Lab 3: Iterative Algorithms

Learning Outcomes

- Design an algorithm for a given computational problem statement
- Justify the correctness of an algorithm
- Perform asymptotic complexity analysis of the run time of an algorithm
- Generate test cases for an algorithm
- Correctly implement an algorithm from pseudocode
- Design and execute benchmarks for an algorithm

For this lab we selected
Problem 1: Determine if a Point is Located Inside a Polygon.
The challenge is to determine whether a point lies inside or outside a polygon. The inputs are:

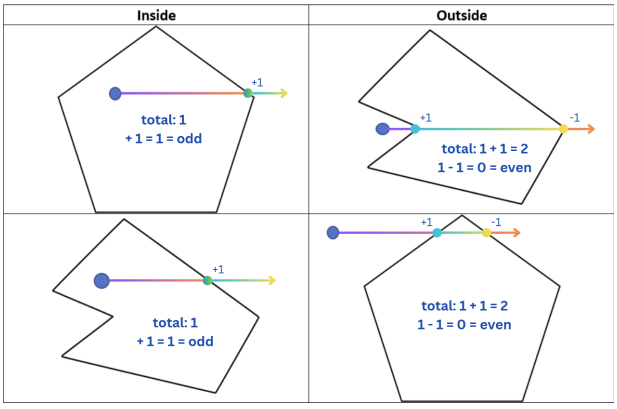
- A sequence $< p_1, p_2, ..., p_n >$ of $n \geq 3$ 2D points. Each point is a pair of x and y coordinates. The points correspond to the vertices of a simple (non-intersecting) polygon. The polygon is connected by line segments between each adjacent pair of points, including a line segment from the last point to the first point.
- The x and y coordinates for a single point distinct from the vertex points.

The output is:

- A Boolean value indicating whether the point is located inside the polygon.

Decision Rule

Draw a horizontal line originating from the point infinitely to the right. If the number of times the line intersects with the point is odd, the point is located inside the polygon.



Pseudocode

```
inclusive = false
points = [list of polygon
   points]
distinct_point = (x, y)

vectors = drawVectors(points)
right_line = rightVector(
   distinct_point)

wn = 0
for vector in vectors:
    if counterclockwise_check(
       vector, right_line) or
       onsegment_check(vector,
       right_line, inclusive):
         wn += 1

return wn % 2 == 1
```

Explanation:

- For each vector, check if the line intersects.
  - If it intersects, increment.
- If the total is odd, return true.

Algorithm Justification

This algorithm is the same one as is used in the *ray-casting method*. The ray-casting parity argument, which the ray-casting method solves, says that a ray from the right of a given point extended to infinity will cross an even number of lines of a polygon if it is outside, and an odd number if it is outside every time. This works because every time the ray crosses a polygon edge, it switches between being "inside" and "outside."

Time Analysis

This algorithm exact analysis runtime is $T(n) = 105n + 6$, with a time complexity of O(n), where n represents the number of vectors which represent the polygon's edges.

Test Cases

Test Case	Shape Type	Test Point(s)	Expected Result
1	Triangle	Polygon Center	Inside (True)
2	Square	Polygon Center	Inside (True)
3	Pentagon	Polygon Center	Inside (True)
4	Hexagon	Polygon Center	Inside (True)
5	Octagon	Polygon Center	Inside (True)
6	Triangle	Various outside points	Outside (False)
7	Square	Various outside points	Outside (False)
8	Pentagon	Various outside points	Outside (False)
9	Hexagon	Various outside points	Outside (False)
10	Octagon	Various outside points	Outside (False)
11	Concave1	Polygon Center	Inside (True)
12	Concave2	Polygon Center	Inside (True)
13	Concave3	Polygon Center	Inside (True)
14	Concave4	Polygon Center	Inside (True)
15	Concave5	Polygon Center	Inside (True)
16	Concave1	Inside Concavity	Outside (False)
17	Concave2	Inside Concavity	Outside (False)
18	Concave3	Inside Concavity	Outside (False)
19	Concave4	Inside Concavity	Outside (False)
20	Concave5	Inside Concavity	Outside (False)
21	Concave5	Inside Concavity	Outside (False)

Table 1: Test Cases and Results for Polygon Inclusion. All Passed

Benchmarking

Polygon Size	Run Time (seconds)
3	3.6×10^{-5}
5	2.2×10^{-5}
10	3.8×10^{-5}
50	1.63×10^{-4}
100	3.20×10^{-4}
500	2.18×10^{-3}
1000	3.90×10^{-3}
2000	7.876×10^{-3}

Table 2: Polygon Size vs. Run Time