Queens College of CUNY
Department of Computer Science
Design and Analysis of Algorithms (CSCI 323)
Summer 2024

Assignment #6: "Optimal Triangulation Problem"

Due: June 18, 2024

<u>Overview:</u>

The "Optimal Triangulation Problem" (aka "Minimum-Cost Polygon-Triangulation Problem") is the challenge to divide a convex polygon into non-overlapping triangles such that the total cost of the set of triangles is minimized. The "cost" of each triangle is defined as its perimeter, i.e. the sum of the lengths of its three sides where length is computed using the typical distance formula.

We compare three approaches: recursive, dynamic programming, and memoization. Because of the exponential time complexity associated with the recursive approach, we will need to keep the number of points small.

The following article will be helpful: https://www.geeksforgeeks.org/minimum-cost-polygon-triangulation/

<u>Submissions:</u>

In TopHat, please submit the following, Save the.py and .txt as PDF files so that TopHat can recognize them.

- Assignment6.py (source code)
- Assignment6.txt (console output, including OT calculations and run-time chart)
- Assignment6.png (bar graph comparing runtimes of three OT algorithms)
- Assignment6-ch.png (drawing of convex hull for one size)

<u>Tasks:</u>

[0] Please consult previous assignments for a program template and structure, general guidelines about best practices, as well as sources of and policies concerning using public code.

Wherever possible, import previous assignments (particularly Assignments #4 and #5) and invoke their functions.

[1] Generate a set of n points. Use an n somewhat larger than what you actually want for the triangulation, since points will get eliminated when forming the convex hull / polygon.

[2] Find the convex hull of the points. This will give you a reduced set of points.

You may want to start the exercise here - get the three triangulations to work for fixed inputs - and then randomly obtain a set of points using Tasks 1 and 2.

[3] Compute the optimal triangulation using recursion.

[4] Compute the optimal triangulation using memoization.

[5] Compute the optimal triangulation using dynamic programming.

[6] Verify algorithmically that the answers using all three approaches (Tasks 3, 4, 5) are the same.

[7] Fit the three algorithms into our infrastructure from earlier assignments so all algorithms get executed for random data and their times get tracked, tabulated and plotted.

[8] Modify the triangulation algorithm(s) so it returns not just the minimum cost, but the actual triangulation.

[9] Plot the triangulation in the same grid as the points and the polygon.

You can use this code to draw the triangulation:

```
def draw_triangulated_convex_hull(hull, tri, file_name):
    n = len(hull)
    x = [hull[i][0] for i in range(n)]
    y = [hull[i][1] for i in range(n)]
    plt.plot(x, y, 'ro')
    plt.axis('equal')
    # draw the convex hull
    for i in range(n - 1):
        plt.plot(x[i:i + 2], y[i:i + 2], 'bo-')
    xx = [x[n - 1], x[0]]
    yy = [y[n - 1], y[0]]
    plt.plot(xx, yy, 'bo-')
    # draw the triangulation
    for i in range(n):
        for j in range(n):
            if tri[i][j] >= 0:
                a = i
                b = tri[i][j]
                xx = [hull[a][0], hull[b][0]]
                yy = [hull[a][1], hull[b][1]]
                plt.plot(xx, yy, 'go-')
    plt.savefig(file_name)
    plt.show()
```

You can use this code to run and track the various algorithms:

```
def run_algs(algs, sizes, trials):
    dict_algs = {}
    for alg in algs:
        dict_algs[alg.__name__] = {}
    for size in sizes:
        for alg in algs:
            dict_algs[alg.__name__][size] = 0
        for trial in range(1, trials + 1):
            points = as5.generate_points(size, -10, 10)
            cv = as5.convex_hull_dc(points)
            for alg in algs:
                start_time = time.time()
                ot, tri = alg(cv)
                end_time = time.time()
                net_time = end_time - start_time
                dict_algs[alg.__name__][size] += 1000 * net_time
                if size == sizes[0]:
                    print("\n", alg.__name__, round(ot, 4))
                    print(tri)
            as5.draw_lines(cv, f"Assignment6-cv-{size}.png")
           # draw_triangulated_convex_hull(cv, tri, f"Assignment6-tri-{size}.png")
    return dict_algs
```