Homework 2, additional problems, solution set

1. We use the procedures ToHOne(n, A) and ToHTwo(n, A) to respectively move the n rings from the start pole (A) one position to the right (i.e. to pole B), and two positions to the right (i.e. to pole C).

Here are the two procedures.

ToHTwo(n,A)
  ToHTwo($n-1, A$) (* moves the top $n-1$ rings to pole C *)
  move ring n to post B;
  ToHOne($n-1, C$); (* moves the top $n-1$ rings to pole A *)
  move ring n to post C;
  ToHTwo($n-1, A$) (* moves the top $n-1$ rings to pole C *)

ToHOne(n,A)
  ToHTwo($n-1, A$) (* moves the top $n-1$ rings to pole C *)
  move ring n to post B
  ToHTwo($n-1, B$) (* moves the top $n-1$ rings to pole B *)

The base cases simply omit the recursive calls.

The inductive assertion is that ToHOne(n, A) moves all n rings from pole A to pole B, while performing only single clockwise moves, and that ToHTwo(n, A) moves all n rings from pole A to pole C, while performing only single clockwise moves. Note that we interpret pole A as being a single clockwise move from pole C.

2. If v is a leaf or $v.clr$ = Red then $v.allb = 0$. Otherwise, the length of a longest all-blue path to a leaf in the subtree rooted at v's blue child w is $1 + w.allb$. This yields the following recursive solution.

Lgstb(v):
$v.allb \leftarrow 0$;
for each child w of v **do**
    Lgstb(w);
    if ($v.clr$ = Blue **and** $w.clr$ = Blue) **then** $v.allb \leftarrow \max\{v.allb, 1 + w.allb\}$
    end if
end for

3. In the first pass, for each node v, we will determine the next node on a longest path descending from v and store it in $v.nxt$. We we also store the length of this longest path in $v.lgst$.

In the second pass, we perform a root to leaf traversal along the longest path using the values in $v.nxt$ to guide the traversal.

The code follows.

Pass 1
LgPth(v)
$v.lgst \leftarrow 0$; $v.nxt \leftarrow$ **nil**;
for each child w of v **do**
    LgPth(w);
    if $1 + w.lgst > v.lgst$ **then** $v.lgst \leftarrow 1 + w.lgst$; $v.nxt \leftarrow w$
    end if
end for

Pass 2
PrtPth(v)
Print(v);
if v is not a leaf **then** PrtPth($v.nxt$)
end if

This question has demonstrated the important technique of path recovery.

4. We need to know the length of the longest path in the subtree rooted at v, and the length of the longest path descending from v, and the leaf endpoints of these paths. We store the lengths in $v.lp$ and $v.lpd$, respectively, and the leaves in $v.lpf$, $v.lps$, and $v.lpdlf$, respectively.

This leads to the following recursive code.

LgstEnds(v)
$v.lpd \leftarrow 0$; $v.lp \leftarrow 0$; $v.lpf \leftarrow$ **nil**; $v.lps \leftarrow$ **nil**; $v.lpdlf \leftarrow v$
for each child w of v **do**
    LgstEnds(w);
    if $w.lp > \max\{v.lp, v.lpd + w.lpd + 1\}$ **then** $v.lp \leftarrow w.lp$; $v.lpf \leftarrow w.lpf$;
        $v.lps \leftarrow w.lps$
    else if $v.lpd + w.lpd + 1 > v.lpd$ **then** $v.lp \leftarrow v.lpd + w.lpd + 1$; $v.lpf \leftarrow v.lpdlf$;
        $v.lps \leftarrow w.lpdlf$
    end if
    if $w.lpd + 1 > v.lpd$ **then** $v.lpd \leftarrow w.lpd + 1$; $v.lpdlf \leftarrow w.lpdlf$
    end if
end for

Note that this code merely adds leaf updates to the code given in the lecture notes, as well as breaking the max in the earlier code into multiple cases.