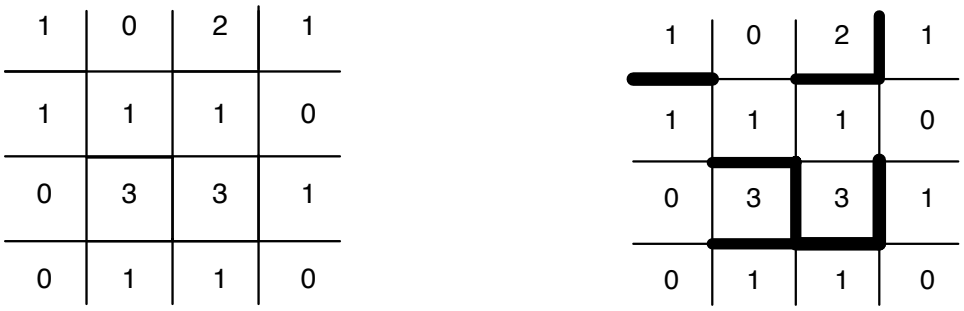
Algorithms	**Problem Set 2**
CS 6820 Fall 2009	**Due October 2, 2009**

Recall the general instructions for handing in homework:

- If possible, please typeset the homework (i.e. format your solutions as an electronic file using latex or Word with mathematical notation).
- Homework solutions done electronically can be handed in by directly uploading them to CMS. Please mail Ashwin (`ashwin85@cs.cornell.edu`) if you have any trouble with this.



(a) *An instance of the grid solitaire puzzle.*

(b) *A solution to this instance.*

Figure 1: An instance of the grid solitaire puzzle.

(1) Consider the following *grid solitaire puzzle*, shown in Figure 1(a). We create a grid by having n vertical lines cross n horizontal lines, creating a set of square cells. (The lines cross in the style of a "tic-tac-toe" board, so the cells on the sides have "open" boundaries.) Each line gets chopped into $n+1$ short line segments by the lines crossing it in the other direction — we call these short line segments the *segments* of the grid. Note that each cell of the grid is bordered by some of the segments; the cells on the sides of the grid are bordered by either 2 or 3 segments, while the cells in the interior are all bordered by 4 segments.

Now we write one of the numbers 0, 1, 2, 3, or 4 inside each cell. The problem is to decide whether it's possible to take some of the segments of the grid and shade them in, as in Figure 1(a), in such a way that the number of shaded segments bordering each cell is equal to the number written inside it. Such a choice of shaded segments is called a *solution* to the instance of the puzzle.

Give a polynomial-time algorithm that takes an $n \times n$ instance of this grid solitaire puzzle and either produces a solution or reports (correctly) that no solution exists.

(2) Consider the following *cube solitaire puzzle*. You are given a pile of n wooden cubes. Each cube has a six faces, and there is a number between 1 and n written on each face. Your goal is to decide whether it's possible to arrange the cubes in a line so that the numbers on the faces pointing upward consist of the numbers $1, 2, 3, \ldots, n$ in order. (So your choices consists both of how to order the cubes, and also which face of each cube to choose as the face that points upward.) Such an arrangement is considered a solution of the puzzle.

Give a polynomial-time algorithm that takes an instance of this cube solitaire puzzle and either produces a solution or reports (correctly) that no solution exists.

(3) Suppose you have an electronic document D_0, and n alternate versions of it $D_1, D_2, \ldots, D_n$ that each differ from the original version D_0. (Think for example of a set of variants of a Wikipedia article.) For each pair of documents D_i and D_j, you have an *edit script* that, when applied to D_i, produces D_j. In this question, we won't be concerned with how these edit scripts are specified, except to note that some are longer than others: the length of the edit script transforming D_i to D_j has length ℓ_{ij}. Here are two observations about the lengths of edit scripts.

- It can easily be the case that $\ell_{ij} \neq \ell_{ji}$. (For example, perhaps D_j consists of D_i plus an extra paragraph, in which case the edit script transforming D_j to D_i needs only specify where to delete some of the lines, whereas the edit script transforming D_i to D_j needs to include the text of this extra paragraph.)

- On the other hand, you can assume that the lengths satisfy an asymmetric version of the triangle inequality: $\ell_{ij} \leq \ell_{ik} + \ell_{kj}$, since one way to transform D_i into D_j would be to first transform it into D_k, and then transform D_k into D_j.

You've been asked to store D_0 together with enough information so that all of the versions $D_1, \ldots, D_n$ can be reconstructed if necessary. You've decided to do this by building the following kind of *reconstruction tree*: it will be a rooted tree, with D_0 as the root and $D_1, \ldots, D_n$ as the other nodes; and for each edge of the tree that connects a parent D_i to a child D_j, you'll store the edit script transforming D_i into D_j on the edge from D_i to D_j. The *complexity* of the reconstruction tree is the sum of the lengths of all the edit scripts stored on the edges.

Give a polynomial-time algorithm that takes the documents and the edit scripts, and produces a reconstruction tree whose complexity is as small as possible.

(4) Consider defining a function $f(\cdot)$ on the subsets of $U = \{1, 2, \ldots, n\}$ as follows. We specify a list of sets $A_1, A_2, \ldots, A_m \subseteq U$, and for any $S \subseteq U$, we define $f(S)$ to be the number of sets among $A_1, \ldots, A_m$ that are subsets of S. In other words, $f(S) = |\{i : A_i \subseteq S\}|$. Notice that although $f(\cdot)$ is defined on all subsets, it has a succinct description via the sets $A_1, \ldots, A_m$.

Give a polynomial-time algorithm that takes the function $f(\cdot)$ (described in the input by the set $A_1, \ldots, A_m$), and decides whether there is a set S for which $f(S) > |S|$.

(5) *(KT Exercise 7.27)* 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 carpool to work.

Unfortunately, they all hate to drive, so they want to make sure that any carpool 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 carpool plan lasts for d days, and on the i^{th} day a subset $S_i \subseteq S$ of the people will be going 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) Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

(b) Give an algorithm to compute a fair driving schedule in time polynomial in k and d.