Lab 2: Iterative Algorithm Design

Carson Willms, Hudson Arney, David Waas

10/05/2023

Decision Rule:

The decision rule that we found for our algorithm is that all angles before i within the loop constitutes a convex polygon, meaning that if the algorithm is still running, it has yet to determine that the polygon is not a convex polygon. It does this by iterating through each set of three points in the set of points that make of the polygon in question. If the angle between the three points is larger than (or smaller than) 180 degrees.

High Level Pseudocode:

```
If len(dot_list) > 3:
        return “ERROR”
for i in range(-2, dot_list - 2):
        if(angle between dots i, i2, i3 > 180 OR < 180 depending on angle function):
                return false
```

Proving Algorithm is Correct:

To prove the correctness of this algorithm, we first need to understand the decision rule, which states that if the algorithm is continuing through the rest of the points, the previous points all constitute a convex polygon. Also, the algorithms halt once all angles within the polygon have been analyzed. Therefore, given that the algorithms iterate through each angle in the polygon, and only halt when an exception to the convex rule has been found or if the entire polygon is determined to be convex, this algorithm is correct.

Worst Case Run Time w/ Asymptotic Analysis:

The worst case run time can be explained as O(n), because no matter the length of the list, we are iterating through every angle in the polygon, and every angle within a polygon as an associated point. Also, because we are only doing constant time functions within the for loop, the worst-case time complexity is strictly O(n)

```python
def is_convex(dot_list):
    if len(dot_list) < 3:
        return "ERROR"

    prev_cross_product = 0

    for i in range(len(dot_list)):
        i1 = dot_list[i]
        i2 = dot_list[(i + 1) % len(dot_list)]
        i3 = dot_list[(i + 2) % len(dot_list)]

        edge1 = (i2[0] - i1[0], i2[1] - i1[1])
        edge2 = (i3[0] - i2[0], i3[1] - i2[1])

        current_cross_product = cross_product(edge1, edge2)
        # Skip collinear points
        if current_cross_product == 0:
            continue

        if prev_cross_product == 0:
            prev_cross_product = current_cross_product

        # Cross products with different signs indicate concavity
        elif prev_cross_product * current_cross_product < 0:
            return False

    return True
```

Test Cases:

Non-Benchmarked Tests:

“Empty List”: []

“Line”: [(0, 0), (1, 1)]

"Circle": [(0.5 * np.cos(theta), 0.5 * np.sin(theta)) for theta in np.linspace(0, 2 * np.pi, 100)],

"Rectangle": [(0, 0), (2, 0), (2, 1), (0, 1)],

"Star": [(0, 0.5), (0.118, 0.118), (0.5, 0), (0.118, -0.118), (0, -0.5), (-0.118, -0.118), (-0.5, 0), (-0.118, 0.118), (0, 0.5)],

"Pentagon": [(0, 0.5), (0.4755, 0.1545), (0.2939, -0.4045), (-0.2939, -0.4045), (-0.4755, 0.1545)],

"Nothing Polygon": [(0, 0), (0, 0), (0, 0)],

"Concave Triangle": [(0, 0), (1, 0), (0.5, 1)],

"Concave Crescent Moon": [(0, 0), (2, 0), (1.5, 1), (1, 0.5), (0.5, 1)],

"Convex Polygon": [(0, 0), (1, 1), (3, 1), (4, 0), (3, 2), (1, 2)],

"Collinear Points": [(0, 0), (1, 1), (2, 2), (3, 3)]

Benchmarking Estimations:

Shape	Points	Time

Convex	5	10
Concave	5	10
Convex	50	100
Concave	50	100
Convex	500	1000
Concave	500	1000
Convex	5000	10000
Concave	5000	100000
Convex	50000	100000

I'm not quite sure what the actual starting time would take, but considering the algoritm is worst case O(n) time should roughly multiply by 10 each time we increase the number of sides the polygon has by multiplying by 10.

Benchmarking (Also Contained in the Notebook):

	Shape	Points	Time (µs)
0	Convex	5	28.199982
1	Concave	5	9.800016
2	Convex	50	100.900012
3	Concave	50	94.200019
4	Convex	500	901.499996
5	Concave	500	642.999978
6	Convex	5000	5999.100016
7	Concave	5000	5926.999991
8	Convex	50000	63113.200013
9	Concave	50000	60957.100010

Concave Equation: 1.21928x - 21.22413

Convex Equation: 1.26141x + 13.01004

Both equations are near the worst case estimate of O(n) as the leading term "x" is near one, although slightly greater.

Appendix:

Lab 2 Notebook PDF

Notebook Source Code