CMPS101: Homework #4 Solutions

TA: Krishna (vrk@soe.ucsc.edu)

Due Date: March 5, 2013

12.1-5

Problem

Argue that since sorting n elements takes $\boldsymbol{\Omega(nlgn)}$ time in the worst case in the comparison model, any comparison-based algorithm for constructing a binary search tree from an arbitrary list of n elements takes $\boldsymbol{\Omega(nlgn)}$ time in the worst case.

Solution

The value of the nodes in the tree can be printed in sorted order in O(n) time using an inorder traversal of the tree. Thus any algorithm that builds a binary tree can be used to solve a sorting problem. Now if it were possible to devise an algorithm that can construct a binary tree with a worst case time bound better than $\boldsymbol{\Omega(n\ logn)}$, then we would have a sorting algorithm that has better bound than $\boldsymbol{\Omega(n\ logn)}$. Since binary tree construction also uses key comparisons, the $\boldsymbol{\Omega(nlgn)}$ bounds must apply to such an algorithm too. The existance of such an algorithm for constructing a binary tree in o(n lg n) time would thus contradict the lower bound for sorting.

12.2-4

Problem

Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys $\boldsymbol{a \in A, b \in B}$,and $\boldsymbol{c \in C}$ must satisfy $\boldsymbol{a \leq b \leq c}$. Give a smallest possible counterexample to the professors claim.

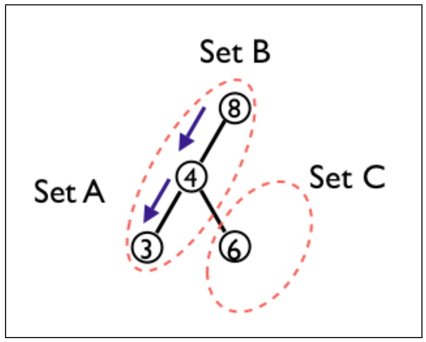

Figure 1: Counter example to Professor Bunyan's claim. The set A is empty. The search path, where search is performed for the key 3 is marked. The search proceeds from root 8 to the node 4 and then to node 3. So $\boldsymbol{B = \{8, 4, 3\}}$. Set C is the only key to the right of the path, i.e., $\boldsymbol{C = \{6\}}$.

Solution

The claim is wrong. A simple counter example is shown in figure 1. In the figure, the search is being done for leaf node 3, so the set $\boldsymbol{B = \{8, 4, 3\}}$. There is nothing to the left of the path and so set $\boldsymbol{A = \{\phi\}}$. Set C is all elements to the right of the path, so set $\boldsymbol{C = \{6\}}$. For any element $\boldsymbol{a \in A}$, and $\boldsymbol{b \in B}$ the claim is true, since A is an empty set. But if set $\boldsymbol{b = 8}$ and $\boldsymbol{c = 6}$, the claim fails to hold.

12.3-3

Problem

We can sort a given set of n numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

Solution

```
Tree-Sort(A)
   // let T be an empty binary search tree
   for i <- 1 to n
      do  Tree-Insert(T, A[i])
   Inorder-Tree-Walk(root[T])
```

Worst case of $\boldsymbol{\Theta(n^2)}$ occurs when a linear chain of nodes results from the repeated insert operations. It can be easily verified that this follows from the

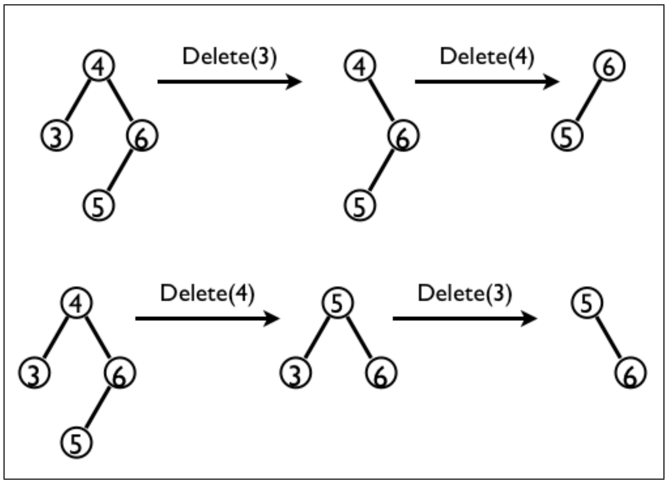

Figure 2: Example demonstrating that bst delete operation is not commutative.

following recurrence. $\boldsymbol{T(n) = T(n-1) + cn}$, i.e., to insert $\boldsymbol{n}$ nodes, the cost is $\boldsymbol{T(n-1)}$ (the cost of inserting n-1 nodes) and the cost of inserting the $\boldsymbol{n^{th}}$ node. Solving this recurrence, we get the $\boldsymbol{\Theta(n^2)}$ as runtime cost.

Best case of $\boldsymbol{\Theta(n\ lg\ n)}$ occurs when a binary tree of height $\boldsymbol{\Theta(lg\ n)}$ results from the insert operations. When inserting $\boldsymbol{n^{th}}$ node, we are inserting it into a tree with height $\boldsymbol{log(n)}$ since the tree is perfectly balanced. Thus the runtime cost is $\boldsymbol{\sum_{i=1}^{n}(lgi + d) = \Theta(nlgn)}$.

12.3-4

Problem

Is the operation of deletion commutative in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

Solution

The deletion is not a commutative operation. A counter example is shown in the figure 2.

13.1-6

Problem

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

Note that the black height $\boldsymbol{bh(x)}$ is defined as number of black nodes on any path from node $\boldsymbol{x}$ to a leaf, not including $\boldsymbol{x}$.

The smallest possible number of internal nodes is $\mathbf{2^k - 1}$, which occurs when every node is black. This is produced by a complete binary tree with k levels with all nodes black. This tree has $\mathbf{1}$ root at level $\mathbf{0}$, $\mathbf{2}$ internal nodes at level $\mathbf{1}$ so on. Adding up we get, total internal nodes = $\sum_{l=0}^{k} \mathbf{2^l = 2^k - 1}$.

The largest possible number of internal nodes is $\mathbf{2^{2k} - 1}$ which occurs when every other node in each path is a black node. This is produced by a complete binary tree which has alternating levels of black and red nodes. Since the black height is $\boldsymbol{k}$, the height of the tree is $\mathbf{2}\boldsymbol{k}$. Using similar calculations as before, we find that total number of internals nodes is $\mathbf{2^{2k} - 1}$.

13.3-2

Problem

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

The resulting red-black trees are shown in the figure 3.

14.2-5

Problem

Given an element x in an n-node order-statistic tree and a natural number i, how can we determine the ith successor of x in the linear order of the tree in O(lgn) time?

Solution

This can be done easily using the OS-RANK() and OS-SELECT() methods given in the text.

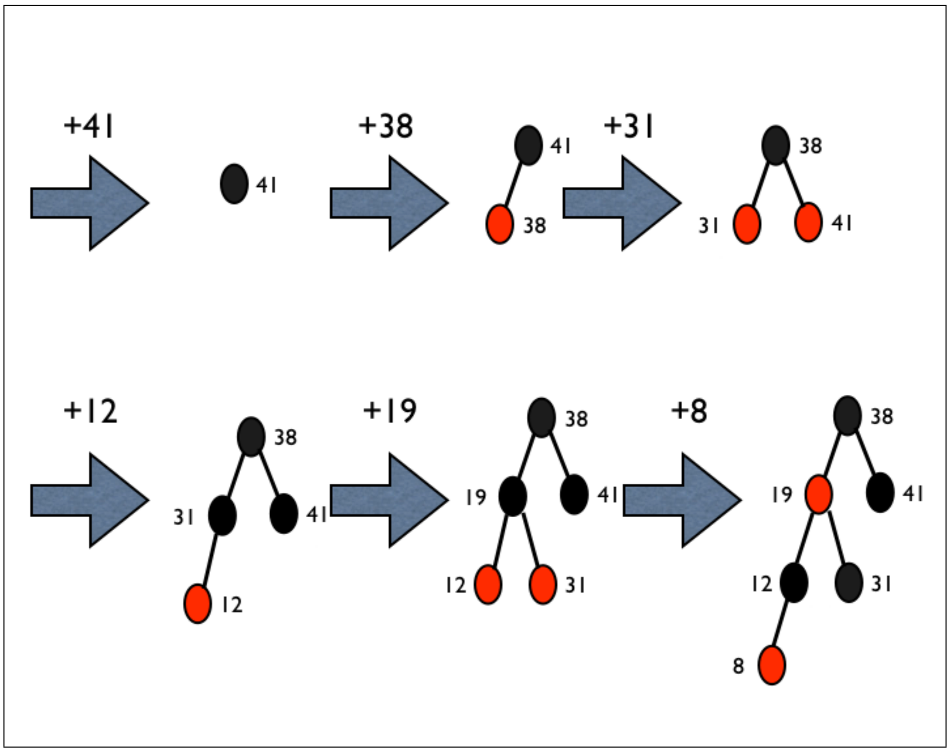

Figure 3: red-black trees that result after successively inserting the keys 41; 38; 31; 12; 19; 8

```
SUCCESSOR(T, x, i) {
   rank = OS-RANK(T, x);
   srank = rank + i;
   Node t = OS-SELECT(T.root, s);
   return t;
}
```

Since OS-RANK() and OS-SELECT() both have $\boldsymbol{O(lg\ n)}$ runtimes, SUCCESSOR() also takes $\boldsymbol{O(lg\ n)}$.