1 Review

1.1 Binary Search Trees

Problem: Figure out the *rank* of a node in $O(\log n)$ time. You may use augmentation. The rank of a node is its position in a sorted list of the nodes.

Solution: We keep the number of children in a node's right and left subtree (denoted as $R(n)$ and $L(n)$) as an augmentation. To find the rank of a node x, we begin with $r = 0$. We begin at the root node and traverse the tree to find x. Every time we go right at a node n, we add $r = r + L(n) + 1$. When we find x, we finally add $r = r + L(x)$. The number r is the rank of x.

Augmentation: When inserting a node, we add 1 to a node's right subtree if we go right and 1 to its left subtree if we go left. Similarly with deleting except we subtract 1. For a left rotation, let x be the element around which we are rotating and y be the new root. Then the number of children in x's left subtree remains unchanged and the number of children in its right subtree is the number is $L(y)$. The number of children in y's right subtree remains unchanged and the number of children in its left subtree is $1 + L(x) + R(x)$ (where $L(x)$ and $R(x)$ are after rotating x).

Extra point: We cannot keep a node's rank as an augmentation because it cannot be kept updated in $O(\log n)$ time through inserts and deletes. For example, inserting the smallest number into the tree would require $O(n)$ time to update the ranks of every other number.

Problem: *Select* the rth largest number from a balanced binary search tree in $O(\log n)$ time.

Solution: Use the same augmentation. Let $k = r$. When $L(n) > k$, search recursively for rank k in left subtree. Otherwise, search recursively for rank $k - L(n)$ in right subtree.

Problem 2d from PSet2: Find the smallest node larger than t such that the node's successor is at least 6 greater.

Solution: Store the number of gaps at least 6 in left and right subtrees. Now find the successor of t in the tree. If this successor has a gap, return it. Otherwise, if it has a gap in its right tree, recursively search for the smallest number with a gap in the right tree. Otherwise, find the first node n such that n is a left child and has a gap or has a gap in its right subtree. If n has a gap in its right subtree, search that subtree for the smallest node with a gap.

Why does this work? Let the x-successor of t be the node that is x nodes larger than t. So the 1-successor is the successor of t and the 2-successor is the successor of t's successor etc. We want to find the smallest x such that the x-successor of t has a gap. Now the successor of a node is in its right subtree if it has one or one of its ancestors if it does not. Therefore, if a node n does not have a gap in its right subtree, the successor of the largest node in its right subtree is the first ancestor of n that is a left child. Therefore, we check all x-successors until we find the first one with a gap.

1.2 Recursions

$T(n) = T(n/3) + T(2n/3) + O(n)$.

Make a tree and solve it that way. Approximately $\log n$ levels, n work per level.

Can also do the Master Theorem examples from last week's notes.

1.3 Hash Tables

Problem: We insert 4 items with keys k_1, k_2, k_3, and k_4 into a hash table of size m using linear probing and assuming simple uniform hashing. What is the probability that inserting the fourth element requires at least three probes?

Solution: We must have at least two elements adjacent to each other and k_4 must hash to the top one. There are several possible cases:

1. k_2 is above k_1 $(1/m)$, k_3 is anywhere (1), k_4 collides with k_2 $(1/m)$: $1/m^2$
2. k_2 is below k_1 ($2/m$ since k_2 could hash to k_1 *or* just hash to one below k_1), k_3 is anywhere (1), k_4 collides with k_1 $(1/m)$: $2/m^2$
3. etc. Actually these start to get pretty complicated. I only did the first two or three cases.

Problem: Assuming simple uniform hashing with chaining, after n insertions to a table of size m, what is the probability the first slot has a chain of at least size 1?

Solution: The probability that the ith key did not hash to the first slot is $(m-1)/m$. Therefore the probability that none of the keys hashed to the first slot is $((m-1)/m)^n$. Thus, the probability that at least one key hashed to the first slot is $1-((m-1)/m)^n$.

1.4 Heaps

Question 4b from Spring 2008 exam. There is another solution that doesn't use heaps, but the heap solution is educational.