Homework #3 CS1332
Summer 2015

Binary Search Trees

Goal: You will program a basic Binary Search Tree (BST) WITHOUT rotations. This code will then be the basis of HW#4 where you will code an AVL Tree.

General:
Binary search trees (BST) are very useful in maintaining dynamic sorted data. If we knew all the data ahead of time, we could put it in an array, sort it once, and then find things efficiently using binary search in O(log n) time. If we don't know all the data ahead of time, then we need a way to dynamically add new data, yet not have to sort it each time we add something.

BST's provide a way to do this, by basically encoding the binary search algorithm into their structure. Recall from class that a BST will provide us with an expected O(log n) search time, but can degenerate to O(n) if the data is inserted in sorted order.

Implementation:

You will implement the provided BST interface. Your implementation should be named BST<> . The interface includes standard methods plus various traversals. You may use the List<E> and ArrayList<E> classes to return values from these traversals. In the level-order traversal you may use the Queue<E> interface and ArrayDeque<> classes also to perform the algorithm.

You will also implement your own Iterator for the tree, which will return items using your in-order traversal. You may use the ArrayList's own iterator to help you with this function.

Make both your iterator (if needed) and your BNode class inner classes of your BST class.

Specific instructions on each method are found in the javadoc comments of the provided interface.

NOTE: While you are free to use an iterative or a recursive approach to the algorithms, keep in mind for the AVL homework, a recursive strategy may prove easier.

NOTE: If you want to use parent pointers, you may add them to the Bnode class and use them.

NOTE: For removal, you are required to use the in-order successor.

Code Quality:

You should javadoc your classes. Be sure and include the @author tag to identify yourself. If you used code from the textbook or from class resources, be sure and include that information in the class comments.

Remember that it is poor coding practice not to return a boolean directly. So do this:

```
return x > 5;
```

NOT THIS:

```
if (x > 5) return true
else return false
```

I like to name parameters differently from my instance variable names so that I do not have to preface everything with this. It also reduces the chances of name hiding logic errors and scoping bugs.

For example I like this:

```
void doSomething(int cnt)  { count = cnt; }
```

NOT THIS:

```
void doSomething(int count) { this.count = count; }
```

Any helper methods you create should be made private. You do not want someone able to call those helper methods from outside the class.

Testing:

Remember that this homework starts the period where I will provide you with *some* tests, but passing them does not ensure that you will receive 100. When adding test cases, remember to use the @Test tag for your method. By convention, all test methods begin with the lower case word test, and then describe the test you are writing, for example:

```
@Test
public void testRemoving RootWhenThreeItems( ) { // your code. }
```

There are several asserts provided by JUnit and you have already seen a few. Here is a list:

```
assertTrue (boolean)
assertFalse(boolean)
assertEquals(Expected, Actual)
assertSame(Object 1, Object 2)  // assert that these are the same exact object in memory
assertNull(object1)
assertNotNull(object1)
```

You are not required to implement Junit tests and you will not submit these tests, but it is recommended that you think about all possible cases in implementation and add tests for

Just for Fun:

A basic file named Timer.java is provided to allow you to see the effects of your simple implementation, vs one which guarantees that access is in O(log n) time. If you run Timer after you have your tree working, then it will print out the results of finding something not in the tree, and something in the tree (but the max item) using the built-in balanced tree of Java, and your implementation. Our next homework will add rotations to the tree, so that our tree is much more

competitive with the Java implementation.

Provided Files:

BinaryTree.java
BinaryTreeTest.java
Timer.java

Submission:

Just submit your single file: BST.java. Any supporting classes like Bnode and BSTIterator, should be inner classes of this file.

Grading:

1. Non-compiling code receives a zero.
2. Use of any collection class outside of the traversals will result in a zero.
3. Point deductions will be made using our JUnit, which has all the tests in the provided test file plus others that fully test the data structure.
4. The Timer exercise is just for fun and has no effect on the grade.

Points:

isEmpty …..... 05
size …..... 05
add …..... 10
max …..... 05
min …..... 05
contains …..... 10
remove..... 15
iterator …..... 10
getPostOrder …... 10
getLevelOrder …... 10
getPreOrder …..... 10
getInOrder …..... 10
clear …..... 05

Total …..... 100