CPS 130 Homework 12 - Solutions

1. (CLRS 13.1-5) Show that the longest simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

   Solution: From the red-black properties, we have that every simple path from node x to a descendant leaf has the same number of black nodes and that red nodes do not occur immediately next to each other on such paths. Then the shortest possible simple path from node x to a descendant leaf will have all black nodes, and the longest possible simple path will have alternating black and red nodes. Since the leaf must be black, there are at most the same number of red nodes as black nodes on the path.

2. (CLRS 13.1-6) What is the largest possible number of internal nodes in a red-black tree with black-height k? What is the smallest possible number?

   Solution: The largest possible number of internal nodes in a red-black tree with black-height k is $2^{2k} - 1$. The smallest possible number is $2^k - 1$.

3. (CLRS 13.3-2) Show the red-black trees that result after successively inserting the keys $41, 38, 31, 12, 19, 8$ into an initially empty red-black tree.

   Solution:

4. (CLRS 13.4-3) Use the red-black tree resulting from the previous problem. Show the red-black trees that result from the successive deletion of the keys in order $8, 12, 19, 31, 38, 41$.

   Solution:

5. (CLRS 13-2) The **join** operation takes two dynamic sets S_1 and S_2 and an element x such that for any $x_1 \in S_1$ and $x_2 \in S_2$, we have $key[x_1] \leq key[x] \leq key[x_2]$. It returns a set $S = S_1 \cup \{x\} \cup S_2$. In this problem, we investigate how to implement the join operation on red-black trees.

   (a) Given a red-black tree T, we store its black-height as the field $bh[T]$. Argue that this field can be maintained by RB-INSERT and RB-DELETE without requiring extra storage in the tree and without increasing the asymptotic running times. Show while descending through T, we can determine the black-height of each node we visit in $O(1)$ time per node visited.

   Solution: For an empty red-black tree we initialize its bh to 0. Insertions can only increases the black height and deletions can only decrease the black height. During an insertion if rebalancing goes up to the root and the root becomes red and then painted black, we increase bh by 1. This is the only place where black nodes are added in the tree. During a deletion, if the extra black goes up to the root then we decrease bh by one. This is the only place where the black nodes are removed from the tree. Thus bh is maintained with only $O(1)$ cost per operation. When going down the tree we can determine the bh of a node visited simply by starting with bh

at the root and subtracting 1 for each black node visited along the path. This is done in $O(1)$ per node.

We wish to implement the operation RB-JOIN(T_1, x, T_2) which destroys T_1 and T_2 and returns a red-black tree $T = T_1 \cup \{x\} \cup T_2$. Let n be the total number of nodes in T_1 and T_2.

(b) Assume without loss of generality that $bh[T_1] \geq bh[T_2]$. Describe an $O(\lg n)$ time algorithm that finds a black node y in T_1 with the largest key from among those nodes whose black-height is $bh[T_2]$.

Solution: The idea is to move down in T_1 following always right links (or a left only if a right link does not exist), subtracting 1 from bh (initially a copy of $bh[T_1]$) for each black node encountered until $bh = bh[T_2]$. Then y is the first black node encountered after the condition above is met (at most 2 hops away from where the condition is met). Since $bh[T_1] \geq bh[T_2]$ and there are no discontinuities in the black height, such a node will be eventually found. In the worst case ($bh[T_2] = 1$), y will be found after at most $O(\lg n)$ steps.

(c) Let T_y be the subtree rooted at y. Describe how T_y can be replaced by $T_y \cup \{x\} \cup T_2$ in $O(1)$ time without destroying the binary-search-tree property.

Solution: We create a new subtree T_x with root x, left child T_y, right child T_2 and we attach it to the parent of y. Since both y and the root of T_2 are black, we paint x red (see also next question). The binary-search-tree property holds since all elements in T_y are less than or equal to x and all in T_2 are greater than or equal to x. Moreover the overall black height of T_1 is not affected; T_y and T_2 have the same black height and x is painted red. The operation takes $O(1)$.

(d) What color should we make x so that red-black properties $1, 2,$ and 4 are maintained? Describe how property 3 can be enforced in $O(\lg n)$ time.

Solution: As mentioned above, x must be painted red. Then restructuring operations can be applied in case y's parent was red. Changes will propagate at most up to the root, thus giving a time upper bound of $O(\lg n)$.

(e) Argue that the running time of RB-JOIN is $O(\lg n)$.

Solution: Summing up the costs of the previous steps, the total time requirement is $O(\lg n)$.