Introduction to Algorithms	**Problem Set 5**
CS 482 Spring 2004	**Due March 12, 2004**

Please hand in each problem on a separate sheet with your name and NetID on each.

Reading: Sections 5.1-5.4 and 5.7-5.8. (Also, Section 5.5 discusses a faster maximum flow algorithm; you should know the statement of the result from this section, but are not responsible for all the details.)

(1) *(This is Problem 37 at the end of Chapter 5.)* Suppose you're a consultant for the Ergonomic Architecture Commission, and they come to you with the following problem.

They're really concerned about designing houses that are "user-friendly," and they've been having a lot of trouble with the set-up of light fixtures and switches in newly designed houses. Consider, for example, a one-floor house with n light fixtures and n locations for light switches mounted in the wall. You'd like to be able to wire up one switch to control each light fixture, in such a way that a person at the switch can *see* the light fixture being controlled.

Sometimes this is possible, and sometimes it isn't. Consider the two simple floor plans for houses in the accompanying figure. There are three light fixtures (labeled a, b, c) and three switches (labeled 1, 2, 3). It is possible to wire switches to fixtures in the example on the left so that every switch has line-of-sight to the fixture, but this is not possible in the example on the right.

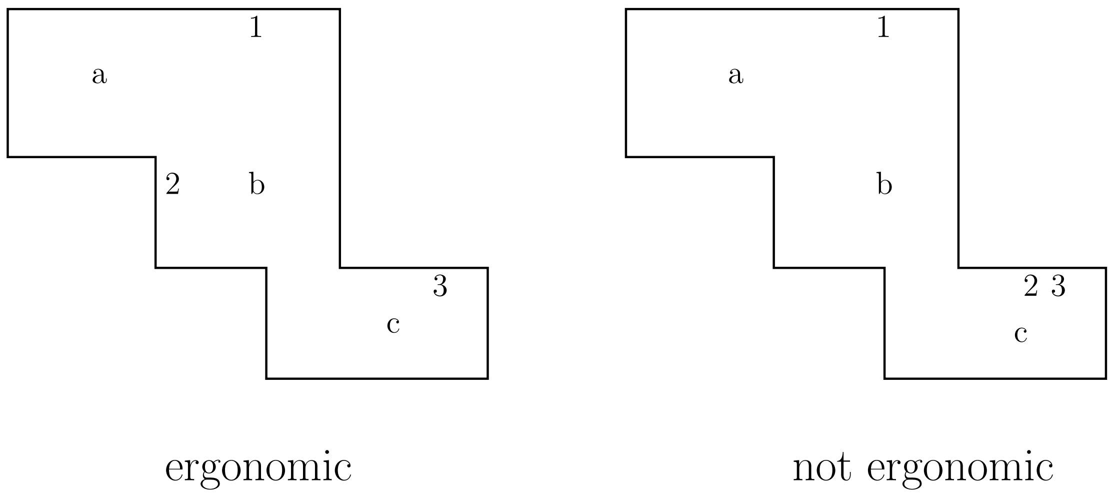

Figure 1: Two floor plans with lights and switches.

Let's call a floor plan — together with n light fixture locations and n switch locations — "ergonomic" if it's possible to wire one switch to each fixture so that every fixture is visible from the switch that controls it. A floor plan will be represented by a set of m horizontal or vertical line segments in the plane (the walls), where the i^{th} wall has endpoints $(x_i, y_i), (x'_i, y'_i)$. Each of the n switches and each of the n fixtures is given by its coordinates in the plane. A fixture is *visible* from a switch if the line segment joining them does not cross any of the walls.

Give an algorithm to decide if a given floor plan, in the above representation, is ergonomic. The running time should be polynomial in m and n. You may assume that you have a subroutine with $O(1)$ running time that takes two line segments as input and decides whether or not they cross in the plane.

(2) *(This is Problem 4 at the end of Chapter 5.)* Suppose you're looking at a flow network G with source s and sink t, and you want to be able to express something like the following intuitive notion: some nodes are clearly on the "source side" of the main bottlenecks; some nodes are clearly on the "sink side" of the main bottlenecks; and some nodes are in the middle. However, G can have many minimum cuts, so we have to be careful in how we try making this idea precise.

Here's one way to divide the nodes of G into three categories of this sort.

- We say a node v is *upstream* if for all minimum s-t cuts (A, B), we have $v \in A$ — that is, v lies on the source side of every minimum cut.
- We say a node v is *downstream* if for all minimum s-t cuts (A, B), we have $v \in B$ — that is, v lies on the sink side of every minimum cut.
- We say a node v is *central* if it is neither upstream nor downstream; there is at least one minimum s-t cut (A, B) for which $v \in A$, and at least one minimum s-t cut (A', B') for which $v \in B'$.

Give an algorithm that takes a flow network G, and classifies each of its nodes as being upstream, downstream, or central. The running time of your algorithm should be within a constant factor of the time required to compute a *single* maximum flow.

(3) You've been called in to help some network administrators diagnose the extent of a failure in their network. The network is designed to carry traffic from a designated source node s to a designated target node t, so we will model it as a directed graph $G = (V, E)$, in which the capacity of each edge is 1, and in which each node lies on at least one path from s to t.

Now, when everything is running smoothly in the network, the maximum s-t flow in G has value k. However, the current situation – and the reason you're here – is that an attacker has destroyed some of the edges in the network, so that there is now no path from s to t using the remaining (surviving) edges. For reasons that we won't go into here, they believe the attacker has destroyed only k edges, the minimum number needed to separate s from t (i.e. the size of a minimum s-t cut); and we'll assume they're correct in believing this.

The network administrators are running a monitoring tool on node s, which has the following behavior: if you issue the command *ping*(v), for a given node v, it will tell you whether there is currently a path from s to v. (So *ping*(t) reports that no path currently exists; on the other hand, *ping*(s) always reports a path from s to itself.) Since it's not practical to go out and inspect every edge of the network, they'd like to determine the extent of the failure using this monitoring tool, through judicious use of the *ping* command.

So here's the problem you face: give an algorithm that issues a sequence of ping commands to various nodes in the network, and then reports the *full* set of nodes that are not currently reachable from s. You could do this by pinging every node in the network, of course, but you'd like to do it using many fewer pings (given the assumption that only k edges have been deleted). In issuing this sequence, your algorithm is allowed to decide which node to ping next based on the outcome of earlier ping operations.

Give an algorithm that accomplishes this task using only $O(k \log n)$ pings.