Question 1. (10+5 points) Let T be a tree whose 16 nodes are named $a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p$. A preorder traversal of T lists its nodes in the following order:

$$d\ g\ c\ m\ l\ f\ b\ p\ j\ k\ a\ e\ o\ h\ n\ i$$

A postorder traversal of T lists its nodes in the following order:

$$m\ c\ l\ g\ p\ j\ b\ k\ o\ h\ n\ e\ i\ a\ f\ d$$

Draw T, then give a listing of its nodes according to an inorder traversal.

Question 2. (10 points) Let T be an n-node tree. For each node v in T, let $Desc(v)$ be the number of descendants of v, i.e., the number of nodes in the subtree of v (counting v as being in its own subtree). For example, if v is the root then $Desc(v) = n$, and if v is a leaf then $Desc(v) = 1$. The $Desc$ information can easily be computed during a postorder traversal of T: At the time of giving a node v its postorder number, you increment $Desc(v)$ by 1 and then add $Desc(v)$ to $Desc(Parent(v))$. Assume that the $Desc$ array is already given to you. You are also given the preorder numbers of all the nodes: For every node v you know $Preorder(v)$. This question is about how to use the $Desc$ and $Preorder$ information to quickly determine, for any pair of vertices, their ancestral relationship to each other in T (or the lack of such a relationship):

- Explain how, using only the $Desc$ and $Preorder$ information, you can determine in constant time (i.e., in time that does not depend on the height of the tree or its number of nodes) whether two nodes v and w are related by the "ancestor" relationship or not; if one is ancestor of the other, you have to determine which is ancestor, and if neither is ancestor you have to determine which one is to the left of the other in T.

Question 3. (10 points). Let T and $Desc$ be as in the previous question, except that for the present question we assume that T is binary (i.e., every node has zero, one, or two children) and that its number of nodes n is a multiple of 3. Consider the following algorithm for finding a node v in T such that $n/3 \leq Desc[v] \leq 2n/3$.
The procedure, $FindNode(v)$, is initially called with $v =$ the root:

- If $Desc[v] \leq 2n/3$ then return v as the answer, otherwise recursively call $FindNode(w)$ for $w =$ the child of v that has the larger $Desc$ value among the (at most two) children of v.

Is this algorithm correct? Give a counterexample if your answer is "No", a proof of correctness if your answer is "Yes". (Note that the algorithm assumes that it can always return some node as answer, and has no provision for a "Not Found" answer.)

Question 4. (15 points). Let T and $Desc$ be as in Question 2 (hence T is not necessarily binary). For this question, a node of T does not necessarily have constant degree, e.g., it

might have $n/10$ children. By "removing node v from T" we mean removing node v and all the edges that touch v (the edges that link v to its children, but also the edge that links v to its parent). Note that removing from T a node v that has k children breaks the tree into k pieces if v is the root of T, into $k+1$ pieces if v is not the root of T. Consider the following algorithm for finding a node whose removal from T breaks the tree into pieces none of which has more than $n/2$ nodes.
The procedure, $FindNode(v)$, is initially called with $v =$ the root:

- If every child w of v has $Desc(w) \leq n/2$ then return v as answer. Otherwise recursively call $FindNode(w)$ for $w =$ the child of v that has the largest $Desc(w)$ value.

Is this algorithm correct? Give a counterexample if your answer is "No", a proof of correctness if your answer is "Yes". (Note that the algorithm assumes that it can always return some node as answer, and has no provision for a "Not Found" answer.)

Date due: April 21, 2011 at the beginning of class.