MATH 8160, Spring 2022
Network Algorithms and Data Structures
Project 2: Shortest Paths

Due: 17 March 2022

In this project, you will develop an all-pairs shortest-path code with a general algebraic structure. You will implement the Floyd-Warshall all-pairs algorithm, but applying customizable operations for the parallel and series reductions, as described in Section 4.6 of Shier's notes.

Input is in the DIMACS format for graph and network problems. The DIMACS format supports representation of minimum-cost flow problems, so you should encode your problems as a minimum-cost flow.

You are provided with a C program, `netgen`, which can generate instances of MST according to specifications provided in an input file or stdin.

DIMACS format is line-oriented. The type of record a line contains is indicated by the character in position 1.

- A `c` indicates a comment line.
- There is one record with type `p`. This record tells the problem type (for shortest-path problems, this field is always `min`), the number n of nodes and the number m of arcs.
- There is a record of type `n` for each node with nonzero supply or demand. For shortest-path problems, the origin node should have a supply of $n - 1$ and each non-source node should have supply -1. The nodes generated by the netgen problem generator are indexed from 1 to n. You may translate node numbers to internal indices by subtracting 1 from the input index.

- There is a record for each arc with type `a`. Each `a` record contains three integers: the indices of the tail end head nodes of the arc and the arc weight.

The algebra is selected either by including a `c` record containing the keyword `algebra` and the code `mp`, `mt`, `mm`, or `oa` corresponding to the entries in the table on page 61 of Shier's notes, or by providing the code as a command-line input option.

Output is a pair of $n \times n$ matrices W and P, one row per line, with w_{ij} equal to the total path weight according to the algebra selected and p_{ij} equal to the predecessor of node i on the path to node j.

Testing

Test your algorithm thoroughly on problems generated by `netgen` and other problems of your devising. I will run your code on a test set of my devising to check correctness and robustness.

Notes

- Make your code modular, robust, and general. Detect errors gracefully, rather than just crashing.
- Document your code adequately.
- Be efficient. Avoid excessive copying and temporary variables, unnecessary special cases, etc.
- Timings should exclude input and output time, but should include setup and running time.
- Check the input for reasonableness, e.g, that probabilities in the reliability algebra are between zero and one, capacities are positive, etc.

Turn in readable, well documented code and an a PDF report documenting your algorithm and data structures, key program variables, implementation details, and performance analysis.