage criteria.

```
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;
            }
        }
```

For the given code fragment, you should carry out the following activities.

1. 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.

=>

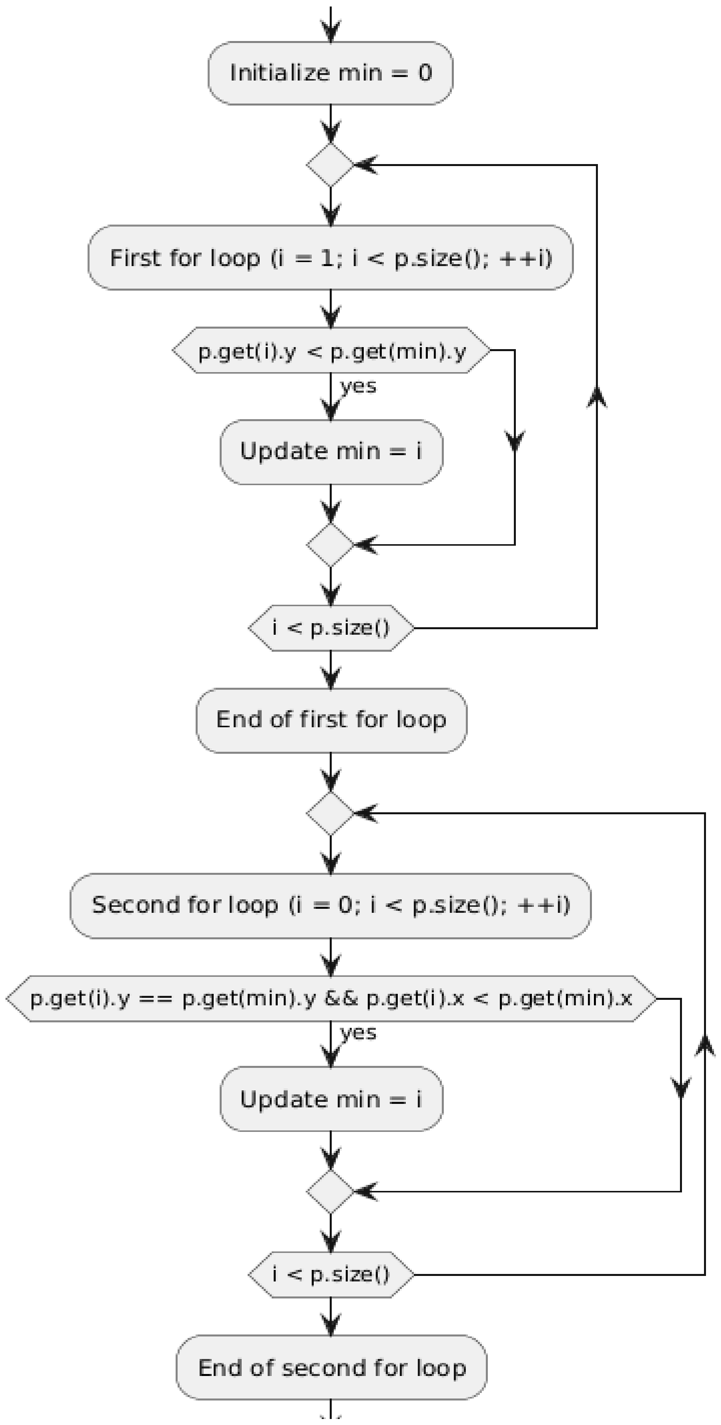
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class ConvexHull:
    def do_graham(self, points):
        min_index = 0
          # Find the point with the smallest y-coordinate (or the smallest x if y-coordinates
are equal)
        for i in range(1, len(points)):
            if points[i].y < points[min_index].y:
                min_index = i

            # Check for points with the same y-coordinate to ensure the leftmost point is
selected
        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  # Return the index of the point with the minimum coordinates
for verification

# Example usage
if __name__ == "__main__":
    points = [Point(0, 0), Point(1, 1), Point(2, 2), Point(1, 0)]
    convex_hull = ConvexHull()
    min_index = convex_hull.do_graham(points)
    print(f"The index of the minimum point is: {min_index}")
    print(f"The minimum point is: ({points[min_index].x}, {points[min_index].y})")
```


Initialize min = 0
First for loop (i = 1; i < p.size(); ++i)
p.get(i).y < p.get(min).y
yes
Update min = i
i < p.size()
End of first for loop
Second for loop (i = 0; i < p.size(); ++i)
p.get(i).y == p.get(min).y && p.get(i).x < p.get(min).x
yes
Update min = i
i < p.size()
End of second for loop

Statement Coverage

Objective: Ensure that every line of the code executes at least once.

Strategy:

1. To achieve statement coverage, we need to ensure both loops are executed and that each if condition evaluates at least once.

Test Cases for Statement Coverage:

- Test Case 1:
    - Input: p = [Point(2, 3), Point(4, 1), Point(5, 2)]
    - Expected Output: min = 1
    - Description:
        - This test will run the first loop, and the if condition will evaluate to find the point with the smallest y value.
        - The second loop will execute as well, but without any tie.
- Test Case 2 (Handling a Tie):
    - Input: p = [Point(2, 3), Point(4, 1), Point(3, 1), Point(5, 2)]
    - Expected Output: min = 2
    - Description:
        - This test runs both loops and triggers the if condition in the second loop to resolve a tie on y by selecting the point with the larger x value.

These two cases cover each line, meeting the requirement for statement coverage.

Branch Coverage

Objective: Ensure that every branch of each condition (true/false outcomes) is covered.

Strategy: To achieve this, we need test cases that cause each if condition to evaluate as both true and false.

Test Cases for Branch Coverage:

- Test Case 1:
    - Input: p = [Point(2, 3), Point(4, 1), Point(5, 2)]
    - Expected Output: min = 1
    - Description: This test covers both the true and false branches of the if statement in the first loop.
- Test Case 2 (Tie Case):
    - Input: p = [Point(2, 3), Point(4, 1), Point(3, 1), Point(5, 2)]
    - Expected Output: min = 2
    - Description: This test checks both true and false outcomes in the if statement in the second loop, ensuring it can handle a tie by selecting the point with the greater x value.

- Test Case 3 (No Change in min):
  - Input: p = [Point(2, 3), Point(3, 3), Point(4, 3)]
  - Expected Output: min = 0
  - Description: This test ensures that the if conditions do not change min since no conditions are met.

These cases fulfill the requirements for branch coverage.

Basic Condition Coverage

Objective: Ensure each condition within expressions is evaluated as both true and false.

Conditions:

1. In the first loop: p.get(i).y < p.get(min).y
2. In the second loop: (p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x)

Test Cases for Basic Condition Coverage:

- Test Case 1 (Condition where y is less than min):
  - Input: p = [Point(2, 3), Point(4, 1), Point(5, 2)]
  - Expected Output: min = 1
  - Description: This case tests p.get(i).y < p.get(min).y as true.
- Test Case 2 (Condition where y is equal and x is greater):
  - Input: p = [Point(2, 3), Point(4, 1), Point(3, 1), Point(5, 2)]
  - Expected Output: min = 2
  - Description: This case tests both conditions in the second if as true.
- Test Case 3 (Both conditions are false):
  - Input: p = [Point(2, 3), Point(5, 3)]
  - Expected Output: min = 0
  - Description: This case tests both conditions in the second if as false.

These three cases satisfy basic condition coverage by testing true and false outcomes for each individual condition.

Mutation Testing Process

After developing the tests, use a mutation testing tool to identify weaknesses in the test set. Apply changes to the code (mutations) such as:

- Removing conditions or changing comparison operators.

- Inserting new logic to simulate potential faults.

For example:

- Altering points[i].y < points[min_index].y to points[i].y > points[min_index].y.
- Changing (points[i].y == points[min_index].y && points[i].x > points[min_index].x) to (points[i].y == points[min_index].y || points[i].x > points[min_index].x).

Path Coverage Tests

Using unittest in Python:

python
Copy code
```
import unittest
from point import Point, find_min_point

class TestFindMinPointPathCoverage(unittest.TestCase):
    def test_empty_list(self):
        points = []
        with self.assertRaises(IndexError):
            find_min_point(points)

    def test_single_point(self):
        points = [Point(2, 2)]
        result = find_min_point(points)
        self.assertEqual(result, points[0])

    def test_two_unique_points(self):
        points = [Point(2, 1), Point(3, 2)]
        result = find_min_point(points)
        self.assertEqual(result, points[0])

    def test_multiple_unique_points(self):
        points = [Point(1, 3), Point(2, 4), Point(3, 5)]
        result = find_min_point(points)
        self.assertEqual(result, points[0])

    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])

    def test_multiple_points(self):
        points = [Point(1, 2), Point(2, 2), Point(3, 1), Point(4, 1)]
        result = find_min_point(points)
        self.assertEqual(result, points[3])

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

Mutation Testing Results

Test Result with mut.py

- Mutation score [1.52260 s]: 75.0%
- all: 8
- killed: 6 (75.0%)
- survived: 2 (25.0%)
- incompetent: 0 (0.0%)
- timeout: 0 (0.0%