CS 573: Graduate Algorithms, Fall 2008
Homework 5

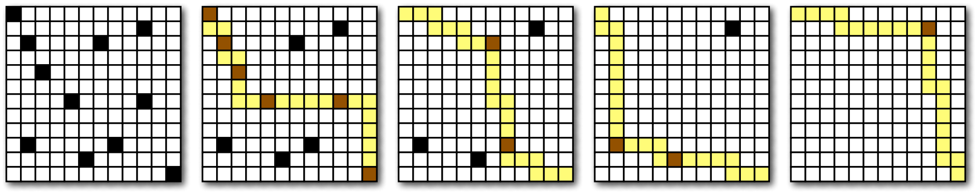
Due at 11:59:59pm, Wednesday, November 19, 2008

- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.

---

1. Recall the following problem from Homework 3: You are given an $n \times n$ grid, some of whose cells are marked; the grid is represented by an array $M[1..n, 1..n]$ of booleans, where $M[i,j] = \text{TRUE}$ if and only if cell (i,j) is marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell.

   Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell.


Greedily covering the marked cells in a grid with four monotone paths.

2. Suppose we are given a directed graph $G = (V, E)$, two vertices s an t, and a capacity function $c: V \to \mathbb{R}^+$. A flow f is *feasible* if the total flow into every vertex v is at most $c(v)$:

$$\sum_u f(u \to v) \le c(v) \quad \text{for every vertex } v.$$

   Describe and analyze an efficient algorithm to compute a feasible flow of maximum value.

3. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

4. *Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle[1], these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other situations where people might want to monitor conditions in hard-to-reach areas. The idea is that a large collection of cheap, simple devices could be dropped into the area from an airplane (for instance), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

   The devices can communication only within a limited range. We assume all the devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D.

   We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit the information it has to some other *backup* device within its communication range. To improve reliability, we require each device x to have k potential backup devices, all within distance D of x; we call these k devices the ***backup set*** of x. Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

   So suppose we are given the communication radius D, parameters b and k, and an array $d[1..n, 1..n]$ of distances, where $d[i, j]$ is the distance between device i and device j. Describe an algorithm that either computes a backup set of size k for each of the n devices, such that that no device appears in more than b backup sets, or reports (correctly) that no good collection of backup sets exists.

5. Let $G = (V, E)$ be a directed graph where for each vertex v, the in-degree and out-degree of v are equal. Let u and v be two vertices G, and suppose G contains k edge-disjoint paths from u to v. Under these conditions, must G also contain k edge-disjoint paths from v to u? Give a proof or a counterexample with explanation.

*6. ***[Extra credit]*** A *rooted tree* is a directed acyclic graph, in which every vertex has exactly one incoming edge, except for the *root*, which has no incoming edges. Equivalently, a rooted tree consists of a root vertex, which has edges pointing to the roots of zero or more smaller rooted trees. Describe a polynomial-time algorithm to compute, given two rooted trees A and B, the largest common rooted subtree of A and B.

   [Hint: Let $LCS(u, v)$ denote the largest common subtree whose root in A is u and whose root in B is v. Your algorithm should compute $LCS(u, v)$ for all vertices u and v using dynamic programming. This would be easy if every vertex had $O(1)$ children, and still straightforward if the children of each node were ordered from left to right and the common subtree had to respect that ordering. But for unordered trees with large degree, you need another trick to combine recursive subproblems efficiently. Don't waste your time trying to reduce the polynomial running time.]

[1]but not really in practice