CIS 313, Intermediate Data Structures
Fall 2018

CIS 313 Lab 2

Due: November 1, 2018

This lab involves implementing a Binary Search Tree

Overview

Fill out all of the methods in the provided skeleton code. You may add additional methods, but should NOT add additional public fields to the BST class. You also should not change the name of any of the classes or files. You will implement a working Binary Search Tree in the BST.java class. You will relay input commands in lab2.java using a scanner. 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 preserve 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.

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.
This will also help you debug your other functions.

Input Description

The input will be a text file, for example *inSample.txt* below will be provided. The first line will contain an integer N, which is the number of lines to follow. Each of the 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 a sequence of actions on that tree.

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

Note: When using an editor, you may also manually type in input to the command window. However, you will be tested with a file similar to *inSample.txt*.

Output Description

The only output will be from the traverse commands. Print each the data in each element in the BST separated by spaces in the correct order (depending on which traverse command was called). For example, using the sample input above, your program should output:

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

Testing Protocol

We will test your program in the same fashion as previous labs. We strongly suggest you test your program in the following ways:

- While creating it
- With inSample.txt
- With the provided test.sh file found in `/home/users/smergend/public/cs313/lab2/test.sh` on ix-dev.

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

Extra Credit 20%

To receive points for the extra credit you will create two Binary Search Trees based on given input, and then you must determine if they have the same shape:

- Create a new public class file called TreeCompare
- TreeCompare will be similar to lab2.java
  - It will read in a list of commands from an input text file
  - The input text file will create two different BSTs
  - The input text file will start with a number N_1
  - Then N_1 lines of insert commands will follow to create the first BST
  - Then there will be a line with a number N_2
  - Then N_2 lines of insert commands will follow to create the second BST
- After you have created the two BSTs, TreeCompare will compare the two trees
- If the trees have the same shape, output "The trees have the same shape."
- Otherwise, output "The trees do not have the same shape."

Here is an example of what the input text file may look like:

```
5
insert 10
insert 20
insert 30
insert 40
insert 50
5
insert 50
insert 40
insert 30
insert 20
insert 10
```

The corresponding output would then be:

```
The trees do not have the same shape.
```