CIS 313, Intermediate Data Structures
Winter 2019

CIS 313 Lab 2

Due: Monday, February 18

This lab task is to implement a Binary Search Tree

Overview

Fill out all of the methods in the skeleton code provided. 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.py class. You will relay input commands in lab2.py through argv[1].
(See `https://www.pythonforbeginners.com/system/python-sys-argv` if you need a reminder on passing arguments to python) 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.
This method is private, therefore it can only be used from within the BST class.

Exists
Exists is like Find, but is a public method that only returns True or False rather than the node itself.
Exists receives a value and returns true if there is a node with the value in the tree.

Traversals: pre-order, post-order, in-order
Print out the elements of the binary search tree, space separated.

You should do this for the pre-order, post-order, and in-order representation of the elements.

Recommended Strategy
Create a print function to visualize the shape of trees.
(one idea: print parent data and child data for all nodes of the tree, then draw on paper to make sure it looks correct).
Create trees which should have the same shape, and trees which should have different shapes.
This will also help you debug your other functions.
As always– write pseudo-code before implementing your methods.
Drawing pictures will help immensely when writing your pseudo-code, especially for the delete method.
Start Early! There are two weeks to complete this assignment for a reason.
If you start on the last day you will probably not finish. Please don't take this warning as a challenge.

Input Description

The input will be a text file, for example *inSample.txt* below will be provided.
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.

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

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 your own examples and test-cases.

Grading

This assignment will be graded as follows:

Correctness 50%

Your program runs without errors: 20%
Your program produces the correct output: 30%
(5% for insertion, deletion, exists, 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 Python standard library You may use lists (i.e., string.split()) while parsing input in your main function, but cannot use lists or dictionaries in any of your methods.

Last submissions will not be accepted after 72 hours past the deadline. Late assignments will receive -10% for each day that they are late.