The Design and Analysis of Algorithms	Final
CS 681 Fall 2002	Due Friday, December 12 at 4pm, 2003

The final is individual work. Please do not discuss these questions with anyone expect with Swamy, Tom and Eva. We will do our best to answer your emails promptly during the week, and also will have office hours every day (see Web).

In writing down your solutions you may use any algorithm we discussed in class without writing out the details of the algorithm. In proving a problem NP-complete, you may use the NP-completeness of any of the problems that we proved NP-complete in class or as a homework. Please do not search the Web for answers. You may use the course packet, Kozen's book, handouts, or any of the recommended books. If you use books other than the course packet, or Kozen's book in your solutions, please give clear reference to the source you used.

(1) Suppose you are managing a system in which asynchronous processes make use of shared resources. Thus, the system has a set of n *processes* and a set of m *resources*. At any given point in time, each process specifies a set of resources that it requests to use. Each resource might be requested by many processes at once; but it can only be used by a single process at a time. Your job is to allocate resources to processes that request them. If a process is allocated all the resources it requests, then it is *active*; otherwise it is *blocked*. You want to perform the allocation so that as many processes as possible are active. Thus, we phrase the Resource Reservation problem as follows: given a set of process and resources, the set of requested resources for each process, and a number k, is it possible to allocate resources to processes so that at least k processes will be active?

Consider the following list of problems, and for each problem either give a polynomial time algorithm or prove that the problem is NP-complete.

(a) The general Resource Reservation problem defined above.

(b) The special case of the problem when k=2.

(c) The special case of the problem when there are two types of resources, say rooms and equipments, each process requires at most one resource of each type.

(d) The special case of the problem when each resource is requested by at most two processes.

(2) Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems like the only option. So they decide to car-pool to work.

Unfortunately, they all hate to drive, so they want to make sure that any car-pool arrangement they agree upon is fair, and doesn't overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

Here's one way to define *fairness*. Let the people be labeled $S = \{p_1, \ldots, p_k\}$. We say that the *total driving obligation* of p_j over a set of days is the expected number of times that p_j would

have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the car-pool plan lasts for d days, and on the i^{th} day a subset $S_i \subseteq S$ of the people go to work. Then the above definition of the total driving obligation Δ_j for p_j can be written as $\Delta_j = \sum_{i:p_j \in S_i} \frac{1}{|S_i|}$. Ideally, we'd like to require that p_j drives at most Δ_j times; unfortunately, Δ_j may not be an integer.

So let's say that a *driving schedule* is a choice of a driver for each day — i.e. a sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ with $p_{i_t} \in S_t$ — and that a *fair driving schedule* is one in which each p_j is chosen as the driver on at most $\lceil \Delta_j \rceil$ days.

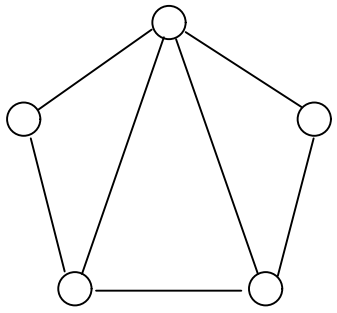
(a) Give an algorithm to compute a fair driving schedule, if one exists, with running time polynomial in k and d.

(b) Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

(c) One could expect k to be a much smaller parameter than d (e.g. perhaps $k = 5$ and $d = 365$). So it could be worth reducing the dependence of the running time on d even at the expense of a much worse dependence on k. Give an algorithm to compute a fair driving schedule whose running time has the form $O(f(k) \cdot d)$, where $f(\cdot)$ can be an arbitrary function. (The function $f(k)$ does not have to be polynomial in k.)

(3) We say that a graph $G = (V, E)$ is a *outerplanar graph* if it consists of the vertices of a convex n-gon in the plane all edges drawn as straight lines in the plane with no edges crossing — in other words, if it can be drawn in the plane as follows.

The vertices are all placed on the boundary of a convex set in the plane (we may assume on the boundary of a circle), with each pair of consecutive vertices on the circle joined by an edge. The remaining edges are then drawn as straight line segments through the interior of the circle, with no pair of edges crossing in the interior. An outerplanar graph is pictured below.



Prove that every outerplanar graph has a tree decomposition of width at most 2, and describe an efficient algorithm to construct such a decomposition. Hint: recall that cycles have a tree decomposition of width at most 2. In the tree decomposition of the cycle we discussed in class, each edge of the cycle is contained in exactly one node of the decomposition tree. You may want to construct a tree decomposition for outerplanar graphs where each edge on the outside n-gon is contained in exactly one node of the decomposition.

(4) Assume you have n balls and n bins, and each ball is placed in a bin selected independently at random (with each bin equally likely). Throughout this problem use the approximation $(1-1/n)^n \approx 1/e$ whenever it is useful.

(a.) Prove that the expected number of empty bins is approaches n/e for large n. Hint: remember that expectation is linear.

(b.) Assume that you have n jobs and n machines, and each job selects a machine independently at random (with each machine equally likely). Assume that if a machine is selected by more than one job, it will do the first job, and reject the rest. What is the expected number of rejected jobs?

(c.) Now assume in the above job-machine example each machine will do the first two jobs, and reject the rest if more than two jobs are assigned to it. What is the expected number of rejected jobs now?