CS 4820, Summer 2010 Out: July 20. **Due: Friday, July 23, 8:30am**

Homework 5

Reading: Section 7.1, 7.2, 7.5–6, 7.10–12.

Since Network Flow has a lot of notation, and we won't have time for another homework before prelim, I am giving some practice problems that you might want to try in order to understand network flows better. The problems marked "practice problems" should not be submitted as homework: they are only for your own benefit.

Practice Problem (Please Don't Submit). Decide whether you think the following statements are true or false. If true, give a short explanation. If false, give a counterexample.

(a) Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity $c(a)$ on every arc a. If f is a maximum s-t flow in G, then f saturates every arc out of s with flow. (I.e. for all arcs a out of s, we have $f(a) = c(a)$.)

(b) Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity $c(a)$ on every arc a; and let (S, T) be a minimum s-t cut with respect to these capacities. Now suppose we add 1 to every capacity; then (S, T) is still a minimum s-t cut with respect to these new capacities $\{1 + c(a) : a \in A\}$.

Practice Problem (Please Don't Submit). Solve Problem 3 at the end of Chapter 7 in Kleinberg Tardos book.

Practice Problem (Please Don't Submit). Let f be a flow in a network $G = (V, A)$, and let α be a real number. The scalar flow αf is a function from $A \to \mathbb{R}$ defined by $(\alpha f)(a) = \alpha \cdot f(a)$.

Prove that the set of s-t flows in a network is a convex set by showing that if f and g are s-t flows, then for any $\alpha \in [0, 1]$, the function $h = \alpha f + (1 - \alpha)g$ is also a flow. In addition, prove that $\text{value}(h) = \alpha \cdot \text{value}(f) + (1 - \alpha) \cdot \text{value}(g)$.

Problem 1. The Ford Fulkerson algorithm solves the problem of finding a minimum cut in a directed as well as undirected graph which separates two given nodes s and t. But we want to do something different: we want to find a global minimum cut, that is a minimum cut that disconnects at least one pair of vertices. It does not matter which pair of vertices is separated. One possible solution for this problem is to run the Ford-Fulkerson algorithm $n(n-1)$ times for all the (s, t) pairs, and among these solutions take the smallest one. You goal is to find a better algorithm.

(a) Let $G = (V, E)$ be an *undirected* graph. Show that you can use the Ford-Fulkerson algorithm only $n - 1$ times to find a global minimum cut in this undirected graph.

Bonus problem: Here is a bonus problem (you don't need to do it to finish the homework, but you will receive extra points if you do). Let $D = (V, A)$ be a *directed* graph. Show that you can use the Ford-Fulkerson algorithm only n times to find a global minimum cut in this directed graph D.

Problem 2. Consider the task of tiling a chessboard (8×8 squares) with dominoes which are the size of two squares. That is, placing dominoes on the board such that every domino covers exactly two squares of the chessboard and every square on the chessboard is covered by exactly one domino. Dominoes may be placed horizontally (1×2) or vertically (2×1). This is clearly not hard to do, and in fact, it is pretty clear that there are many many ways to do this.

Now suppose you remove two opposite corners from the chessboard. Can this new area still be tiled with dominoes? It turns out that the answer is "no". After messing around with dominoes for a few minutes you may begin to suspect that it can't be done. But the easy way to see that it is impossible is to notice that the two opposite corner squares of a chessboard are always of the same color. If we consider the original black and white coloring on the chessboard, then the new area has 32 squares of one color and 30 squares of the other color. But every domino always covers exactly 1 white square and 1 black square. So any tiling will always leave at least 2 squares uncovered.

(a) One might ask whether this sort of problem is the only sort of problem preventing you from tiling a region. More precisely, is it the case that for any $n \times n$ chess board where n is even, removing an equal number of white squares and black squares leaves a region that can be tiled with dominoes? Give a proof or a counterexample.

(b) Give an efficient algorithm that, given a region (a subset of squares from an $n \times n$ chessboard), decides whether that region can be tiled with dominoes.

(c) Suppose the answer for a particular input is "no". Show how your algorithm can be used to provide a simple proof of this fact. This proof should be a simple one-line statement that you could give to your *grandmother* and have her *understand* why tiling would be impossible. In other words, your proof (that tiling cannot be done for a particular input instance) shouldn't require any knowledge of flows.

Problem 3. Some of your friends have recently graduated and started a small company, which they are currently running out of their parents' garages in Lansing. They're in the process of porting all their software from an old system to a new, revved-up system; and they're facing the following problem.

They have a collection of n software applications, $\{1, 2, \ldots, n\}$, running on their old system; and they'd like to port some of these to the new system. If they move application i to the new system, they expect a net (monetary) benefit of $b_i > 0$. The different software applications interact with one another; if applications i and j have extensive interaction, then the company will incur an expense if they move one of i or j to the new system but not both — let's denote this expense by $x_{i,j} > 0$.

So if the situation were really this simple, your friends would just port all n applications, achieving a total benefit of $\sum_{i=1}^{n} b_i$. Unfortunately, there's a problem (as there always is!).

Due to small but fundamental incompatibilities between the two systems, there's no way to port application 1 to the new system; it will have to remain on the old system. Nevertheless, it might still pay off to port some of the other applications, accruing the associated benefit and incurring the expense of the interaction between applications on different systems.

So this is the question they pose to you: which of the remaining applications, if any, should be moved? Give a polynomial-time algorithm to find a set $S \subseteq \{2, 3, \ldots, n\}$ for which the sum of the benefits minus the expenses of moving the applications in S to the new system is maximized.