IT314- Software Engineering

Lab 09 – Mutation Testing

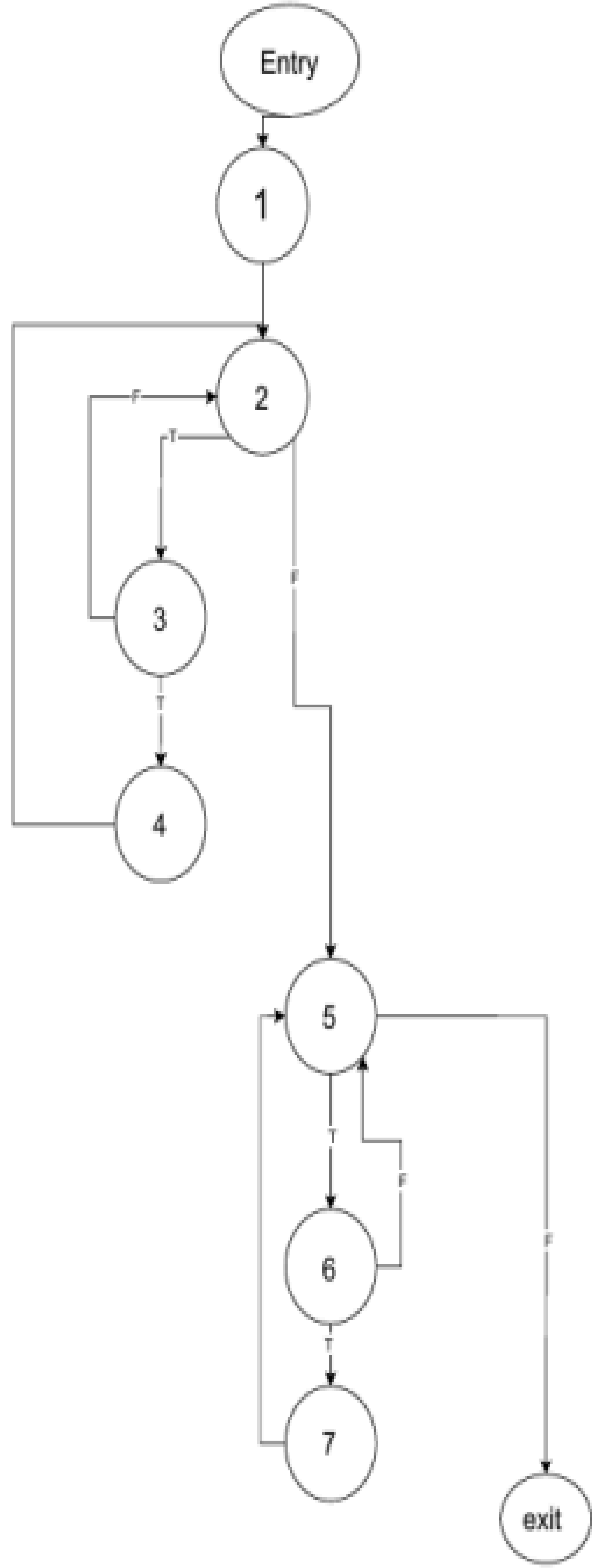
Gireesh Reddy

202201122(G2)

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

def find_min_point(points):
  (1) min_index = 0
  (2) for i in range(1, len(points)):
        (3)if points[i].y < points[min_index].y:
              (4)min_index = i
   (5) for i in range(len(points)):
        (6) if points[i].y == points[min_index].y and
points[i].x > points[min_index].x:
              (7) min_index = i
```


Entry
1
2
F
T
3
F
T
4
5
T
F
6
T
7
F
exit

1. After generating the control flow graph, check whether your CFG matches 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

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

a. Statement Coverage

Statement coverage requires that each statement in the code is executed at least once. We need to ensure that all lines (1 to 7 in the code) are executed in at least one of the test cases.

Test Set for Statement Coverage:

Test Case 1: Single Point

Input: points = [Point(0, 0)]

Expected Result: min_index = 0

Path: Entry → 1 → 2 (False) → 5 → 6 (False) → Exit

Test Case 2: Multiple Points with Unique Minimum

y-Coordinate Input: points = [Point(1, 3), Point(2, 2), Point(0, 1)]

Expected Result: min_index = 2

Path: Entry → 1 → 2 (True) → 3 (True) → 4 → 2 (False) → 5 → 6 (False) → Exit

b. Branch Coverage
Branch coverage requires that each branch in the code (True/False paths for each decision) is taken at least once.

Test Set for Branch Coverage:

Test Case 1: Single Point

Input: points = [Point(0, 0)]

Expected Result: min_index = 0

Path: Entry → 1 → 2 (False) → 5 → 6 (False) → Exit

Test Case 2: Multiple Points with Unique Minimum

y-Coordinate Input: points = [Point(0, 3), Point(1, 2), Point(2, 1)]

Expected Result: min_index = 2

Path: Entry → 1 → 2 (True) → 3 (True) → 4 → 2 (False) → 5 → 6 (False) → Exit

Test Case 3: Tied Minimum y-Coordinate with Different

x-Coordinates Input: points = [Point(0, 1), Point(2, 1), Point(1, 3)]

Expected Result: min_index = 1

Path: Entry → 1 → 2 (True) → 3 (False) → 2 (False) → 5 → 6 (True) → 7 → 5 → 6 (False) → Exit

This test set achieves branch coverage by ensuring that each branch (True/False paths for both loops and conditions) is taken.

c. Basic Condition Coverage

Basic Condition Coverage requires that each individual condition within every decision is evaluated as both True and False at least once.

Test Set for Basic Condition Coverage:

Test Case 1: Single Point (ensures points[i].y < points[min_index].y is False)

Input: points = [Point(0, 0)]

Expected Result: min_index = 0

Path: Entry → 1 → 2 (False) → 5 → 6 (False) → Exit

Test Case 2: Unique Minimum y-Coordinate (ensures points[i].y < points[min_index].y is True)

Input: points = [Point(0, 3), Point(1, 2), Point(2, 1)]

Expected Result: min_index = 2

Path: Entry → 1 → 2 (True) → 3 (True) → 4 → 2 (False) → 5 → 6 (False) → Exit

Test Case 3: Tied Minimum y-Coordinate with Larger x (ensures points[i].y == points[min_index].y is True, and points[i].x > points[min_index].x is True)

Input: points = [Point(0, 1), Point(2, 1)]

Expected Result: min_index = 1

Path: Entry → 1 → 2 (True) → 3 (False) → 2 (False) → 5 → 6 (True) → 7 → Exit

Test Case 4: Tied Minimum y-Coordinate with Smaller x (ensures points[i].y == points[min_index].y is True, and points[i].x > points[min_index].x is False)

Input: points = [Point(2, 1), Point(0, 1)]

Expected Result: min_index = 0 (point with the smallest x-coordinate is selected)

Path: Entry → 1 → 2 (True) → 3 (False) → 2 (False) → 5 → 6 (True) → Exit

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%)
```

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
      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%)
```