CIS 313 Lab 4: AVL Trees

Due: November 30th, 2016 at 11:59pm

Overview

Construct an AVL tree of numbers. You will extend your code in programming assignment 2 to now include a balancing operations.
If you prefer, we have posted a working BST and Node class for you to build off of.
Note: BST.java is there as a reference. No need to turn it in along with your code.

Fill out all of the methods in the provided skeleton code.
You may add additional methods, but should NOT add public fields. You also should not change the name of any of the classes or files.
The program should continually accept instructions until exit is entered.
On entering any instruction - the result of the operation should be displayed.

In particular, you will implement the following functionality for your AVL Tree:

Insert
Preform BST insert
Check to see if the tree remains balanced
If not, preform the appropriate rotation

delete
Preform BST delete
Check to see if the tree remains balanced
If not, preform the appropriate rotation
Note: You will only be asked to remove the most difficult nodes
(ie) you won't need to recursively rebalance
See Extra Credit

search
Return the node corresponding to the given number
Print "Found" if the corresponding key is in the tree.
Otherwise, print "Not Found"

leftRotate
If x is the root of the tree to rotate with left child subtree T1 and right child y, where T2 and T3 are the left and right children of y:

- x becomes left child of y and T3 as its right child of y
- T1 becomes left child of x and T2 becomes right child of x

rightRotate
If y is the root of the tree to rotate with right child subtree T3 and left child x, where T1 and T2 are the left and right children of x:

- y becomes right child of x and T1 as its left child of x
- T2 becomes left child of y and T3 becomes right child of y

Additionally, implement the following instructions in the main() function in HW4.java

traverse
Preform the preorder traversal of the AVL Tree

exit
Exit the program
Print "Successful Exit"

Extra Credit

The extra credit will consist of two parts:

- Given a binary tree, determine if it is an AVL Tree
    1. Determine if it is a BST
    2. If yes, determine if it is an AVL Tree
    3. Print "Is AVL" or "Is not AVL" depending on the given tree
- You will be asked to preform a complicated delete
    - (ie) After preforming trinode reordering on the parent of the deleted node, you may need to preform it again on a different node

Grading

This assignment will be graded as follows:

Correctness 40%

Your program compiles without errors (including the submitted files NOT containing package names at the top. Delete the package name from the top of the files before you submit them): 10%
Your program runs without errors: 10%
Your program produces the correct output: 20%

Implementation 40%
Insert, delete, search, traverse, and exit all preform in the correct time complexity for the AVL tree: 10%
leftRotate, and rightRotate are also preform in the correct time complexity: 30%

Documentation 20%
To earn points for implementation, your code must be clear enough for us to understand

Extra Credit 40%
Each part of the extra credit will be worth 20%
We will test the extra credit similarly to how we test the rest of the assignment.
Add an additional public function isAVL() to your AVL.java that checks if the current tree matches AVL rules.
We will have our own function that creates a tree from you AVL.java and calls the function isAVL()

For the other half, we will simply run a more in depth test input file.

Further, you may not use any data structures from the Java standard library or the C foreign function interface

Input
Input will be a list of commands (unknown how many), from insert, delete, search, traverse, or exit.
insert, delete, and search will all be followed by a corresponding number.
(eg) insert 4

Note: You should implement your AVL tree with generics, but create an AVL tree that takes in integers.

Sample input
insert 10
insert 5
insert 20
insert 3
traverse
insert 2
traverse
search 17
insert 1
delete 20
traverse
exit

Sample output
10 5 3 20
10 3 2 5 20
Not Found
3 2 1 10 5
Successful Exit