Homework 2, Solution set

1. R_2 moves to Pole B.
For to allow R_5 to move to Pole B, all the other rings need to move to Pole C, with R_4 at the bottom. To allow R_4 to move to Pole C, all the smaller rings need to move to Pole A, with R_3 at the bottom of these rings. To allow R_3 to move to Pole A, all the smaller rings need to move to Pole B, with R_2 at the bottom of these rings. But there is nothing blocking this move, so this is the first move.

2. For the case $n = 1$, we move the one ring from post A to post B to post C, thereby traversing each configuration once.

Our recursive solution MaxToH(n, A, B, C), interleaves this with calls to MaxToH$(n-1, X, Y, Z)$ as follows.

1: Input: all n rings on post A
2: MaxToH$(n-1, A, B, C)$
3: move ring n to post B
4: MaxToH$(n-1, C, B, A)$;
5: move ring n to post C
6: MaxToH$(n-1, A, B, C)$;

Now we argue that all configurations are visisted. The inductive hypothesis ensures that in Step 2 all configurations with ring n on post A are traversed and no others. Similarly, in Step 4 all configurations with ring n on post B are traversed and no other, and in Step 6 all configurations with ring n on post C are traversed and no others. Thus, over the 6 steps, all configurations of the n rings are traversed, as desired.

3. We will need to compute the number of paths from v to a descendant leaf, as well as the total length of these paths.

Let $v.cnt$ store the number of paths from v to a descendant leaf. Let $v.tl$ store the total length of all such paths. Let us look at the base case. When v is a leaf, $v.cnt = 1$, $v.tl = 0$.

When v is not a leaf, any path from v to a leaf will go via one of its children w. We compute $w.cnt$ and $w.tl$ recursively. Clearly, the total length of all the paths from v to a leaf in the subtree rooted at w is $w.tl + w.cnt$. In other words, we have the following recursive solution.

TotLen(v):
$v.tl \leftarrow 0$;
if v is a leaf **then** $v.cnt \leftarrow 1$
else $v.cnt \leftarrow 0$
end if
for each child w of v **do**
    TotLen(w);
    $v.tl \leftarrow v.tl + w.tl + w.cnt$;
    $v.cnt \leftarrow v.cnt + w.cnt$
end for

4. This is similar to the longest path problem considered in class except that now there is the extra constraint that the path be all blue. For each vertex v, we will need to compute two values: the length of the longest all-blue path descending from v, and the length of the longest all-blue path in the subtree rooted at v. These values will be stored in $v.lbp$ and $v.allb$, respectively.

If v is red or a leaf node then $v.lbp = 0$. Otherwise, if $v.lbp$ holds the value of this variable when considering the tree formed from v's first i subtrees, then the value when including the $i+1$st subtree rooted at node w will be unchanged if w is red and otherwise it will be $\max\{v.lbp, 1 + w.lbp\}$.

Turning to $v.allb$, if v is a leaf, $v.allb = 0$. Otherwise, if $v.allb$ and $v.lbp$ hold the values of these variables when considering the tree formed from v's first i subtrees, then the value of $v.allb$ when including the $i+1$st subtree rooted at node w is determined as follows: if v or w are red the value will be $\max\{v.allb, w.allb\}$, and if both v and w are blue the value will be $\max\{v.allb, w.allb, v.lbp + 1 + w.lbp\}$.

The code follows.

LBP(v):
$v.allb \leftarrow 0$; $v.lbp \leftarrow 0$;
for each child w of v **do**
  LBP(w);
  if ($w.clr = \text{RED}$ **or** $v.clr = \text{RED}$) **then** $v.allb \leftarrow \max\{v.allb, w.allb\}$
  else (* $w.clr = \text{BLUE}$ **and** $v.clr = \text{BLUE}$ *) $v.allb \leftarrow \max\{v.allb, v.lbp{+}1{+}w.lbp, w.allb\}$
  end if;
  if ($v.clr = \text{BLUE}$ **and** $w.clr = \text{BLUE}$) **then** $v.lbp \leftarrow \max\{v.lbp, 1 + w.lbp\}$
  end if
end for