CS 4820, Summer 2010 Out: July 6. **Due: July 9, 8:30am**

Homework 1

Reading: Read Chapter 1, Review Chapters 2 and 3.

Problem 1. A more advanced civilization from the Alsaurian homeworld finds our practice of putting stop signs on every intersection a little too indulging. They say that on every intersection, two road meet, and one of them is smaller than the other. Well, that is a little simplification, but let us give them that. They argue that the idea behind the stop signs is to let the people on the smaller road know that they are about to intesect a bigger road and therefore they should stop and let traffic on the other road clear up (and yes, they are right about this thing). So, they propose a "better" way: each road should have a number associated with it, and when two roads meet, the traffic on the lower numbered road yields to the traffic on the higher numbered road. (You can think of road number as "priority", where higher number means higher priority.) With this modified system, you just need to remember the priority of the road you are travelling on, and on every intersection, there must be a sign telling the priority of the intersecting road. (Yes, that sounds complicated, but Alsaurian habitants argue that they have done simulations on humans and have discovered that humans can deal with such information.) Here is the problem though. The only problem. These alien beings don't quite know how to assign priorities to road such that on every intersection, the bigger road has a higher number and the smaller road has the lower number (seems like they haven't taken 4820!). That is where an algorithms enthusiast like you come in.

There are n roads in the city. As you move on road i from one end to the other, you intersect a number of roads, say road $j, k, l, ...$, and at every intersection (i, j), one road is bigger than the other. Note that a road can be bigger road for one intersection and smaller for another.

This is how the problem is specified to you. The number n (of roads) is specified, and for each road i, you are given a list of intersections and corresponding priorities on these intersections. The list L_i for road i might look like

$$L_i := (j, \text{bigger}), (k, \text{bigger}), (l, \text{smaller}), ...;$$

which says that road i intersects j (and j has higher priority than i), then i intersects k (and k also has higher priority than i), then i intersects l (and l has lower priority than i), and so on. The goal is to assign n positive numbers to n roads (each road gets a unique number), such that if road i and j meet and road i is bigger than road j for that intersection, then the number assigned to road i is larger than the number assigned to road j. See Figure 1.

Given an input, your job is to

(i) either convince the alien representative that it is impossible to assign numbers to roads such that at every intersection higher priority road gets a higher number, or

(ii) or assign the numbers to roads satisfying the above mentioned property,

and you must do one of the two. Analyze your algorithm for its running time.

Problem 2. The summer in upstate New York is so gorgeous that all 4820 students decide to go out camping and hiking over a weekend. Everything is great in the great hills of Catskill on late Friday and Saturday, before they realize on Sunday afternoon that they are *very* tired and *very* lost in the big forest. They want to attend the 8:30 class on Monday morning so badly, that they decide to buckle down and explore their options in the hope of getting out quickly. All of them agree that it is not a good idea to go out in the wilderness alone. They decide that they will go out in pairs in different directions, and report back about what they found (and hopefully one of the group would have found a way out). The only problem is that they cannot seem to agree who will pair up with whom, they just have some preference (over others) about who they want to go with. They want to pair up according to there preferences such that the pairing is *stable*. And lo and behold! One of them remembers the Gale-Shapley stable matching algorithm from the first lecture. They suggest that they should use some variant of stable matching algorithm to find a stable pairing. Can you help them?

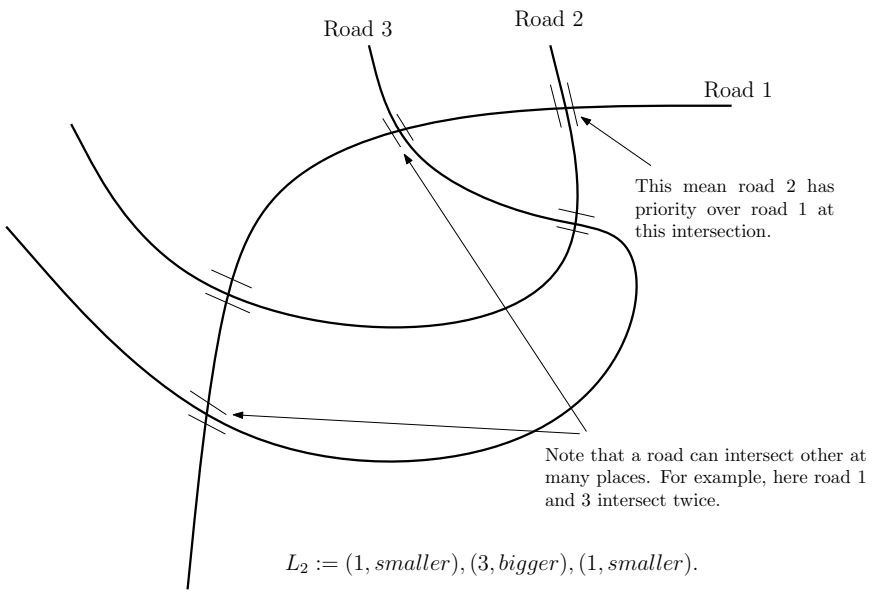

Figure 1: Example for roads in a city for Problem 1.

So, here is the problem. There are $2n$ students. Each student $i \in \{1, 2, ..., 2n\}$ has a preference over rest of them $\{1, 2, ..., i-1, i+1, ..., 2n\}$. Can you find a perfect matching (in which everybody is matched to exactly one person) such that there is no *instability*. Two people (i, j) constitute an instability in a matching M if (i, j) pair is not in the matching and i prefers j to $M(i)$ (partner of i in matching M) and similarly j prefers i to $M(j)$ (partner of j in matching M).

Your task is to

(i) either provide an efficient algorithm that always finds such a stable matching,

(ii) or provide an example of preference lists for which no stable matching exists.

Problem 3. Aliens from the planet Alpha-441 have decided to take over the blue planet. They have sent an intergalactic battleship to do so, which appears to be impervious to every weapon we've ever created. Our top alien experts say there is only one possible way to prevent this tragedy.

They have discovered (somehow!) that this alien vessel is powered by n generators which are spread throughout the massive ship. They also know that there are n service robots, each of which follows its own fixed (and known to alien experts on earth) maintenance schedule. A schedule specifies when the robot is servicing each of the n generators. Note that a robot may sometimes be in transit, and thus at times will not be working on any generator. By the end of each day, each robot has serviced every generator exactly once. Furthermore, no two robots ever service the same generator at the same time.

The plan is as follows. Each of the robots will be infected with a custom-made virus (don't ask me how!). While a virus can't alter a robot's schedule, a virus can cause a robot to sabotage a specific generator while performing its scheduled maintenance. The goal is simply to cause the sabotage of all the generators. But there is a problem. When a generator is sabotaged, it melts down, destroying not only the robot who did the sabotage, but also any other robot that later attempts to service it.

Unfortunately, this makes the task a bit trickier: For example, suppose we make robot r sabotage generator g and robot r' sabotage generator g'. If robot r''s schedule has it servicing g before g' but after r has serviced g (remember we have no control over the timings in the exact schedule, they are arbitrary as long they are consistent with the ordering in the schedule), then r' will be destroyed when it visits generator g and thus will not have the opportunity to sabotage g'.

Give an efficient algorithm to specify which robot should sabotage which generator so that every generator is successfully destroyed.