Question 1. (20 points) Suppose you are given a rooted n-node tree T: For each node v you have a linked list $L[v]$ containing the children of that node (only the immediate children of v, not all its descendants). Give an $O(n)$ time algorithm that, when given such a T as input, computes a largest subset S of the vertices such that no two vertices in S have a parent-child relationship.

Question 2. (20 points) Let G be an undirected graph whose set of vertices is V and whose set of edges is E. Let $\hat{V}$ be a minim*um* subset of V such that every edge in E has at least one endpoint in $\hat{V}$; that is, $\hat{V}$ is a smallest subset of the vertices that satisfies the requirement that every edge has at least one endpoint in $\hat{V}$.

Consider the following greedy algorithm for computing a subset S of the vertices.

1. Initialize S to be empty
2. Go through the edges and, for each edge (u, v) in turn, include u and v in S if *neither* u *nor* v occur in S (if either of them is in S, then S stays the same).

Prove that $|S| \leq c|\hat{V}|$ for some constant integer $c > 1$. That is, state what c is, and prove the inequality.

Question 3. (30 points) Let $S = \{1, 2, \ldots, m\}$ be a set of m machines, and let $J = \{J_1, \ldots, J_n\}$ be a set of n jobs each of which has to be done by one of the machines. No more than one machine can work on the same job, and a machine cannot work on more than one job. Moreover, with each job J_k is associated a contiguous interval $[l_k, r_k]$ of integers, $1 \leq l_k \leq r_k \leq m$, whose significance is that only machines i for which $l_k \leq i \leq r_k$ can work on job J_k. Give an $O(mn)$ time algorithm for assigning machines to jobs so as to maximize the number of busy machines (which is the same as the number of jobs being done). There is no need to prove the correctness of your algorithm.

Note. An algorithm that takes sub-quadratic time is possible, but you are not responsible for it (it consists of a more clever implementation of the greedy idea that you are asked to design in this question).

Question 4. (30 points) Give an $O(n)$ time algorithm for the weighted version of Question 1, i.e., when every vertex v has a positive weight $w[v]$ associated with it and the value of S is the sum of the weights of its vertices (rather than how many vertices are in S). Unlike question 1, here it is enough to compute the weight of the best S (without the S that achieves that weight). Do this using dynamic programming: Compute the two vectors *With* and *Without* where *With*$[v]$ and *Without*$[v]$ are defined relative to the subtree of v (call it T_v): *With*$[v]$ (respectively, *Without*$[v]$) is the weight of the best solution in T_v that is constrained to contain (respectively, not contain) v.

Date due: October 4, 2012