MATH 8160, Spring 2019
Network Algorithms and Data Structures
Project 2: Minimum Spanning Trees

Due: 1 March 2019

In this assignment, you will develop a solver for the *minimum spanning tree* (MST) problem. The goals of the assignment are

- to implement an efficient MST solver; and
- to investigate empirical performance of your code.

You may choose to implement one of Kruskal's, Prim's, or Sollin's algorithms. Use an appropriate heap structure to maintain the list of candidate edges/nodes and an appropriate union-find structure for the connected components if you implement Kruskal or Sollin. If you implement d-heaps (Prim's algorithm), your class should include d as a parameter for the constructor. You should have methods for heapify() (turn a list in arbitrary order into a heap), find_min(), delete_min(), insert () and decrease_key(). If you implement Sollin's algorithm, you should use leftist heaps.

Input format. Input is in a version of the DIMACS format for graph and network problems. This is a line-oriented format. 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 (always `mst`), the number n of nodes and the number m of edges.

- There is a record for each edge with type `e`. Each `e` record contains three integers: the indices of the two end nodes of the edge and the edge weight. The nodes are indexed from 0 to $n-1$.

You are provided with a short Python program, `mstgen`, which can generate instances of MST according to specifications provided on the command line.

Usage:

`mstgen` n m $c_{\min}$ $c_{\max}$ $[h\ [s]]$

where n is the number of nodes, m is the number of edges (must be between $n-1$ and $n(n-1)$), and the range of edge weights is from $c_{\min}$ to $c_{\max}$. The optional parameter h is the minimum percentage of edges set to c_{max} and the optional parameter s is the random-number generator seed. The parameter h must be provided if s is, but may be set to zero (the default). The distribution of costs is uniform in the given range, except that at least $h\%$ are set to $c_{\max}$.

Output format. Output should be in DIMACS format, with `e` records for the edges in the MST including end nodes and weights and a `c` record with the total cost of the tree.

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

2. Investigate the empirical complexity of your algorithm using $d \in \{2,3,4\}$ and any other values you think are of interest. Use the Python timeit module to generate timing data. Trying to time a single run on a small dataset will result in times that are too short to measure reliably. The timieit facility runs multiple instances of your code with the same data and finds an average time.

3. Determine the time and space complexity of your algorithm in terms of relevant network parameters.

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.
- Make your graph and heap facilities standalone modules.
- Timings should exclude input and output time, but should include setup and running time.

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