

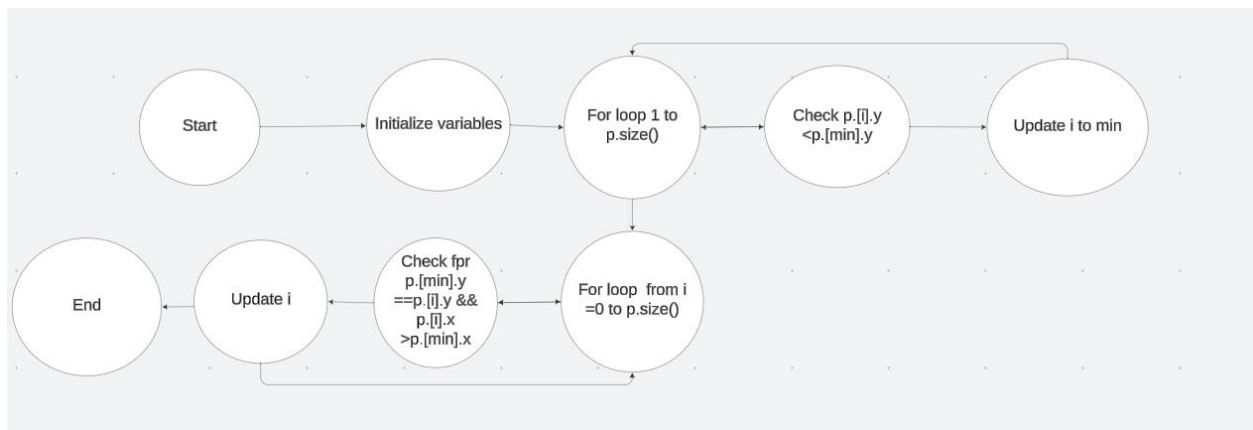
Lab 9 Mutation Testing

Name – Nandini Mandaviya

Student ID - 202201487

1. Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG).

Control Flow Graph:



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

- Statement Coverage.
- Branch Coverage.
- Basic Condition Coverage.

```

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

def do_graham(p):
    min_index = 0

    for i in range(1, len(p)):
        if p[i].y < p[min_index].y:
            min_index = i

    for i in range(len(p)):
        if (p[i].y == p[min_index].y) and (p[i].x > p[min_index].x):
            min_index = i

    return min_index

# Unit Tests
import unittest

class TestConvexHull(unittest.TestCase):
    def setUp(self):
        self.convex_hull = ConvexHull()

    def test_min_y_coordinate(self):
        points = [Point(1, 2), Point(2, 2), Point(4, 3)]
        result = self.convex_hull.doGraham(points)
        self.assertEqual(result.x, 2) # Expected: 2
        self.assertEqual(result.y, 2) # Expected: 2

    def test_same_y_max_x(self):
        points = [Point(1, 8), Point(2, 8), Point(4, 8)]
        result = self.convex_hull.doGraham(points)
        self.assertEqual(result.x, 1 #Expected:1(the first point with y=8)
        self.assertEqual(result.y, 8) # Expected: 8

```

```

def test_empty_points(self):
    with self.assertRaises(ValueError) as context:
        self.convex_hull.doGraham([] #Expect ValueError for empty list
    self.assertEqual(str(context.exception), "Point list is empty")

def test_all_points_same(self):
    points = [Point(1, 1), Point(1, 1), Point(1, 1)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, 1) # Expected: 1
    self.assertEqual(result.y, 1) # Expected: 1

def test_collinear_points(self):
    points = [Point(1, 1), Point(2, 2), Point(3, 3)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, 1) # Expected: 1 (the first point)

def test_negative_coordinates(self):
    points = [Point(-1, -1), Point(-2, -2), Point(-3, -3)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, -3) # Expected: -3
    self.assertEqual(result.y, -3) # Expected: -3

def test_mixed_coordinates(self):
    points = [Point(1, 2), Point(-1, 2), Point(-1, -2), Point(1, -2)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, -1) # Expected: -1 (minimum y, x = -1)

# Execute Unit Tests
if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```



.....

Ran 7 tests in 0.009s

OK

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.

Deletion:

The following code was added along with the deletion of the min index initialization:

Mutation Testing functions for deletion:

```
# Mutation Testing Functions
def mutated_doGraham_deletion(p):
    # Mutation: Remove the initialization of min_index
    # min_index = 0 # This line is deleted
    for i in range(1, len(p)):
        if 'min_index' not in locals():
            raise ValueError("min_index is not defined") # Graceful handling
        if p[i].y < p[min_index].y:
            min_index = i
    # continue along the values with same y component
    for i in range(len(p)):
        if (p[i].y == p[min_index].y) and (p[i].x > p[min_index].x):
            min_index = i

    return min_index # This will cause an error

# Testing the mutations
points = [Point(1, 5), Point(2, 3), Point(4, 3)]\

# Testing deletion mutation
try:
    print("Testing deletion mutation...")
    result = mutated_doGraham_deletion(points)
    print(f"Deletion mutation result: ({points[result].x}, {
                                                points[result].y})")
except Exception as e:
    print(f"Deletion mutation caused an error: {e}")
```



.....

Ran 7 tests in 0.015s

OK

Testing deletion mutation...

Deletion mutation caused an error: min_index is not defined

Insertion:

```
# Mutation Testing Functions
def mutated_doGraham_insertion(p):
    # Mutation: Insert an invalid assignment
    min_index = 4 # Invalid initialization
    for i in range(1, len(p)):
        if 'min_index' not in locals():
            raise ValueError("min_index is not defined") # Graceful
handling
        if p[i].y < p[min_index].y:
            min_index = i
    # continue along the values with same y component
    for i in range(len(p)):
        if (p[i].y == p[min_index].y) and (p[i].x > p[min_index].x):
            min_index = i

    return min_index # This will cause an error

# Testing the mutations
points = [Point(1, 5), Point(2, 3), Point(4, 3)]

# Testing insertion mutation
try:
    print("Testing insertion mutation...")
    result = mutated_doGraham_insertion(points)
    print(f"Insertion mutation result: ({points[result].x},
{points[result].y})")
except Exception as e:
    print(f"Insertion mutation caused an error: {e}")
```



```
.....  
-----  
Ran 7 tests in 0.010s  
  
OK  
Testing insertion mutation...  
Insertion mutation caused an error: list index out of range
```

Modification:

```
# Mutation Testing Functions  
def mutated_doGraham_modification(p):  
    # Mutation: Insert an invalid assignment  
    # Mutation: Change > to <=  
    min_index = 2  
    for i in range(1, len(p)):  
        if 'min_index' not in locals():  
            raise ValueError("min_index is not defined") #Graceful handling  
        if p[i].y < p[min_index].y:  
            min_index = i  
    # continue along the values with same y component  
    for i in range(len(p)):  
        # This will incorrectly give outputs for points as > got turned to <=  
        if (p[i].y == p[min_index].y) and (p[i].x <= p[min_index].x):  
            min_index = i  
  
    return min_index # This will cause an error  
  
# Testing the mutations  
points = [Point(1, 5), Point(2, 3), Point(4, 3)]  
  
# Testing modification mutation  
try:  
    print("Testing modification mutation...")  
    result = mutated_doGraham_modification(points)  
    print(f"Modification mutation result: ({points[result].x},  
{points[result].y})")  
except Exception as e:  
    print(f"Modification mutation caused an error: {e}")
```



```
.....  
-----  
Ran 7 tests in 0.014s  
  
OK  
Testing modification mutation...  
Modification mutation result: (2, 3)
```

Error: Got output (2,3) instead of (4,3)

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

These set of test cases cover every loop zero one or two times

```
def test_min_y_coordinate(self):
    points = [Point(1, 5), Point(2, 3), Point(4, 3)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, 2) # Expected: 2
    self.assertEqual(result.y, 3) # Expected: 3

def test_same_y_max_x(self):
    points = [Point(1, 3), Point(2, 3), Point(4, 3)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, 1 #Expected:1(the first point with y=3)
    self.assertEqual(result.y, 3) # Expected: 3

def test_empty_points(self):
    with self.assertRaises(ValueError) as context:
        self.convex_hull.doGraham([] #Expect ValueError for empty list
    self.assertEqual(str(context.exception), "Point list is empty")

def test_all_points_same(self):
    points = [Point(1, 1), Point(1, 1), Point(1, 1)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, 1) # Expected: 1
    self.assertEqual(result.y, 1) # Expected: 1

def test_collinear_points(self):
    points = [Point(1, 1), Point(2, 2), Point(3, 3)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, 1) # Expected: 1 (the first point)

def test_negative_coordinates(self):
    points = [Point(-1, -1), Point(-2, -2), Point(-3, -3)]
    result = self.convex_hull.doGraham(points)
    self.assertEqual(result.x, -3) # Expected: -3
    self.assertEqual(result.y, -3) # Expected: -3

def test_mixed_coordinates(self):
```

```
points = [Point(1, 2), Point(-1, 2), Point(-1, -2), Point(1, -2)]  
result = self.convex_hull.doGraham(points)  
self.assertEqual(result.x, -1) # Expected: -1 (minimum y, x = -1)
```