

CSE3358 Problem Set 7

02/24/05

Due 03/04/05

Problem 1: A remarkable property of BST (10 points)

Professor R. E. Mark Cable 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 R. E. Mark Cable claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counter example to the professor's claim.

Problem 2: Height and average depth (30 points)

Consider a randomly built binary search tree, i.e. distinct keys are inserted into an initially empty binary search tree in random order. It is a fact that a randomly built binary search tree on n nodes has an expected height $\Theta(\log n)$. In class we argued this fact by drawing a similarity to the recursive tree of Randomized Quicksort (see your notes).

A common mistake is to claim the above by showing that the expected average depth in the binary search tree is $\Theta(\log n)$, where the average depth is given by $\frac{\sum_{i=1}^n \text{depth}(x_i)}{n}$, because that's easier to prove.

This problem is designed to expose this common mistake.

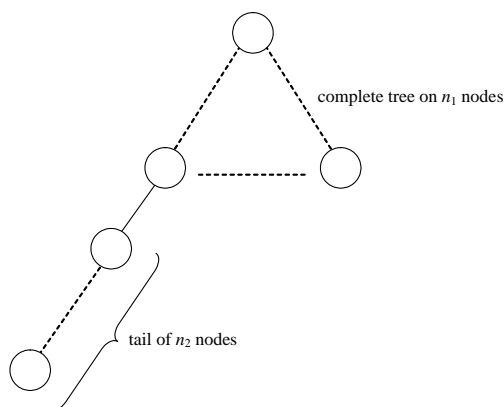
Let $P(n) = \sum_{i=1}^n \text{depth}(x_i)$ for a binary search tree on n nodes. Therefore, the average depth in the binary search tree is $\frac{P(n)}{n}$.

(a) (5 points) Show that $P(n)$ obeys the following recurrence for a **complete** binary search tree:

$$P(n) = 2P(n/2) + \Theta(n)$$

and conclude that the average depth in a complete binary search tree is $\Theta(\log n)$ by solving for $P(n)$.

Consider the following binary search tree consisting of a complete binary tree with a long tail.



(b) (5 points) What is the height of the above binary search tree in Θ notation?

(c) (5 points) Show that $P(n = n_1 + n_2) = \Theta(n_1 \log n_1 + n_2 \log n_1 + n_2^2)$ for the above binary search tree and conclude that the average depth in the above binary search tree is $\Theta(\log n_1 + \frac{n_2^2}{n_1 + n_2})$.

(d) (10 points) Find a relation between n_1 and n_2 to make the average depth in the above binary search tree $\Theta(\log n)$ but its height $\neq \Theta(\log n)$. What is that height in Θ notation?

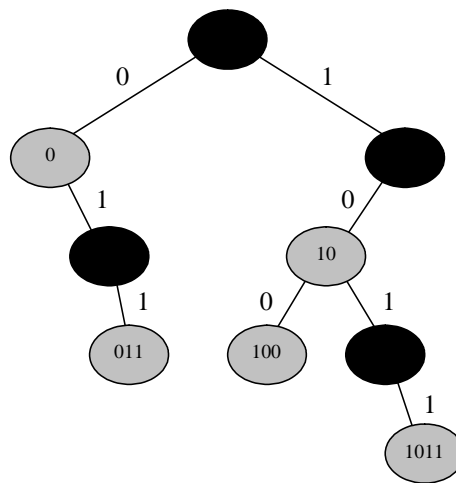
(e) (5 points) How can you intuitively explain the fact that a bound on the average depth in a tree does not imply the same bound on the height of the tree?

Problem 3: Tree walks (30 points)

(a) (10 points) Write recursive pseudocodes similar to the INORDER-TREE-WALK we discussed in class for PREORDER-TREE-WALK and POSTORDER-TREE-WALK (see book page 254 for definitions).

(b) (10 points) Assume the INORDER-TREE-WALK outputs g,b,e,d,f,a,c,j,k,h,i and the POSTORDER-TREE-WALK outputs g,e,b,f,a,j,k,i,h,c,d . Construct the tree and determine the output of the PREORDER-TREE-WALK.

(c) (10 points) Consider a set S of distinct bit strings, i.e. string over the alphabet $\{0, 1\}$. A special tree, called radix tree, can represent such a set S . Here's the radix tree for $S = \{1011, 10, 011, 100, 0\}$.



Given a set of distinct binary strings whose length sum is n , show how you can build its radix tree in $\Theta(n)$ time and then use it to sort S lexicographically (dictionary order) in $\Theta(n)$ time. For example, the output of the sort for the radix tree above should be: 0, 011, 10, 100, 1011.

Hint: use one of the tree walks.

Problem 4: Binary search tree and Min-Heap (20 points)

(a) (10 points) Design an algorithm that converts a binary search tree on n elements into a Min-Heap containing the same elements in $\Theta(n)$ time.

(b) (10 points) Very Challenging (EXTRA CREDIT): Do the same thing *inplace*, i.e. convert the tree into a Min-Heap in $\Theta(n)$ time without using any extra storage. We adopt the standard representation of a tree with the three *left*, *right*, and *parent* pointers.

(b) (10 points) Can you do the reverse process, i.e. convert a Min-Heap containing n elements into a binary search tree on the same elements in $\Theta(n)$ time? How? or Why not?