CS 482 Summer 2005

Homework Assignment #1

Out: July 8 Due: July 13

Question 1

Cornell housing services is trying out a new system for assigning sophomores to their dorm rooms, and they've asked you (yes, *you*) to help. Here's the sitch: they have n double-occupancy rooms, and $2n$ sophomores requiring housing. They've asked each of these $2n$ students to rank the other students in decreasing order of roommate-worthiness. What they want *you* to do is to assign each student a room (and hence a roommate) so that, even if everyone is not necessarily happy with their roommates, they're at least unable to find another student willing to switch and room with them.

Given the students and their preference lists, is there always a way to assign students to rooms such that the assignment is stable? If so, give an algorithm to find such an assignment. If not, give an example of students and preferences such that for any assignment, at least one pair of students prefers each other to their assigned roomies.

Question 2

With your computer science expertise and a good eye for market trends, you've decided to start up an Ithaca speed-dating service. For those unfamiliar with the concept, here's how it will work: you will have n single men and n single women mosey on down to a hip Ithaca bar to try their luck in love. Each man will be seated at his own table, and the women will rotate from table to table for 8-minute one-on-one dates with each man.

As the organizer (and a dater yourself) you know better than to underestimate the power of liquor. . .which is why you serve complimentary champagne to each single. However, champagne is not the cheapest of beverages, and you are trying to figure out how to dispense it without seeming too much a scrooge.

What you've decided to do is to hand out the champagne couple-by-couple, but with two restrictions. The more obvious restriction is to not serve anyone champagne more than once, as that would not be cost effective, or fair to the others, really. The less obvious restriction is to not serve champagne to a woman who will later date a man to whom you haven't yet served champagne.

To be concrete, each single woman has a fixed schedule: for each 8-minute session she is either at a table on a date, or sitting the session out in the lounge area, probably bitching with the other women about the lack of good men in this town. You want to determine at which tables to serve champagne during each session so that everyone is served bubbly by the evening's end, no one gets any twice, and no sober man ever gets paired with a tipsy woman throughout the evening.

Given the women's fixed table schedules, give an algorithm that determines when and at which tables to serve champagne. Hint: Try using the Gale-Shapley algorithm.

Question 3

You've probably heard of the game The Six Degrees of Kevin Bacon. Kevin Bacon has been in a significant number of ensemble films, from *A Few Good Men* to *Mystic River*. Due to this fact, it turns out that if you use Bacon as an end point, you can link him in six degrees or less to almost any other performer. For example, Seth Green takes 2 steps to make a chain: He was in *The Italian Job* with Mos Def, who was in *The Woodsman* with Kevin Bacon. It's actually quite hard to come up with chains of length 3, not to mention 4, 5, or 6 (go to *www.oracleofbacon.org* to give it your best shot).

In terms of graph theory, this statement is equivalent to saying that in the graph where vertices are actors and edges connect two actors if they appear in the same film, almost all vertices are linked to the Kevin Bacon vertex by paths of length at most 6.

Similarly, if you consider the graph where vertices are all the people in the world and edges connect two people if they are acquainted, it is posited that you are connected to any other person in the world by 6 edges, or 6 degrees of circumstance or acquaintance.

Now what if instead of asking about connectivity in this graph, you ask how fast gossip could get back to you? More concretely, suppose there are weights on these acquaintance edges, denoting the length of time it takes for gossip to travel from one endpoint to the other. (So for good friends this number will presumably be low, unless your friends don't like to gossip, but then what kind of friends are they anyway.) Then assuming the gossip begins with you, give an algorithm that finds the shortest time before gossip gets back to you, and along which path. Make sure gossip doesn't travel through any single acquaintance pair more than once.