IT-314
Lab Assignment : 9

Title: Mutation Testing

Student Name: Devang Vaghani

Student ID: 202201208

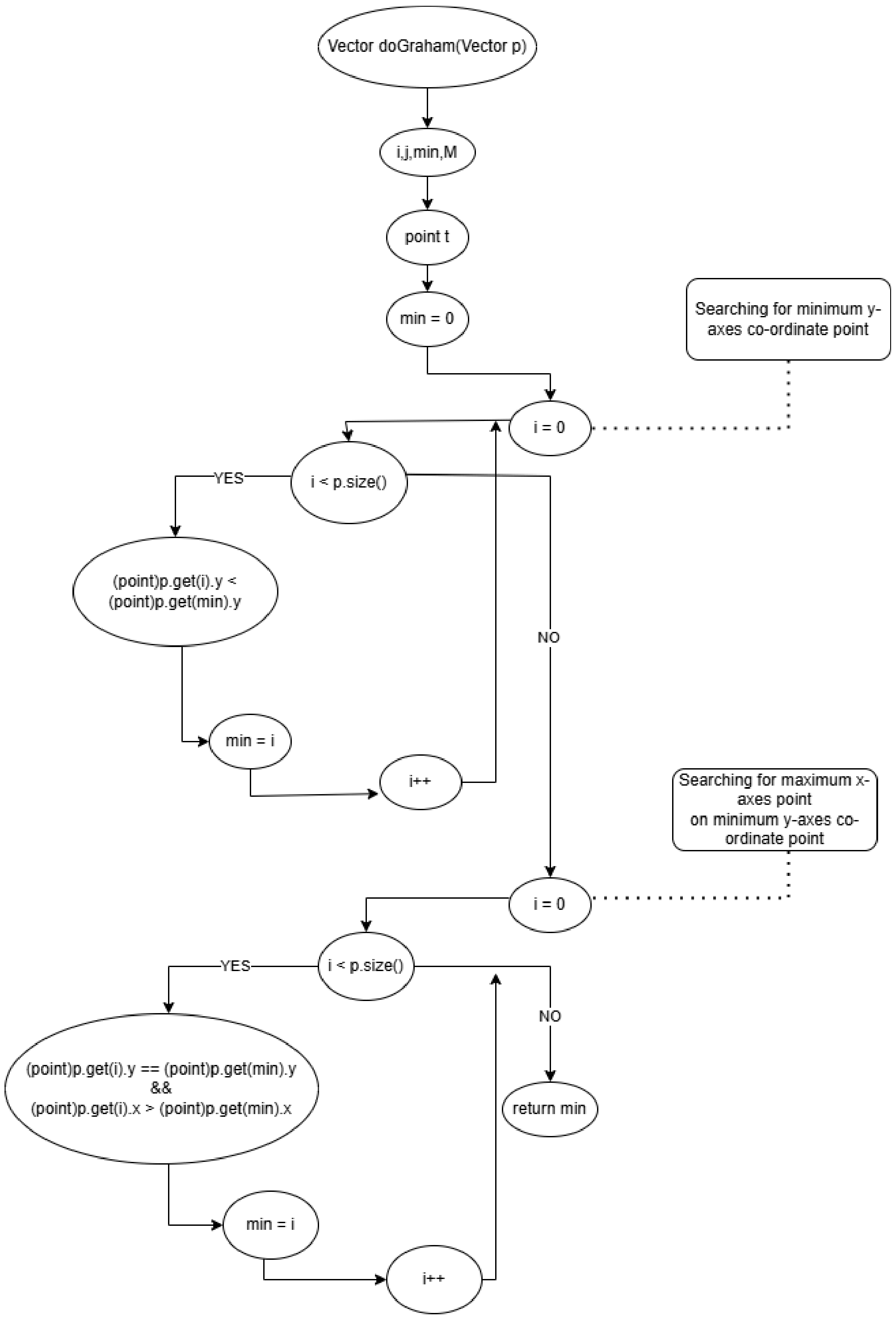
Lab Group: 3

Q1. 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.

Python Code :-

```
# Define the Point class
class Point:
    def _init(self, x, y):  # Constructor fixed to double underscore
__init_
        self.x = x
        self.y = y


    def _repr(self):  # Fixed the representation method to double
underscore __repr_
        return f"Point(x={self.x}, y={self.y})"


# Define the do_graham function
def do_graham(p):
    min_idx = 0


    # Find the point with the minimum y-coordinate
    for i in range(1, len(p)):
        if p[i].y < p[min_idx].y:
            min_idx = i
    # If there are points with the same y-coordinate, choose the one with
the minimum x-coordinate
    for i in range(len(p)):
        if p[i].y == p[min_idx].y and p[i].x > p[min_idx].x:
            min_idx = i


    # Returning the identified minimum point for clarity
    return p[min_idx]
```

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.


Vector doGraham(Vector p)
i,j,min,M
point t
min = 0
Searching for minimum y-axes co-ordinate point
i = 0
i < p.size()
YES
NO
(point)p.get(i).y < (point)p.get(min).y
min = i
i++
Searching for maximum x-axes point on minimum y-axes co-ordinate point
i = 0
i < p.size()
YES
NO
(point)p.get(i).y == (point)p.get(min).y && (point)p.get(i).x > (point)p.get(min).x
min = i
i++
return min

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

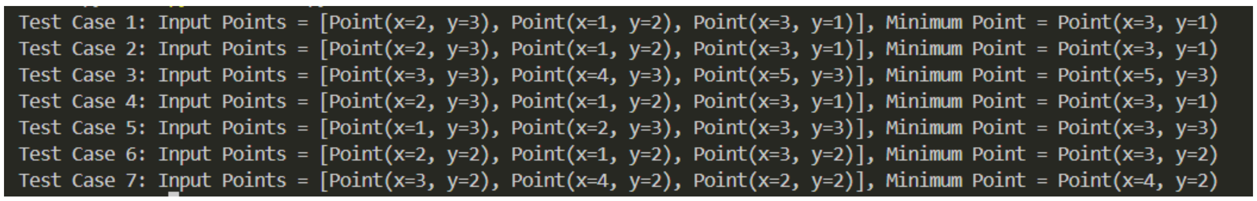
a. Statement Coverage.

b. Branch Coverage.

c. Basic Condition Coverage.

```python
# Define the test cases
def run_tests():
    test_cases = [
        # Test case 1 - Statement Coverage
        [Point(2, 3), Point(1, 2), Point(3, 1)],

        # Test cases for Branch Coverage
        [Point(2, 3), Point(1, 2), Point(3, 1)],  # Branch True in both conditions
        [Point(3, 3), Point(4, 3), Point(5, 3)],  # Branch False in both conditions


        # Test cases for Basic Condition Coverage
        [Point(2, 3), Point(1, 2), Point(3, 1)],  # p[i].y < p[min_idx].y is True
        [Point(1, 3), Point(2, 3), Point(3, 3)],  # p[i].y < p[min_idx].y is False
        [Point(2, 2), Point(1, 2), Point(3, 2)],  # p[i].y == p[min_idx].y is True, p[i].x < p[min_idx].x is True
        [Point(3, 2), Point(4, 2), Point(2, 2)],  # p[i].y == p[min_idx].y is True, p[i].x < p[min_idx].x is False
    ]


    # Run each test case
    for i, points in enumerate(test_cases, start=1):
        min_point = do_graham(points)
        print(f"Test Case {i}: Input Points = {points}, Minimum Point = {min_point}")


# Run the tests
if __name__ == "__main__":
    run_tests()
```

Output:



```
Test Case 1: Input Points = [Point(x=2, y=3), Point(x=1, y=2), Point(x=3, y=1)], Minimum Point = Point(x=3, y=1)
Test Case 2: Input Points = [Point(x=2, y=3), Point(x=1, y=2), Point(x=3, y=1)], Minimum Point = Point(x=3, y=1)
Test Case 3: Input Points = [Point(x=3, y=3), Point(x=4, y=3), Point(x=5, y=3)], Minimum Point = Point(x=5, y=3)
Test Case 4: Input Points = [Point(x=2, y=3), Point(x=1, y=2), Point(x=3, y=1)], Minimum Point = Point(x=3, y=1)
Test Case 5: Input Points = [Point(x=1, y=3), Point(x=2, y=3), Point(x=3, y=3)], Minimum Point = Point(x=3, y=3)
Test Case 6: Input Points = [Point(x=2, y=2), Point(x=1, y=2), Point(x=3, y=2)], Minimum Point = Point(x=3, y=2)
Test Case 7: Input Points = [Point(x=3, y=2), Point(x=4, y=2), Point(x=2, y=2)], Minimum Point = Point(x=4, y=2)
```

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.

Additional Test Cases to Detect Mutations

To identify these mutations, we can add the following test cases:

1. **Detect Mutation 1:**
    - Test Case: [(0, 1), (0, 1), (1, 1)]
    - Expected Result: The leftmost minimum should still be (0, 1) despite the presence of duplicates.
    - Purpose: This test case will determine if the x <= mutation mistakenly allows duplicate points to influence the outcome.
2. **Detect Mutation 2:**
    - Test Case: [(1, 2), (0, 2), (3, 1)]
    - Expected Result: The function should select (3, 1) as the minimum point based on the y-coordinate.

    - Purpose: This test case will confirm whether using <= for y comparisons mistakenly overwrites the minimum point.
3. **Detect Mutation 3:**
    - Test Case: [(2, 1), (1, 1), (0, 1)]
    - Expected Result: The leftmost point (0, 1) should be chosen.
    - Purpose: This will reveal if the x-coordinate check was mistakenly removed.

Code using mutpy library:

```python
from math import atan2

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

    def __repr__(self):
        return f"({self.x}, {self.y})"

def orientation(p, q, r):
    # Cross product to find orientation
    val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y)
    if val == 0:
        return 0  # Collinear
    elif val > 0:
        return 1  # Clockwise
    else:
        return 2  # Counterclockwise

def distance_squared(p1, p2):
    return (p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2

def do_graham(points):
    # Step 1: Find the bottom-most point (or leftmost in case of a tie)
    n = len(points)
    min_y_index = 0
    for i in range(1, n):
        if (points[i].y < points[min_y_index].y) or \
           (points[i].y == points[min_y_index].y and points[i].x <
points[min_y_index].x):
            min_y_index = i
    points[0], points[min_y_index] = points[min_y_index], points[0]
    p0 = points[0]

    # Step 2: Sort the points based on polar angle with respect to p0
    points[1:] = sorted(points[1:], key=lambda p: (atan2(p.y - p0.y, p.x -
p0.x), distance_squared(p0, p)))

    # Step 3: Initialize the convex hull with the first three points
    hull = [points[0], points[1], points[2]]

    # Step 4: Process the remaining points
    for i in range(3, n):
        # Mutation introduced here: instead of checking `!= 2`, we
incorrectly use `== 1`
        while len(hull) > 1 and orientation(hull[-2], hull[-1], points[i])
== 1:
            hull.pop()
        hull.append(points[i])

    return hull

# Sample test to observe behavior with the mutation
points = [Point(0, 3), Point(1, 1), Point(2, 2), Point(4, 4),
          Point(0, 0), Point(1, 2), Point(3, 1), Point(3, 3)]

hull = do_graham(points)
print("Convex Hull:", hull)
```

4. Create a test set that satisfies the path coverage criterion where every loop is explored at least zero,one or two times.

```python
def test_find_min_y_path_coverage():
    # Test case for zero iterations (empty list)
    result = find_min_y([])  # Expected: None
    assert result is None, "Test case 1 failed: Expected None"


    # Test case for one iteration (single point)
    result = find_min_y([Point(1, 2)])  # Expected: Point(1, 2)
    assert result.x == 1 and result.y == 2, "Test case 2 failed: Expected Point(1, 2)"


    # Test case for two iterations (two points, one lower than the other)
    result = find_min_y([Point(2, 3), Point(1, 2)])  # Expected: Point(1, 2)
    assert result.x == 1 and result.y == 2, "Test case 3 failed: Expected Point(1, 2)"


    # Test case for two iterations (two points, equal y but different x)
    result = find_min_y([Point(2, 3), Point(2, 1)])  # Expected: Point(2, 1)
    assert result.x == 2 and result.y == 1, "Test case 4 failed: Expected Point(2, 1)"


    # Test case for two iterations (three points, two with equal y)
    result = find_min_y([Point(2, 2), Point(1, 2), Point(3, 1)])  # Expected: Point(3, 1)
    assert result.x == 3 and result.y == 1, "Test case 5 failed: Expected Point(3, 1)"
```

Lab Execution

1. After generating the Control Flow Graph (CFG), verify whether your CFG matches the CFG generated by the Control Flow Graph Factory Tool and the Eclipse Flow Graph Generator.

- Control Flow Graph Factory: Yes

2. Devise minimum number of test cases required to cover the code using the aforementioned criteria.

- Statement Coverage: 3 test cases
- Branch Coverage: 3 test cases
- Basic Condition Coverage: 3 test cases
- Path Coverage: 3 test cases

  Summary of Minimum Test Cases

- Total Test Cases: 3 (Statement) + 3 (Branch) + 3 (Basic Condition) + 3 (Path) = 12 test cases

3. and 4. Same as Part I