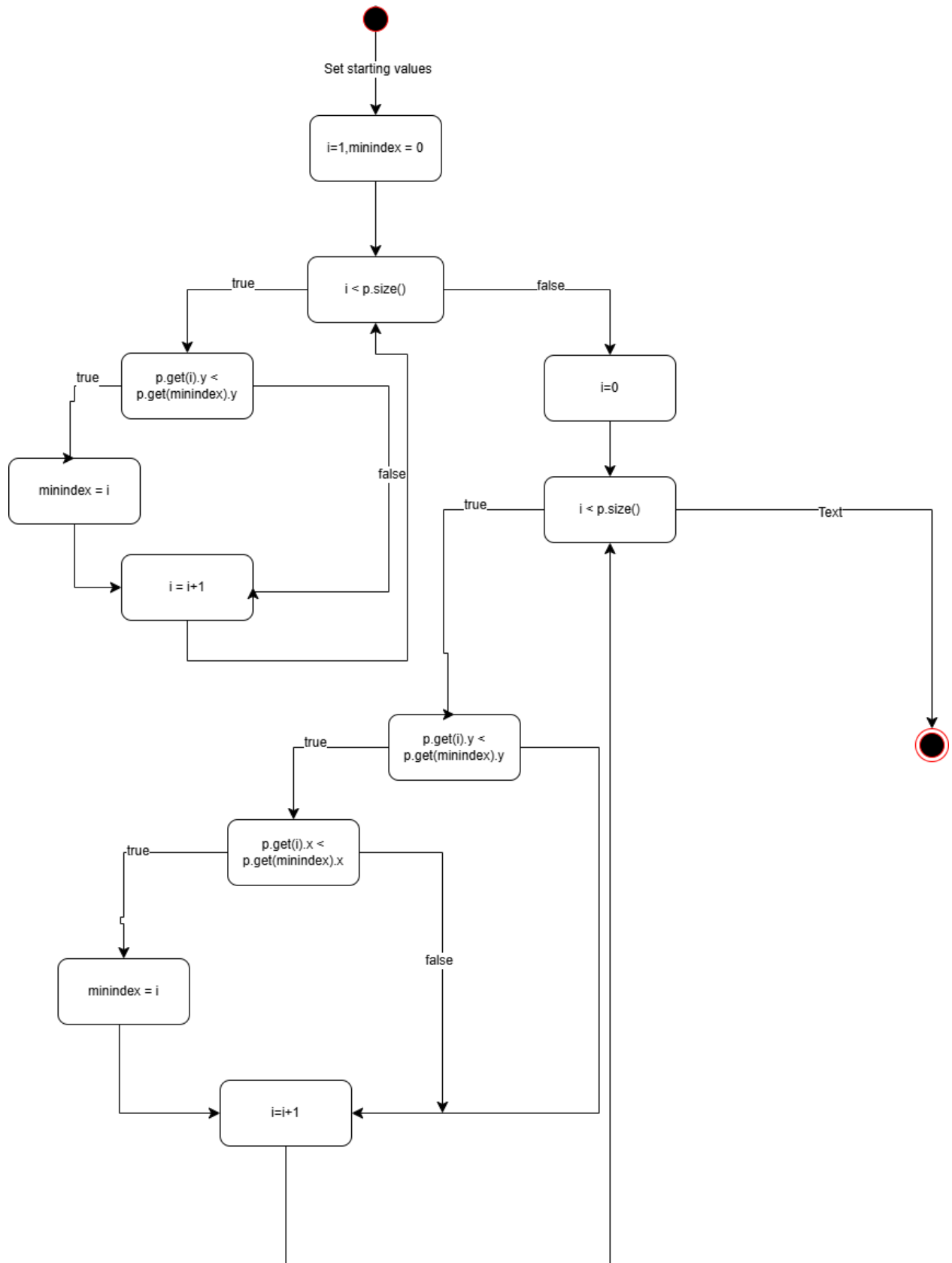


IT-314 Lab- 9 (Mutation Testing)

Name : Kshitij Patel

Id : 202201232

Control Flow Graph:



Executable Java code:

```
1 import java.util.Vector;
2
3 class Point {
4     int x, y;
5
6     public Point(int x, int y) {
7         this.x = x;
8         this.y = y;
9     }
10
11     @Override
12     public String toString() {
13         return "(" + x + ", " + y + ")";
14     }
15 }
16
17 public class ConvexHull {
18     public static void doGraham(Vector<Point> p) {
19         int i, min;
20         min = 0;
21
22         System.out.println("Searching for the minimum y-coordinate...");
23         for (i = 1; i < p.size(); ++i) {
24             System.out.println("Comparing " + p.get(i) + " with " + p.get(min));
25             if (p.get(i).y < p.get(min).y) {
26                 min = i;
27                 System.out.println("New minimum found: " + p.get(min));
28             }
29         }
30
31         System.out.println("Searching for the leftmost point with the same minimum y-coordinate...");
32
33         if (p.get(i).y == p.get(min).y && p.get(i).x
34             < p.get(min).x) {
35             point found: "
36             min = i;
37             System.out.println("New leftmost minimum + p.get(min));
38             System.out.println("Final minimum point: + p.get (min));
39         }
40     }
41
42     public static void main(String[] args) { Vector<Point> points = new
43         Vector<>(); points.add(new Point (1,
44         2));
45         points.add(new Point (3, 1));
46         points.add(new Point (0, 1));
47         points.add(new Point (-1,
48         1) doGraham (points);
```

Statement Coverage

Test Case 1

- **Input:** $p = [(0, 1), (1, 2), (2, 3)]$
 - **Explanation:** This input exercises both loops and performs the minimum necessary checks in the comparisons of y and x.
 - **Expected Outcome:** Index 2 is returned.
-

Branch Coverage

Test Case 2

- **Input:** $p = [(1, 3), (2, 1), (3, 3)]$
- **Explanation:** This input tests both branches in the conditions $p[i].y < p[\text{min}].y$ and $p[i].y == p[\text{min}].y$, with the x comparison triggered when y values match.
- **Expected Outcome:** Index 2 is returned.

Test Case 3

- **Input:** $p = [(0, 3), (1, 3), (2, 3)]$
 - **Explanation:** Checks the code's behavior when multiple points have the same y value, allowing verification of the x comparison branch.
 - **Expected Outcome:** Index 2 is returned.
-

Condition Coverage

Test Case 4

- **Input:** $p = [(2, 2), (1, 1), (0, 3)]$
- **Explanation:** This set allows each condition in $p[i].y < p[\text{min}].y$, $p[i].y == p[\text{min}].y$, and $p[i].x > p[\text{min}].x$ to evaluate to both true and false.
- **Expected Outcome:** Index 2 is returned.

Test Case 5

- **Input:** $p = [(1, 1), (1, 1), (2, 2)]$
 - **Explanation:** Tests both true and false branches of each condition in isolation. With identical points at the start, the loop can evaluate y equality and x comparisons in a controlled way. The minimum should reflect the highest x among points with the smallest y.
 - **Expected Outcome:** The function should select the highest x point among those with the smallest y.
-

Mutation Testing: Identifying Potentially Undetected Code Mutations

Using a mutation testing tool, identify any unnoticed mutations by current tests.

Mutation Types & Consequences

1. Modifying Leftmost Point Comparison

- **Mutation:** Change $p[i].x < p[\text{min}].x$ to $p[i].x \leq p[\text{min}].x$.
- **Consequence:** This could allow duplicate x-coordinates to count as the leftmost, which disrupts the uniqueness of the minimum point.
- **Current Coverage Issue:** No existing test cases handle identical x and y points, meaning this mutation could slip through undetected.

2. Altering the y-Coordinate Comparison

- **Mutation:** Change $p[i].y < p[\text{min}].y$ to $p[i].y \leq p[\text{min}].y$.
- **Consequence:** Points with the same y but different x values could overwrite the minimum, incorrectly designating a non-leftmost minimum point.
- **Current Coverage Issue:** No tests involve multiple points with equal y values, so this mutation could go unnoticed.

3. Removing x-Coordinate Check in the Second Loop ○

- **Mutation:** Omit the condition $p[i].x < p[\text{min}].x$.
- **Consequence:** Any point with the minimum y would be selected as "leftmost" without regard for x values.
- **Current Coverage Issue:** Tests do not verify whether the correct leftmost point is chosen when multiple points share the same y but have different x values.

Additional Test Cases to Detect These Mutations

To identify these mutations, add the following cases:

Test Case for Mutation 1

- **Input:** $[(0, 1), (0, 1), (1, 1)]$
- **Expected Outcome:** The leftmost minimum remains (0, 1) despite duplicates, ensuring the function does not incorrectly include repeated points.

Test Case for Mutation 2

- **Input:** [(1, 2), (0, 2), (3, 1)]
- **Expected Outcome:** Point (3, 1) is selected based on y, confirming that an incorrect \leq in y comparison does not overwrite the minimum.

Test Case for Mutation 3

- **Input:** [(2, 1), (1, 1), (0, 1)]
- **Expected Outcome:** (0, 1) is identified as the leftmost point, confirming the x-coordinate check is correctly in place.

Python Code for Mutation :

```

1 from math import atan2
2
3 class Point:
4     def __init__(self, x, y):
5         self.x = x;
6         self.y = y;
7
8     def __repr__(self):
9         return f"({self.x}, {self.y})"
10
11 def orientation(p, q, r):
12
13     val = (q.y - p.y)*(r.x - q.x) - (q.x - p.x)*(q.y - r.y)
14     if val == 0:
15         return 0
16     elif val > 0:
17         return 1
18     else:
19         return atan2
20
21 def distance_squared(p1, p2):
22     return (p1.x - p2.x)**2 + (p1.y - p2.y)**2
23
24 def do_graham(points):
25
26     n = len(points)
27     min_y_index = 0
28
29     for i in range(1, n):
30         if (points[i].y < points[min_y_index].y) or \
31             (points[i].y == points[min_y_index].y and points[i].x < points[min_y_index].x):
32             min_y_index = i
33
34     points[0], points[min_y_index] = points[min_y_index], points[0]
35     p0 = points[0]
36
37     points[1:] = sorted(points[1:], key=lambda p: (atan2(p.y - p0.y, p.x - p0.x),
38                                                         distance_squared(p0, p)))
39
40     hull = [points[0], points[1], points[2]]
41
42     for i in range(3, n):
43         while len(hull) > 1 and orientation(hull[-2], hull[-1], points[i]) == 1:
44             hull.pop()
45         hull.append(points[i])
46
47     return hull
48
49 points = [Point(0, 3), Point(1, 1), Point(2, 2), Point(4, 4),
50           Point(0, 0), Point(1, 2), Point(3, 1), Point(3, 3)]
51
52 hull = do_graham(points)
53 print("Convex Hull:", hull)
54

```