CS 577: Introduction to Algorithms — Homework P2

Out: 03/27/12 — **Due: 04/13/12 and 04/17/12**

Many phenomena in nature and mathematics display what is called a phase transition: an abrupt or discontinuous change in phase or state. For example, if you lower the temperature of water gradually, at $0°$C, it abruptly transitions from the liquid state to the solid state. In this homework, we will examine phase transitions in graphs. The analog of temperature in our setting will be the density of (i.e. the number of edges in) the graph and the property we are interested in is the connectivity of the graph. Specifically, we will consider adding edges randomly to a graph to observe the following properties: at what point does the graph become connected, how do the sizes of the components in the graph evolve, and how does the diameter of the graph evolve.

This homework involves implementing the union-find data structure and an all-pairs shortest paths algorithm. The data structure will be useful for determining connectivity and component sizes as more and more edges are added, and the algorithm will be useful for studying the evolution of the diameter of the graph.

In the remainder of this document we first describe the experiments, and then discuss implementation issues. Details of what you should submit and when are in the last section.

Guidelines

Same as for homework P1.

Experiments

For this homework you need to perform the three experiments described below on **unweighted** graphs of varying sizes. In each experiment, you should start with a graph with n nodes and no edges. Your program should add edges to the graph **in random order**. Let $p \in (0, 1]$ denote the density of the graph, or the number of edges added divided by $\binom{n}{2}$. Repeat each experiment 100 times for $n = 100, 200, 400, 800$, and 1600.

Experiment 1. In this experiment your goal is to determine at what value of p the graph gets connected. In particular, for every fixed n and for every value of p, note the number of runs out of the 100 for which the graph got connected before $p\binom{n}{2}$ edges were added. Plot this number against p. (You may use a single plot for all of the different n.) This should be an increasing function of p and should display a phase transition.

Let p^* denote the (approximate) value of p at which the phase transition occurs. Formulate a conjecture for how p^* depends on n.

Experiment 2. In this experiment your goal is to understand how the average size of the connected components in the graph grows as more and more edges are added. Suppose that the components are of size $s_1, s_2, s_3, \cdots$. Then, we can write the average size of clusters *weighted proportional to the number of nodes they contain* as $(\sum_i s_i^2)/n$.

For every value of n and for every value of p, compute the sum $(\sum_i s_i^2)/n$ and average it over the 100 runs. Then plot this averaged sum against p. As in experiment 1, you may plot all of these functions in the same figure.

Experiment 3. In this experiment your goal is to understand how the diameter of the graph changes as more and more edges are added. For every value of n and for a few different values of p between p^* and 0.5 (say about 10 different values), determine the diameter of the graph and average this number across the 100 runs (disregarding those runs where the graph is not yet connected). Plot this average diameter against p.

Implementation issues

In order to generate the graph, you will have to use a random number generator (RNG) to pick and add edges to the graph in random order. In order to do so, you might want to first rescale the output of the RNG to be an integer in the range $\{1, \cdots, n\}$, and draw two such integers. What would you do about repeats? Don't forget to seed the RNG properly!

For experiments 1 and 2, you should implement the union find data structure on vertices of the graph to discover components as they are formed. Think about how you might maintain and update the sum $(\sum_i s_i^2)/n$ required for the second experiment as more and more edges are added, without recomputing it from scratch every time.

For experiment 3, you should implement an all-pairs shortest paths algorithm and then return the largest node-to-node distance. Think about the tradeoffs that various shortest paths algorithms give in terms of running time, ease of implementation, etc., before deciding which one to implement.

What to turn-in

Part 1: code; to be turned in via handin; Due: 4/13/12 at 11:59 p.m. Implement an algorithm for computing the diameter of a given unweighted graph. (As mentioned above, we suggest implementing an all-pairs shortest paths algorithm and then returning the largest node-to-node distance.) Your algorithm should have the following input/output behavior.

The input consists of $m + 1$ lines where m is the number of edges in the graph. The first line consists of two integers, n and m. The subsequent lines consist of two numbers each, u, v, where u and v are integers in the range 0 to $n - 1$ and indicate an edge in the graph. The output consists of a single line specifying a single integer, the diameter of the graph.

For example, when the input is as follows:
```
4 4
0 1
0 2
1 2
1 3
```
the output should be as follows:
```
2
```

We will test your code for correctness on "small" graphs with n ranging up to 100 (but not test your analysis or running time). A correctly functioning code is worth 8 points.

Part 2: analysis; to be turned in on paper; Due: 4/17/12 in lecture. Explain any implementation choices that you made. Present and discuss your observations from experiments 1-3 above. Your writeup should be 1-3 pages long. This part of the assignment is worth 7 points. Emphasis will be placed on effort rather than correctness.

Submission instructions for handin

To hand in the assignment, create a new folder on a computer in the CS department running linux. Put all your code files inside along with a short README describing how to compile and run your code. Note that the input/output behavior for your code should be exactly as described above. If your code has the extra functionality of being able to read an input file, please specify clearly in the README how to run your code with that functionality. Also, in the README specify the names of the members in your group. Then run the following code from the command line:

```
/s/handin/bin/handin -c cs577-2 -a proj2 -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 proj2, and the directory name is the name of the folder where you stored your project files.