CSE332 Data Abstractions, Spring 2012
Homework 3

Due: **Friday, April 20, 2012** at the beginning of class. Your work should be readable as well as correct.

This assignment has **five** problems.

Problem 1. AVL Tree Insertion

Show the result of inserting 3, 8, 9, 4, 5, 6, 1, 2, 7 in that order into an initially empty AVL tree. Show the tree after each insertion, clearly labeling which tree is which. It may help with partial credit to show other intermediate steps (e.g., before and after a rotation).

Problem 2. Verifying AVL Trees

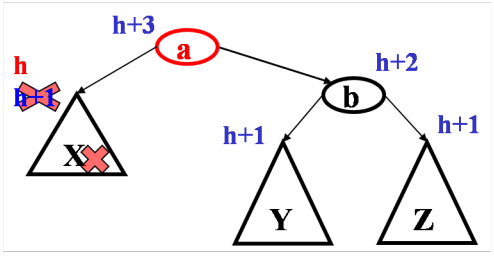
Give pseudocode for a linear-time algorithm that verifies that an AVL tree is correctly maintained. Assume every node has fields `key`, `data`, `height`, `left`, and `right` and that keys can be compared with `<`, `==`, and `>`. The algorithm should verify all of the following:

- The tree is a binary search tree.
- The height information of every node is correct.
- Every node is balanced.

Use `assert` statements so that your pseudocode has an assertion failure if and only if the AVL tree is incorrect.

Problem 3. AVL Deletion

Suppose a deletion somewhere below the left child of an AVL tree node causes both right grandchildren to be too tall, as shown in this picture:



Give a picture showing how to rebalance this portion of the tree. Include height information for all nodes and subtrees shown.

Argue briefly that, in this case, no additional balancing further up the tree should be necessary.

Problem 4. B Tree Insertion

Show the result of inserting 12, 10, 15, 4, 1, 17, 3, 13, 8 in that order into an initially empty B tree with $M = 3$ and $L = 2$. (Recall the text, lecture, and this problem call a B tree what many call a B+ tree.) Show the tree after each insertion, clearly labeling which tree is which. It may help with partial credit to show other intermediate steps.

In an actual implementation, there is flexibility in how insertion overflow is handled. However, in this problem, follow these guidelines:

- Always use *splitting* and *not adoption*.
- Split leaf nodes by keeping the smallest 2 elements in the original node and putting the 1 largest element in the new node.
- Split internal nodes by keeping the 2 children with the smaller values attached to the original node and attach the 2 children with the larger values to the new node.

Problem 5. B Tree Predecessor

In this problem, assume every B tree node has a reference to its parent (except for the root node) as well as its children (except for leaves), so that it is easy to "move up or down" the tree. Suppose you have a direct reference to the leaf holding a data item with a key k.

(a) Describe in English an algorithm for finding the predecessor of k in the B tree (or determining that k is the minimum element and therefore has no predecessor).

(b) Give the worst-case number of total nodes accessed by your algorithm in terms of the tree height h. Describe briefly a worst-case situation.

(c) Give the best-case number of total nodes accessed by your algorithm. Explain briefly why most keys would exhibit the best-case behavior.