EC330 Applied Algorithms and Data Structures for Engineers
Spring 2023

Homework 5

Out: March 26, 2022
Due: April 7, 2022

This homework has a written part and a programming part. Both are due at 11:59 pm on April 7. You should submit both parts on Gradescope.

The written part should be completed individually. The coding problems can be done in pairs. See course syllabus for policy on collaboration.

1. AVL Tree [20 pt]

a) In a binary tree, a child is called an "only-child" if it has a parent node but no sibling, i.e. it is the only child node of its parent. Prove that for any AVL tree with n nodes ($n > 0$), the total number of only-children is at most $n/2$.

b) Is the statement "the order in which elements are inserted into an AVL tree does not matter, since the same AVL tree will be established following rotations" true? If yes, explain why. If not, give a counterexample.

2. Red-Black Tree [20 pt]

An *in-order* traversal of a red-black tree containing the set of numbers 1 to 10 gives the following colors: R B R R B B B R B R.

a) Draw the tree (and indicate the color of each node in the tree).

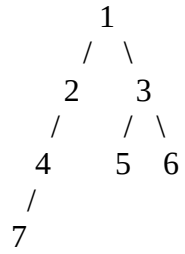
b) Give the sequence of numbers for performing a *pre-order* traversal and a *post-order* traversal of this tree respectively.

3. Programming [60 pt]

Make sure you acknowledge any source you consult at the top of your program.
Do not include a `main` *in your submitted files. Do not modify the header files. You can make a group submission (for both you and your partner) on Gradescope. If you choose to make separate submissions, make sure you write your partner's name as a comment at the top of the submitted .cpp file.*

a) Implement the method *isWeightBalanced(node* root, int k)* for checking if a binary tree is *weight balanced*. Similar to the notion of height balance in AVL trees, we define a binary tree to be *k-weight balanced if for every node in the tree, the difference between the weight of the node's left subtree and the weight of the node's right subtree is no more than k*. The weight of a (sub)tree is simply the number of nodes in that (sub)tree.

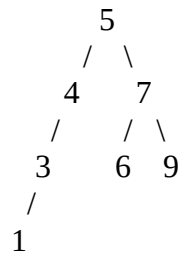
For example, in the binary tree below,



the weight of the subtree rooted at *node-2* is *3*,
isWeightBalanced(*node1*, *0*) should return 0,
isWeightBalanced(*node2*, *1*) should return 0,
isWeightBalanced(*node3*, *0*) should return 1,
isWeightBalanced(*node4*, *1*) should return 1.

Submit *balance.cpp* on Gradescope. **[30 pt]**

b) Consider a binary search tree (BST) whose keys are all unique integers. Implement the *nextKLargest(int x, int k)* function that finds the k^{th} largest key after *x* in the BST. Below are some examples and additional specifications.



Given the BST above, *nextKLargest(3, 4)* should return 7, and *nextKLargest(6, 4)* should return the largest key in the BST which is 9. Note that *x* needs not be present in the tree. For instance, *nextKLargest(2, 4)* should return 6.

Your algorithm must be *asymptotically faster than* $\boldsymbol{O}(n \log n)$ where n is the number of nodes in the BST. Submit *nextKLargest.cpp* on Gradescope. **[30 pt]**