CS 577: Introduction to Algorithms — Homework P1

Out: 02/28/12 — **Due: 03/09/12 and 03/13/12**

In this homework your goal is to study the behavior of random graphs with respect to the sizes of their minimum spanning trees. The assignment is divided into three parts: basic implementation, design exploration, and analysis.

Guidelines

- **Programming language.** You may use any language to implement your algorithm. We will run your program on a standard CS instructional machine. Please avoid using a non-standard software package.
- **Collaboration.** You are highly encouraged to work in pairs.
- **Due date & lateness.** The code part of the homework is due on March 9 at midnight CST. Please see submission instructions below. The write-up is due on March 13 in class. No extensions will be given.
- **Grading.** See the section titled "turn-in" below.
- Start working on your homework early. Plan your work in such a way that you have enough time past the basic programming component to explore the code optimization and analysis components. Good luck!

Part I: Basic implementation

(a.) Implement an algorithm for finding the minimum spanning tree in a given weighted complete graph (i.e. a graph in which every edge is present). Your program should have the following input/output behavior.

The input consists of $\binom{n}{2}+1$ lines where n is the number of nodes in the graph. The first line consists of a single integer, n. The subsequent lines consist of three numbers each, u, v, w, where u and v are integers in the range 0 to $n-1$ and indicate an edge in the graph, and w is the weight of the edge. Weights lie between 0 and 1.

The output consists of n lines. The first $n-1$ lines consist of two integers each, indicating the edges belonging to a minimum spanning tree, and the last line gives the total weight of the tree.

For example, when the input is as follows:

```
3
0 1 0.1
0 2 0.5
1 2 1.0
```

the output should be as follows:

```
0 1
0 2
0.6
```

(b.) Implement a procedure that takes as input an integer n and produces a complete graph with each edge having a weight chosen independently and uniformly at random from the interval $[0, 1]$. The output of this procedure should match the input format for your program in (a.). You will require the use of a random number generator on your system and will need to determine how to seed it — say, with a value from the machine's clock.

Part II: Design exploration/code optimization

(c.) You will find that when n is very large your program will use a lot of memory. Explore heuristic ways of reducing your memory requirements and/or making your program run faster.

Here is a suggestion. When n is large enough, because all edge weights are between 0 and 1, the minimum spanning tree is unlikely to contain an edge with weight close to 1, say 0.9. Then you can just throw away (or "ignore") all of the edges that have weight larger than 0.9 and run your program on the remaining graph. Think about whether this could lead to an incorrect solution and how you would deal with that possibility.

What is a good threshold for screening edges in this manner? This threshold would depend on n – as n gets larger, you can throw away edges with smaller and smaller weight. Experiment on graphs with small n to come up with a conjecture for a good threshold $k(n)$, and extrapolate to larger n.

Part III: Analysis

(d.) Run your program for $n = 8, 16, 32, 64, ..., 4096, 8192$, and larger values, if your program runs fast enough. For each value of n run the program at least five times. (Make sure you seed the random number generator appropriately each time!) Record the following quantities for each run and average them over the runs for each n: the running time of your algorithm, the size of the MST, the weight of the minimum weight edge in the MST, the weight of the maximum weight edge in the MST. Also for the size of the MST, record for each n, the minimum and maximum over the different runs. This gives an idea of the variance in this quantity; the difference between the minimum and the maximum should decrease as n grows.

Part IV: Turn-in

1. Submit your program from (a.) using instructions below by March 9. We will test your code for correctness on "small" graphs with n ranging up to 20 (but not test your analysis or running time). A correctly functioning code is worth 8 points.

2. Submit a 1-4 page write up explaining your design choices and noting your observations as described below. This part of the assignment is due on March 13 and is worth 7 points. Emphasis will be placed on effort rather than correctness. Up to 2 bonus points may be awarded for interesting insights and observations.

   - Which MST algorithm did you use and why?
   - What is the asymptotic worst case running time of your algorithm? How does your empirical observation measure up to this?
   - What design choices did you make in (c.) and why?
   - If you chose to drop edges, how did you estimate $k(n)$ and how effective was this approach?
   - Plot the quantities that you recorded in (d.) against n. Can you give a rough explanation of the behavior you saw? (You don't need to prove anything or try to derive expressions; That is possible to do, but requires sophisticated techniques.)
   - (Optional) Did you have any interesting experiences with the random number generator? Do you trust it?

Submission instructions

To hand in the assignment, create a new folder on a computer in the CS department running linux. Name the folder `<lastname1>-<lastname2>` substituting the names of your group members for the blanks and put all your code files inside along with a short README describing how to compile and run your code. Then run the following code from the command line:

```
/s/handin/bin/handin -c cs577-2 -a proj1 -d <directory>
```

replacing `<directory>` with the path to your directory.

Complete instructions can also be found at: http://research.cs.wisc.edu/twiki/bin/view/CSDocs/HandinSystem. The class name is cs577-2, the assignment name is proj1, and the directory name is the name of the folder where you stored your project files.