CSE 521: Design and Analysis of Algorithms
Assignment #1
January 5, 2005
Due: Wednesday, January 12

Reading Assignment: Kleinberg and Tardos, Chapters 1 and 4

Problems:

1. Gale and Shapley published their paper on the stable marriage problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

   Basically, the situation was the following. There were m hospitals, each with a certain number of available positions for hiring residents. There were n medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the m hospitals.

   The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.)

   We say that an assignment of students to hospitals is *stable* if neither of the following situations arises.

   - First type of instability: There are students s and s', and a hospital h, so that
     - s is assigned to h, and
     - s' is assigned to no hospital, and
     - h prefers s' to s.
   - Second type of instability: There are students s and s', and hospitals h and h', so that
     - s is assigned to h, and

- s' is assigned to h', and
- h prefers s' to s, and
- s' prefers h to h'.

So we basically have the stable marriage problem from the section, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Show that there is always a stable assignment of students to hospitals, and give an efficient algorithm to find one.

2. The stable matching problem, as discussed in class and in the course packet, assumes that all men and women have a fully ordered list of preferences. In this problem we will consider a version of the problem in which men and women can be *indifferent* between certain options. As before we have a set M of n men and a set W of n women. Assume each man and each woman ranks the members of the opposite gender — but now we allow ties in the ranking. For example (with $n = 4$), a woman could say that m_1 is ranked in first place; second place is a tie between m_2 and m_3 (she has no preference between them); and m_4 is in last place. We will say that w *prefers* m to m' if m is ranked higher than m' on her preference list (they are not tied).

   With indifferences in the rankings there could be two natural notions for stability. And for each, we can ask about the existence of stable matchings, as follows.

   (a.) A *strong instability* in a perfect matching S consists of a man m and a woman w, such that m and w prefer each other to their partners in S. Does there always exists a perfect matching with no strong instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a strong instability; or give an efficient algorithm that is guaranteed to find a perfect matching with no strong instability.

   (b.) A *weak instability* in a perfect matching S is a man m and a woman w, such that their partners in S are w' and m' respectively, and one of the following holds:

   - m prefers w to w', and w either prefers m to m' or is indifferent between these two choices; or
   - w prefers m to m', and m either prefers w to w' or is indifferent between these two choices.

In other words, the pairing between m and w is either preferred by both, or preferred by one while the other is indifferent. Does there always exists a perfect matching with no weak instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a weak instability; or give an efficient algorithm that is guaranteed to find a perfect matching with no weak instability.

3. There are many other settings in which we can ask questions related to some type of "stability" principle. Here's one, involving competition between two enterprises.

   Suppose we have two television networks; let's call them AOL-Time-Warner-CNN and Disney-ABC-ESPN, or $\mathcal{A}$ and $\mathcal{D}$ for short. There are n prime-time programming slots, and each network has n TV shows. Each network wants to devise a *schedule* — an assignment of each show to a distinct slot — so as to attract as much market share as possible.

   Here is the way we determine how well the two networks perform relative to each other, given their schedules. Each show has a fixed *Nielsen rating*, which is based on the number of people who watched it last year; we'll assume that no two shows have exactly the same rating. A network *wins* a given time slot if the show that it schedules for the time slot has a larger rating than the show the other network schedules for that time slot. The goal of each network is to *win* as many time slots as possible.

   Suppose in the opening week of the fall season, Network $\mathcal{A}$ reveals a schedule S and Network $\mathcal{D}$ reveals a schedule T. On the basis of this pair of schedules, each network wins certain time slots, according to the rule above. We'll say that the pair of schedules (S, T) is *stable* if neither network can unilaterally change its own schedule and win more time slots. That is, there is no schedule S' so that Network $\mathcal{A}$ wins more slots with the pair (S', T) than it did with the pair (S, T); and symmetrically, there is no schedule T' so that Network $\mathcal{D}$ wins more slots with the pair (S, T') than it did with the pair (S, T).

   The analogue of Gale and Shapley's question for this kind of stability is: For every set of TV shows and ratings, is there always a stable pair of schedules? Resolve this question by doing one of the following two things: (a) Giving an algorithm that, for any set of TV shows and associated ratings, produces a stable pair of schedules; or (b) Giving an example of a set of TV shows and associated ratings for which there is no stable pair of schedules.

4. Peripatetic Shipping Lines, Inc., is a shipping company that owns n ships, and provides service to n ports. Each of its ships has a *schedule* which says, for each

day of the month, which of the ports it's currently visiting, or whether it's out at sea. (You can assume the "month" here has m days, for some $m > n$.) Each ship visits each port for exactly one day during the month. For safety reasons, PSL Inc. has the following strict requirement:

(†) *No two ships can be in the same port on the same day.*

The company wants to perform maintenance on all the ships this month, via the following scheme. They want to *truncate* each ship's schedule: for each ship S_i, there will be some day when it arrives in its scheduled port and simply remains there for the rest of the month (for maintenance). This means that S_i will not visit the remaining ports on its schedule (if any) that month, but this is okay. So the *truncation* of S_i's schedule will simply consist of its original schedule up to a certain specified day on which it is in a port P; the remainder of the truncated schedule simply has it remain in port P.

Now the company's question to you is the following: Given the schedule for each ship, find a truncation of each so that condition (†) continues to hold: no two ships are ever in the same port on the same day.

Show that such a set of truncations can always be found, and give an efficient algorithm to find them.

Example: Suppose we have two ships and two ports, and the "month" has four days. Suppose the first ship's schedule is

port P_1; at sea; port P_2; at sea

and the second ship's schedule is

at sea; port P_1; at sea; port P_2

Then the (only) way to choose truncations would be to have the first ship remain in port P_2 starting on day 3, and have the second ship remain in port P_1 starting on day 2.