CSE12 - Spring 2018 - PR6

Binary Search Tree

(200 points + 50 points extra credit opportunity)
Due 11:59pm 08 June 2018

DUE DATE: This is a FRIDAY Due date. You may turn it in as late as Saturday June 09, 2018, 11:59pm. HOWEVER, tutors will not be available after Friday. Piazza response may be limited or zero on Saturday, June 09.

In this assignment you will test and implement a (non-balanced) binary search tree.

If you would like to attempt extra credit, you may optionally implement a balanced search tree using red-black balancing.

This assignment is an individual assignment. You may ask Professors/TAs/Tutors for some guidance and help, but you can't copy code. You can, of course, discuss the assignment with your classmates, but don't look at or copy each other's code or written answers.

The following files are provided for you and can be found on the HW page:

- BinSearchTree12.java
- BST12Tester.java

You will submit at least the following files for this assignment:

- **BST12Tester.java**
- **BST12Adapt.java**
- **BST12.java**

You may submit additional java source files through Gradescope, as you see fit. IF you are doing extra credit, then you must submit:

- **BST12RB.java**

Likely, you will want to define a TreeNode class, simply code it and turn it in, if this is what you choose. We will compile your code with the following (assuming proper CLASSPATH for JUnit testing)

```
$ javac *java
```

IF you turn in BinSearchTree12.java by mistake, we will OVERWRITE it with the one supplied. You cannot change this interface definition.

Assignment Overview

There are essentially 3 components of this assignment.

- Code a Unit Tester - you are given code as a starter. Add to it.
- Code a Binary Search Tree using the adapter design pattern. You should adapt the TreeSet class from the java collections framework. Your adapted class should called BST12Adapt.java
- Code a Binary Search Tree written from scratch called BST12.java

A fourth, optional, and extra credit

- Code a balanced Binary Search Tree using red-black balancing, called BST12RB.java (Note: if you opt for the extra credit, you may adapt or extend your BST12RB.java class to implement BST12.java). The basic idea is you either do the simpler BST12.java or the more complicated BST12RB.java implementation.

BST12Adapt, BST12, and BSTRB all implement the identical interface called BinSearchTree12. BinSearchTree12 is defined as a subset of the methods defined in the Java Collections Framework class TreeSet. It adds 2 additional methods: height() and numChildren().

You May not define ANY OTHER PUBLIC methods for these files. You may, of course, define as many private/protected methods as you deem fit for your solution

The subset of methods defined in BinSearchTree12 have nearly-identical arguments as those in TreeSet. (in some cases arguments of type Object of have been replaced with arguments of parameterized types E, E must be Comparable):

boolean add(E e) Adds the specified element to this search tree if it is not already present.
boolean addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this search tree.
void clear() Removes all of the elements from this search tree.
boolean contains(E o) Returns true if this search tree contains the specified element.
E first() Returns the first (lowest) element currently in this search tree.

boolean isEmpty() Returns true if this search tree contains no elements.
Iterator<E> iterator() Returns an iterator over the elements in this search tree in ascending order.
E last() Returns the last (highest) element currently in this search tree.
boolean remove(E o) Removes the specified element from this search tree if it is present.
int size() Returns the number of elements in this search tree (its cardinality).

Two additional Methods are defined in BinSearchTree12 interface

int height() Returns the height of the search tree. An empty tree returns 0, a tree with one element returns a height of 1.
int numChildren(E target) throws IllegalArgumentException, NoSuchElementException Returns the number of children of the Node that references target.   If target is not found in the tree, throw NoSuchElementException. Any other problems (e.g., Null Pointer, ClassCastException, …) throw, IllegalArgumentException.

Constructors:
In the following table, BST12 can replaced by BST12Adapt and BST12RB

BST12() Constructs a new, empty binary search tree, sorted according to the natural ordering of its elements.
BST12(Collection<? extends E> c) Constructs a new binary search tree containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.

Class Parameterization

class BST12<E extends Comparable<>? super E>> implements BinSearchTree12<E>
class BST12Adapt<E extends Comparable<? super E>> implements BinSearchTree12<E>
class BST12RB<E extends Comparable<? Super E>> implements BinSearchTree12<E>

Exceptions

- Your constructors and public methods should throw the same set of exceptions as the identically named methods of TreeSet. With the additional constraints on E given in the class, some exceptions may not ever occur. In that case, IGNORE those exceptions.

Extends and Implements

- BST12.java, BST12Adapt.java and BST12RB.java (if doing extra credit) must extend Object (See below in extra credit, you may change this requirement for BST12.java to extend BST12RB instead)
- BST12.java, BST12Adapt.java and BST12RB.java (if doing extra credit) must implement BinSearchTree12<E>

Iterators

- Iterators should NOT implement the optional remove() operation (this INCLUDES the iterator for BST12Adapt, we will test this case!)
- Iterators MUST return Elements using in-order traversal of the BST

Javadoc

- You do NOT need to javadoc public methods defined in the BinSearchTree12 interface.
- You MUST define javadoc for any private/protected helper methods that you create.
- You should javadoc any BST12Tester methods that you add/change - put in a comment to explain the purpose of the test. Add YOUR name and email address in javadoc comments in this file

IMPORTANT NOTE ABOUT BST12Adapt:

Since you are adapting an already existing implementation and do not know the details of the implementation, the numChildren() and height() methods are not really easily coded.

- For height(), return 0 for an empty tree, 1 for a tree with one node, and size() for all other cases
- For numChildren() return -1 if the target node IS in the tree. throw the NoSuchElementException, if the target node is NOT in the Tree, IllegalArgumentException for any other faults.

Testing

You are being supplied a minimal tester, when you turn it it, it should be set to test instances of BST12.. If you are clever, because all classes implement the identical interface, you should be able to come up with a “factory” constructor that allows you to choose (at compile time) whether you are testing the BST12, BST12Adapt, (or BST12RB if doing extra credit).

You should add tests to the supplied code. At minimum, every public method in the interface should have at least one test. You should build tests that help you debug and validate your code. Your add/remove/iterator tests should test a variety of cases and our expectation is that this is where you will spend most of your time in testing/debugging. If you are doing the extra credit, make sure that your tests would exercise the various special cases of Red-Black. There is not a hard and fast rule about how many testers you should have -- simply do your best.

Extra Credit (50 Points)
You are being given the opportunity to earn extra credit on this assignment, Any extra credit points will be added to your homework/programs score. The total possible points for all homework in the class is 800 points, if extra credit puts your total above 800 points, you WILL receive those points. In other words it is possible to earn more than 100% on the homework portion of your class grade.

We will be going over Red-Black trees during lecture in 10th week. You may look at the Wikipedia page for Red-Black trees at https://en.wikipedia.org/wiki/Red%E2%80%93black_tree AND use the sample C-code as a guide for your implementation. **Using that code, which covers the various cases, is not considered to be violating academic integrity for THIS assignment.** One caveat, you may **NOT DOWNLOAD the C code** and then edit. **You must TYPE IN each line of code yourself**. (this is to help you better understand the various cases of Red-Black trees).

Implementing BST12.java as an adaptation of your BST12RB.java class
You may implement BST12 as either an adaptation of BST12RB.java OR as a subclass. If you choose inheritance (highly recommended), then BST12.java may extend BST12RB instead of extending Object

Hints
Write BST12Adapt first. It should be fairly quick to code and you can build some good test cases against a known implementation.

Turning in your code
TAs will provide instructions to turn in via Gradescope