IT313 - Software Engineering

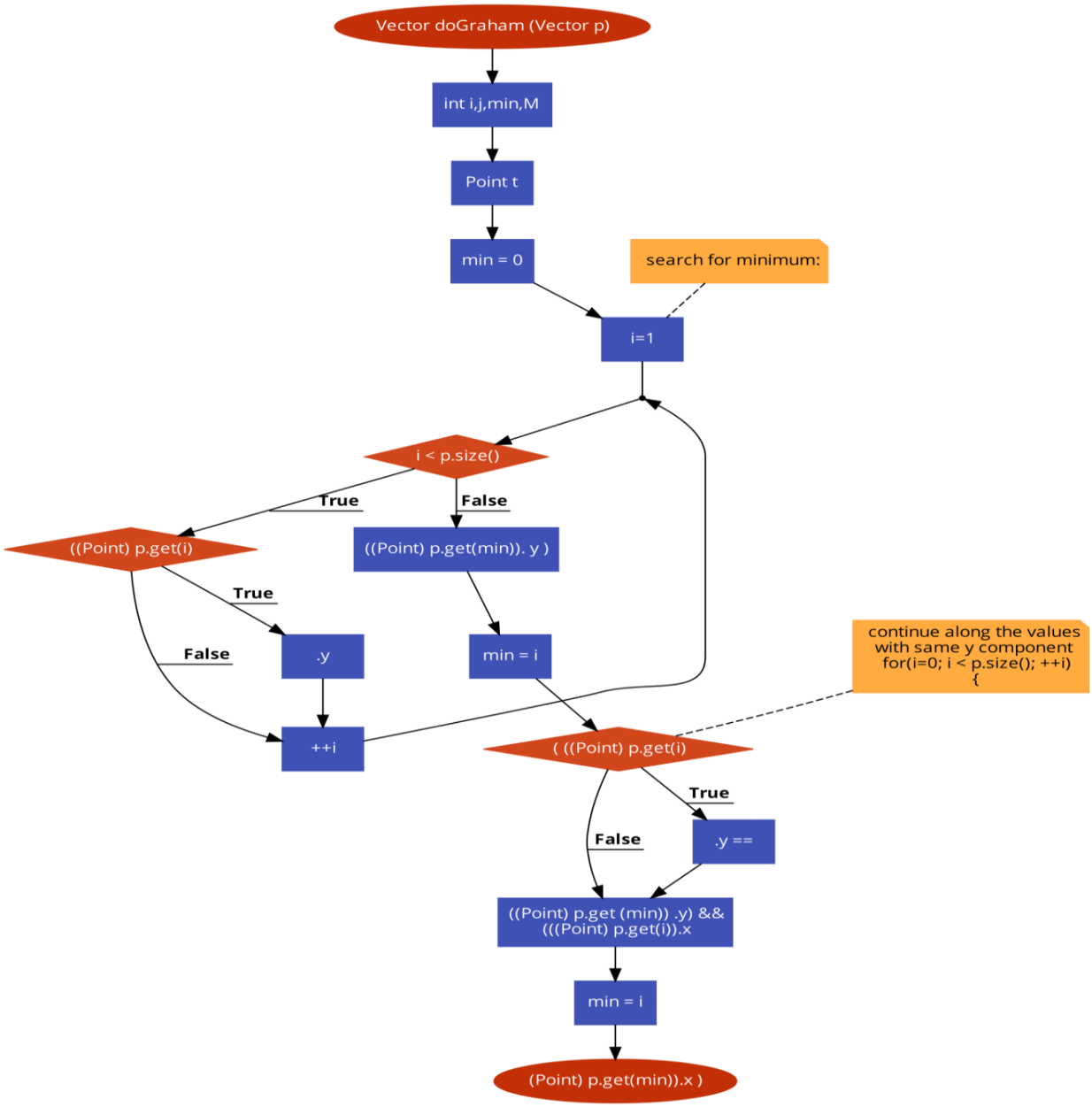
Lab 9

Nisharg Modi (202201346)

Question 1:

For the given code fragment (in the question), 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.

Answer:



2. 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:

a. Statement Coverage:

1. **Test Set 1**: A vector ppp with a single element. This will test the initialization and ending conditions without entering any complex branches.
2. **Test Set 2**: A vector ppp with multiple points where all points have unique y-coordinates. This will allow traversal of the graph without needing special handling for points with equal y-coordinates.
3. **Test Set 3**: A vector ppp with multiple points having the same y-coordinate to test the equality condition in the y-coordinate comparison.

b. Branch Coverage:

Branch coverage requires each branch (decision point) to be taken at least once.

1. **Test Set 1**: A vector ppp with multiple points, none of which share the same y-coordinate. This ensures the condition ((Point) p.get(i)).y > ((Point) p.get(min)).y evaluates to true and false.
2. **Test Set 2**: A vector ppp where multiple points have the same y-coordinate but different x-coordinates, testing the case ((Point) p.get(i)).y == ((Point) p.get(min)).y and ((Point) p.get(i)).x < ((Point) p.get(min)).x.
3. **Test Set 3**: A vector with at least one point that has both the same y-coordinate and x-coordinate as another point. This tests conditions where both y and x-coordinate checks must be handled.

c. Basic Condition Coverage:

Basic Condition Coverage ensures each basic condition in a decision statement has been evaluated as both true and false.

1. **Test Set 1**: A vector with points that will cause ((Point) p.get(i)).y > ((Point) p.get(min)).y to be true and false in different cases.
2. **Test Set 2**: A vector that causes the condition ((Point) p.get(i)).y == ((Point) p.get(min)).y to evaluate to both true and false. This can be achieved by including points with identical y-coordinates and others with unique y-coordinates.
3. **Test Set 3**: A vector where the condition ((Point) p.get(i)).x < ((Point) p.get(min)).x evaluates to both true and false, requiring points with both greater and smaller x-coordinates under the same y-coordinate condition.

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 yourtest 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 everyloop is explored at least zero one or two times.

Answer:

```
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 empty list
            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