CIS 313 Lab 2: Binary Search Trees

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

Overview

Fill out all of the methods in the provided skeleton code.
You may add additional methods, but should NOT add additional public methods or public fields.
You also should not change the name of any of the classes or files.

In particular, you will implement the following functionality

Insertion

To simplify things, we will not test your binary search tree with duplicate elements.
Insertion should change one of the leaves of the tree from null to a node holding the inserted value.
This should also perserve the ordering requirement that all nodes in the right side of a subtree are greater than the value in the root of that subtree, and all elements in the left side are lesser than the value in the root of the subtree.

Deletion

When deleting a value, delete the node which contains that value from the tree.
If said node has no children, simply remove it by setting its parent to point to null instead of it.
If said node has one child, delete it by making its parent point to its child.
If said node has two children, then replace it with its successor, and remove the successor from its previous location in the tree.
The successor of a node is the left-most node in the node's right subtree.
If the value specified by delete does not exist in the tree, then the structure is left unchanged.

Find

Find takes an int and returns the node in the tree which holds that value.
If no such node exists in the tree, return null.

Traversals: preorder, post order, in order

Print out the elements in a binary search tree, space separated.
You should do this for the preorder, post order, and in ordered representation of the elements.

Extra credit: compare shape of trees 20%

Given two trees, determine if they have the same shape.
That is, determine if they are equivalent trees up to the values held in the nodes.

Recommended Strategy
Create a print function to visualize the shape of trees.
Create trees which should have the same shape, and trees which should have different shapes.
You can then test your extra credit solution with these trees.
This will also help you debug your other functions.

Grading
This assignment will be graded as follows:

Correctness 50%
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: 30%
(5% for insertion, deletion, find, and each traversal)

Implementation 50%
Your Binary Search Tree class implements all of the proper methods in O(n): 50%
To earn points for implementation, your code must be clear enough for us to understand

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

Input
This input will not test find, but we will test find when grading. Make sure to test that as well.

The input will be a single line, with a single number n specifying how many lines follow.
The following n lines contain a single word specifying delete, insert, inorder, preorder, or postorder (and for insert and delete, also a number).
You should create an empty binary search tree (with root null), and then perform the above sequence of actions on that tree.

Sample input

10
insert 30
insert 40
insert 20
insert 10
inorder
preorder
postorder
insert 35
delete 30

inorder

Sample output

10 20 30 40
30 20 10 40
10 20 40 30
10 20 35 40