ENGINEERS WITH
SOCIAL RESPONSIBILITY

IT 314 SOFTWARE ENGINEERING

VRAJ GANDHI
202201425

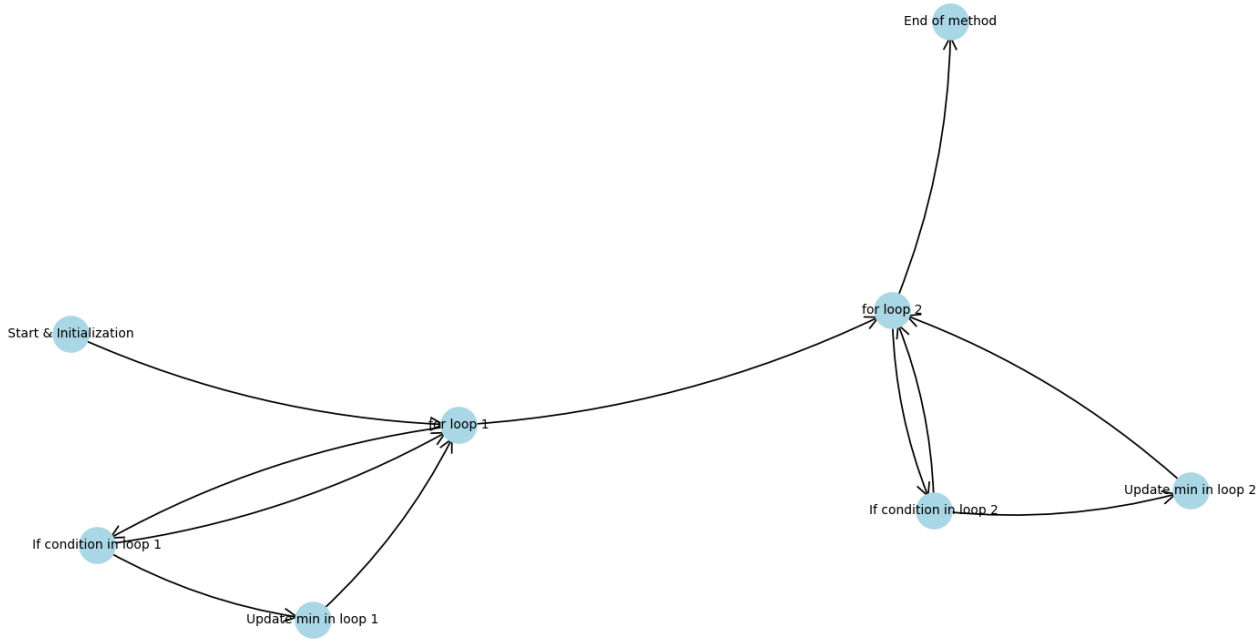
LAB 9
MUTATION TESTING

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 i^{th} 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;
            }
        }
```

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.

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()

nodes = {
    1: "Start & Initialization",
    2: "for loop 1",
    3: "If condition in loop 1",
    3.1: "Update min in loop 1",
    5: "for loop 2",
    6: "If condition in loop 2",
    6.1: "Update min in loop 2",
    8: "End of method"
}

for node, label in nodes.items():
    G.add_node(node, label=label)

edges = [(1, 2), (2, 3), (3, 3.1), (3.1, 2), (3, 2), (2, 5), (5, 6), (6,
6.1), (6.1, 5), (6, 5), (5, 8)]

G.add_edges_from(edges)

plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos, node_color="lightblue", node_size=500)
nx.draw_networkx_labels(G, pos, labels=nx.get_node_attributes(G,
'label'), font_size=5)
nx.draw_networkx_edges(G, pos, edge_color="black", arrowstyle="->",
arrowsize=20, connectionstyle="arc3,rad=0.1")

plt.title("Control Flow Graph for doGraham Method")
plt.axis("off")
plt.show()
```


Control Flow Graph for doGraham Method
End of method
Start & Initialization
for loop 2
for loop 1
If condition in loop 1
If condition in loop 2
Update min in loop 2
Update min in loop 1

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.

COVERAGE CRITERION	TEST SET	INPUT VECTOR (P)	EXPLANATION
Statement Coverage	A	[Point(0,0), Point(1,1), Point(2,2)]	Covers all nodes at least once.
Branch Coverage	B1	[Point(0,0), Point(1,-1), Point(2,2)]	Covers true and false branches in the first loop and false in the second loop.
	B2	[Point(0, 0), Point(2, 0), Point(1, 0)]	Covers false branch in the first loop and true branch in the second loop.
Basic Condition Coverage	C1	[Point(0, 0), Point(1, -1), Point(2, 2)]	First loop condition is true and the second loop condition is false.

	C2	[Point(0, 0), Point(2, 0), Point(1, 0)]	First loop condition is false and both conditions are true in the second loop.
	C3	[Point(0, 0), Point(1, 0), Point(2, -1)]	First loop is both true and false while both conditions are false in the second loop.

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.

P1 (Zero Iterations): []

- **Explanation**: No points, both loops execute zero times.

P2 (One Iteration): [Point(0, 0)]

- **Explanation**: One point, each loop executes once.

P3 (Two Iterations): [Point(0, 0), Point(1, 1)]

- **Explanation**: Two points, each loop executes twice.

P4 (Same yyy-coordinate): [Point(0, 0), Point(1, 0)]

- **Explanation**: Tests the case where two points share the same yyy-coordinate, affecting the second loop.

P5 (Extended Execution): [Point(0, 0), Point(1, 1), Point(2, -1)]

- **Explanation**: Covers cases where the minimum point changes based on different conditions.