Intro to Algorithms, COMP-160, Homework #8

Benjamin Tanen, 03/31/2016

2. Consider a BST with n distinct values and height h. Let x be a value in this BST.

   For both parts, provide an algorithm and analyze its time complexity. In other words, justify your answers.

   (a) **Assume that x is not the largest value in the BST. How fast can you find the smallest value larger than x in the BST, in the worst case.**

   In order to find the smallest value y that is greater than our known value x, we should go the right child node of x and then left as much as possible (until we hit a node with no left child). If the node with value x doesn't have a right child, the smallest value y will be at the parent node.

   In a BST of height h where we are looking for the smallest value y that is larger than our original value x, the worst case comes if x is at the root of our tree and our tree is complete. This is because we are ultimately looking for the left-most value to the right of x. Thus if x is the root of a completed tree, the path between x and y is maximized. Since this path is maximized in a complete tree, we know the length of this path is equal to our height h. Thus, it takes $O(h)$ to traverse the tree and find value $y =$ the smallest value larger than x.

   (b) **Assume that the BST has at least k values larger than x. How fast can you find the k smallest values larger than x, in the worst case? *The answer is not simply k times the answer for (a).***

   The algorithm that we will use for this problem first begins at a node with value x. From this, we will be in one of two cases:

   i. If we are at a node with a right child, go to the right once and then go to the left as much as possible (until we hit a node with no left child).
   ii. If we are at a node with no right child, bounce up to the parent node.

   After doing one of these options, mark the node we are currently at and repeat until we have marked k nodes. These will be the k smallest values larger than x.

   Next, we should consider the complexity of this algorithm in the worst case. In any tree / case, we will need to traverse through some number of subtrees of our whole BST where subtree t_i has a height h_i and has n_i nodes that we mark from it. We can also describe the total nodes in the subtree as a fraction of the total number of nodes k that we need to mark, so $n_i = \frac{c_i}{k}$ where c_i is some integer less than or equal to k. Since we do work to traverse down each subtree, taking a total amount of h_i work, and when we mark and bubble up the tree, coming to $n_i = \frac{c_i}{k}$ work (bubbling up for each of the n_i nodes we mark), the total work for each subtree t_i is $h_i + \frac{c_i}{k}$.

   Since we know we will always traverse a constant number of subtrees in the worst case BST (let's say j is this constant), we can describe the total work for finding the k smallest values larger than some value x as:

$$\sum_{i=1}^{j} h_i + \frac{c_i}{k} = \Theta(h) + \Theta(k) = \Theta(h + k)$$