CSC 364 Homework #5 Instructor: Jeff Ward
Due: 11:59pm, Wednesday, April 5, 2017
Covers: BSTs and AVL Trees 40 points

Part I: Improving the runtime performance of BST iterators (20 points)

Download the following files from Blackboard: BST.java, Tree.java, TestBSTWithFastIterator. Create a file: BSTWithFastIter.java. The BSTWithFastIter class should extend the BST class. However, it will override the iterator() method and provide its own iterator class. The new iterator() method should create an iterator object in O(1) time. The iterator class' hasNext() method should also run in O(1). The iterator's next() and remove() methods should run in O(h) time where h is the height of the underlying BST.

Here is pseudocode for the BSTWithFastIter's iterator methods. It assumes that an iterator object will have the following three datafields:

```
    private java.util.Stack<TreeNode<E>> stack =
      new java.util.Stack<>();
    private TreeNode<E> current = root;
    private E lastReturned = null;
```

We maintain as an invariant that the nodes remaining in the iteration are (1) current and its descendants and (2) the nodes on the stack and their right subtrees.

Pseudocode --

```
hasNext():
  return true iff current is not null or stack is not empty

next():
  if there is no next element
    throw NoSuchElementException
  while current != null
    push current
    current = current.left
  pop node
  lastReturned = node.element
  current = node.right
  return lastReturned

remove():
  if lastReturned is null
    throw IllegalStateException
  call delete on the BST, deleting last returned element
  set lastReturned to null
```

TestBSTWithFastIterator.java will run some tests of the correctness and performance of your BSTWithFastIter's iterator() method amd iterator class, comparing them to the performance of those from Liang's BST class. The comment's at the beginning of the file describe the exact tests in more detail. Run this program to make sure that your code performs correctly and quickly. Here are the results that I obtained when I ran it on my solution to this problem:

```
Adding 10000000 random values to plain BST and to BSTWithFastIter.
Time to create trees:  31.262 seconds
Using plain BST:
      Time to create iterator:  0.960 seconds
      Time to iterate through 10 values:  0.000 seconds
      Time to iterate through all 9988472 values:  1.056 seconds
      Time to iterate through 1000 values, removing every fourth one:
78.236 seconds
      Time to iterate through remaining 9988222 values:  1.636 seconds
Using BSTWithFastIter:
      Time to create iterator:  0.001 seconds
      Time to iterate through 10 values:  0.000 seconds
      Time to iterate through all 9988472 values:  0.576 seconds
      Time to iterate through 1000 values, removing every fourth one:
0.000 seconds
      Time to iterate through remaining 9988222 values:  0.545 seconds
Good -- Results match correctly.
```

Part II: Finding the k-th Smallest Element in an AVL Tree quickly (20 points)

This problem is based on Programming Exercise 26.5 of the textbook.

Download TestFindKthSmallest.java. You are required to add a `find` method to the AVLTree class:

public E find(int k) // Returns the k-th smallest element of this tree.

Follow the instructions and suggestions provided in the Programming Exercise 26.5 writeup:

(1) Add a size data member to the AVLTreeNode class. The size value on a node is the number of nodes in the subtree rooted at that node.
(2) The find method should return null if k < 1 or k > size of the tree.
(3) Otherwise, the find method will use a recursive helper method: find(int k, AVLTreeNode<E> root).
(4) find(k, root) should use the recurrence relation defined near the bottom of page 983:

Let A and B be the left and right children of the root, respectively. Assuming that the tree is not empty and $k \le$ root.size, then

find(k, root) = root.element, if A is null and k is 1;
= B.element, if A is null and k is 2;
= find(k, A), if $k \le$ A.size;
= root.element, if k = A.size + 1;
= find(k – A.size – 1, B), if k > A.size + 1;

The insert, search, and delete methods should still run in O(log n) time where n is the size of the AVL tree. Your find method should also run in O(log n) time

Although in the paragraph at the bottom of page 983, the textbook suggests that you modify the insert and delete methods, you may be able to leave those methods unmodified but modify instead the updateHeight method. (Both insert and delete call updateHeight.)

TestFindKthSmallest.java will test your code for correctness and also time it against an approach that uses an iterator to find the k-th smallest element. The iterator approach takes O(n) time, so your find method should provide much faster performance.

Here is the output from my solution:

```
Creating AVL tree with 20000 elements.

Starting 20000 tests using iterators.
Runtime using iterators:  4.612 seconds
Checking results ... okay.

Starting 20000 tests using find method.
Runtime using find method:  0.014 seconds
Checking results ... okay.
```

What to turn in:
Submit on Blackboard your BSTWithFastIterator.java file for Part I and your modified AVLTree.java file for Part II.