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

Assignment #8
"Shortest Path Algorithms"
Due: June 24, 2024

Overview:

This assignment builds on the graph-related functions of the previous assignment, as well as the general infrastructure of several earlier assignments, to implement and study the empirical performance of several algorithm for the Single-Source Shortest Path (SSSP) and All-Pairs Shortest Path (APSP) problems, namely

- Floyd's APSP algorithm (dynamic programming)
- Bellman-Ford SSSP algorithm (dynamic programming)
- Dijkstra's SSSP algorithm, using adjacency/cost matrix and minimization over array (greedy strategy)
- Dijkstra's SSSP algorithm, using adjacency/cost table and minimization by binary heap (greedy strategy)

Submissions:

Use the Google form to submit the following:.

- Assignment08.py (source code)
- Assignment08.txt (console output)
- Assignment08-times.png (bar graph of timings)
- Assignment08-graph.png (graph of points and path)

Tasks:

Follow the template and general guidelines for previous assignments. Wherever possible, import those assignments and invoke their functions rather than creating and maintaining copies of the same code.

[1] Define a function floyd_apsp(graph) that solves APSP for a graph using Floyd's dynamic programming algorithm.
See https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

[2] Define a function bellman_ford_sssp(es, n, src) that takes an edge-set of the graph, the size of the graph, and a starting point, and solves SSSP using the Bellman Ford dynamic programming algorithm.
See https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

[3] Define a wrapper function bellman_ford_apsp(graph) that converts the graph to an edge-set (using the function defined earlier), then calls bellman_ford_sssp for each of the n possible sources.

[4] Define a function dijkstra_sssp_matrix(graph, src) that takes a graph and a starting point, and solves SSSP using Dijsktra's greedy SSSP algorithm, assuming an adjacency matrix and minimization over an array.
See https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[5] Define a wrapper function dijkstra_apsp_matrix(graph) that calls dijsktra_sssp_matrix for each of the n possible sources.

[6] Define a function dijkstra_sssp_table(table, src) that takes a graph and a starting point, and solves SSSP using Dijsktra's greedy SSSP algorithm, assuming an adjacency table and minimization via a min-heap.

See https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

[7] Define a wrapper function dijkstra_apsp_table that calls dijsktra_sssp_table for each of the n possible sources.

[8] Now run and time these four APSP algorithms for ten sizes [10, 20, …, 100], one trial each:

- floyd_apsp
- bellman_ford_apsp (wrapper function)
- dijkstra_apsp_matrix (wrapper function)
- dijkstra_apsp_table (wrapper function)

[9] Print out the timings in a Pandas data frame, sizes across the top, algorithms along the side

[10] Plot the timings in a bar-graph. Make sure that titles, labels, and ticks are updated for the current assignment.

[11] Have the program print out and draw the weighted graph in its various formats, as well as the results for the four algorithms.Print only for the case size = 10 and verify that the results of the algorithms are consistent. Do not print for the larger sizes as that will overwhelm the output.