# CS369N: Problem Set #2

Due in class on Tuesday, November 8, 2011

**Instructions:** Same as the first homework.

## Problem 6

(20 points) This problem is about parameterized running time analysis with respect to an input property.

Recall that in the Steiner tree problem you are given an undirected graph with costs on edges, $k$ vertices $t_1, \ldots, t_k$ are distinguished *terminals*, and the goal is to compute the minimum cost subgraph that spans all of the terminals (and possibly other vertices as well, if needed). This problem is NP-hard in general.

Give an algorithm that solves the Steiner tree problem correctly in every graph and, morover, runs in time $O(f(t) \cdot g(n, k))$, where $f$ is an arbitrary function, $g$ is a polynomial function, and $t$ denotes the *treewidth* of the input graph. You can use without proof any of the definitions, theorems, and algorithms from Sections 10.4 and 10.5 of the book *Algorithm Design*, by Kleinberg and Tardos. (Copies of the book are on reserve, for my CS161 class, at the Engineering library.)

## Problem 7

Recall from Lecture #7 the model of online paging with access graphs.

(a) (4 points) Consider the online paging problem in which the access graph $G$ is a line on $k+1$ pages. (Recall that in every legal input, every page must be a neighbor of the previous one; the first page can be arbitrary.) Recall from the lecture notes that LRU has competitive ratio 1 on this graph. Show that the competitive ratio on $G$ of the FIFO paging algorithm is $\Omega(k)$, where $k$ is the size of the cache.

(b) (6 points) Consider the online paging problem in which the access graph $G$ is a cycle on $k+1$ pages. Recall from the lecture notes that LRU has competitive ratio $k$ on this graph. Give an online algorithm that has competitive ratio $O(\log k)$ for $G$.

(c) (5 points) Suppose we change the cost model so that every cache hit costs 1 and every cache miss costs $m \geq 1$. Recall from Lecture #3 how we break a sequence $\sigma$ into blocks $\sigma_1, \sigma_2, \ldots, \sigma_b$, where each $\sigma_i$ is a maximal sequence in which only $k$ distinct pages are requested. A simplistic and sequence-dependent model of locality is as follows: for a given $\sigma$, define its locality $L(\sigma)$ as its average block length (the number of requests divided by $b$). Note that this does make at least some intuitive sense as a locality measure. Prove that for every sequence $\sigma$ with locality at least $\alpha m$, the cost (in this new cost model) of LRU is at most $1 + \frac{k-1}{\alpha+1}$ times that of the optimal (furthest-in-future) algorithm.

(d) (Extra credit) Formulate a definition of a "graph-independent" algorithm (like FIFO or LRU but unlike the one you presumably designed in part (b)). Say whatever you can (examples, lemmas, conjectures) about whether or not there are better graph-independent algorithms than LRU.

## Problem 8

(15 points) Recall from Lecture #6 that randomly ordering the vertices of a graph $G = (V, E)$ with vertex weights $w$ and then running a simple greedy algorithm produces an independent set with expected value at

least $\sum_{v \in V} \frac{w_v}{deg(v)+1}$. (Just analyze one vertex and then use linearity of expectations.) Give a deterministic algorithm with the same guarantee.

One approach to this problem is to derandomize the randomized algorithm above. Alternatively, you can prove the guarantee directly for a suitable deterministic greedy algorithm.

## Problem 9

(15 points) Recall that in the *Vertex Cover* problem, you are given an undirected graph $G = (V, E)$ where each vertex has a nonnegative weight $w_v$. The goal is to compute the subset $S$ of $V$ of minimum total weight with the property that every edge has at least one of its endpoints in $S$.

Call a Vertex Cover instance $\gamma$-*stable* if its optimal solution $S^*$ remains optimal even after each vertex $v$ is scaled by an arbitrary factor $\sigma_v \in [1, \gamma]$. Prove that in $\Delta$-stable Vertex Cover instances, the optimal solution can be recovered in polynomial time. (Here $\Delta$ denotes the maximum degree of the graph.)

## Problem 10

(20 points) This problem is about resource augmentation, as discussed in Lectures #3 and #4. The biggest "killer application" for this technique has been in scheduling problems. First, familiarize yourself with the notes from Lecture #8 of the 2009 version of this course (listed on the Web page for this year under the third lecture).

Recall the Balance algorithm for non-clairvoyant online scheduling described in those notes. There, we studied the objective of minimizing the average flow (or response) time, $\sum_j (C_j - r_j)$. One concern about such objectives is that minimizing the average might require assigning huge delays to a small number of jobs. This problem proves that this concern is unwarranted for the Balance algorithm.

Precisely, consider the objective of minimizing the maximum idle time of a job, where the idle time is $C_j - r_j - (p_j/s)$, where $C_j$ is the job's completion time, $r_j$ is its release date, $p_j$ is its processing time, and $s$ is the machine speed. Show that the maximum idle time of a job under the Balance algorithm with a machine of speed $1 + \epsilon$ is at most $1/\epsilon$ times that of an optimal (clairvoyant and offline) solution with a machine of unit speed.