The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual
Chapter 2, Problem 2.25

We can provide a purely constructive solution to this problem, with a linear-time recursive algorithm based on depth-first search of the tree.

```
/* n is the total number of nodes of the tree;
   it may already be known; if not, it can be computed in linear time
   with any tree traversal */
/* a is any node of the tree -- it need not be a root or a leaf */
/* the main procedure just calls the recursive subroutine with some
   initialization; the recursive subroutine basically computes the size
   of each subtree, but keeps checking whether the collection of subtrees
   of a node number among them at least n/2 nodes, in which case
   that node is the desired answer and the recursive routine aborts */

main procedure
  begin
    check(a)
  99:
  end

function check(t: tree)
  /* on normal return, the value of the function is the number of nodes
     in the subtree rooted at t */
  begin
    count := 0
    for all (non-nil) children c of t do
        count := count + check(c)
    if count > n/2 go to 99 /* abort the recursion: t is the answer */
    check := count + 1   /* to account for t itself */
  end
```

Correctness is fairly obvious. Termination by the `goto` must occur, since the value of `count` keeps increasing. At termination, the node `t` being explored by the dfs has a collection of dfs children that together include at least `n/2` nodes; but none of its children has that property, because none caused termination (we had a normal return from each of them). Thus none of the dfs subtrees of `t` has more than `n/2` nodes; but the dfs subtree rooted at `t` has at least `n/2` nodes, so that the rest of the tree (which can be viewed as a subtree of `t`, but hanging "up" from it, since its first member, if any, is the parent of `t` in the dfs search), if any, cannot have more than `n/2` nodes. Hence `t` is the desired answer. The dfs search artificially ordered the tree, by assigning it a root `a` and ordering all nodes "downwards" from there; but the ordering does not matter, since any node can cause termination and since the termination condition ensures that any subtree connected to the termination node has at most `n/2` nodes in it—whether that subtree is one of the dfs subtrees or the subtree that would be created by removing the node and its dfs descendants. In particular, note that, if `a` is the answer, it will correctly be selected by the routine, since none of its subtrees will be large enough to cause termination.