Lab Exercise 8
Binary Search Trees

A binary search tree is a binary tree of ordered nodes. The nodes are ordered by some key value, for example: alphabetic or numeric. The left subtree of every node (if it exists) contains nodes with keys less than the parent and the right subtree (if it exists) contains nodes with keys greater than the parent.

This lab has you program various things on a binary search tree using private helper methods. You will do three tasks (your TA will lead you through the first one), each with these three steps. First figure out how to do the computation on an example tree of beer names, then show the general pattern, and finally implement it as a helper method.

Using Eclipse, import “lab8”:

/afs/cs.uwm.edu/users/classes/cs351/401/pantherid/git/lab8.git

But your first task will be on paper.

1 Task 1: Find In Range

For the first task, you will find all the keys in a particular range, your TA will demonstrate the three steps methodology with this task.

1.1 Step 1: Finding Keys in an Example Tree

In the following tree, write next to every node, all the keys that are in the subtree rooted at that node that are in the range (dancingMan, schlitz). For example the set for “bud” is {lakefront}, and the set for “michelob” is {michelob, miller}.

1.2 Step 2: Cases for Finding Keys

We have separate cases depending on whether the key is before the range, within the range or after the range. We want to avoid looking in a subtree where nothing could be in the range (to save time).
Case key \( k \) comes before \((x, y)\):

- Look in left subtree? ___________ (yes or no)
- Look in right subtree? ___________ (yes or no)

Case key \( k \) is in range \((x, y)\):

- Look in left subtree? ___________ (yes or no)
- Look in right subtree? ___________ (yes or no)

Case key \( k \) is after \((x, y)\):

- Look in left subtree? ___________ (yes or no)
- Look in right subtree? ___________ (yes or no)

1.3 Code for Find

Implement private helper method `findInRange`.

Do not have any more than three cases, even if \( x \) or \( y \) is null. If \( x \) is null, then \( k \) can’t come before \((x, y)\). If \( y \) is null, then it can’t come after.

2 Task 2: Maximum Height

A binary search tree will not work very well, if it is very tall; the runtime for many algorithms depends on the height. In this section, you compute the height of a binary search tree node. The
height of a node is how many nodes are there on the longest path from this node down to a null pointer. For a leaf node, the height is 1; the null pointer has height 0.

2.1 Step 1: Height for an Example Tree

On the following tree, please write the height of each node to the left of the node. Hint: the height of “michelob” is 2.

```
leinie
  
bud
  
best
  
anchor

lakefront
  
coors

riverwest
  
pabst
  
redhook

  sprecher

  miller
```

See the TA for checkoff #1.

2.2 Step 2: Cases

Now try to figure out a way to solve the problem in general. Fill in the following table:

- Height of a null pointer: ____________
- Height of a node: ____________________________ (equation using $h_1$ and $h_2$)
  Assuming $h_1$ is the height of the left subtree and $h_2$ is the height of the right subtree.
  Hint: You may use “max($x,y$)” to compute the maximum of two numbers $x$ and $y$.

See the TA for checkoff #2.

2.3 Step 3: Code

Implement a private helper method `doHeight(Node node)` in class `edu.uwm.cs351.BST` that uses a single if to distinguish these two cases. You should use `Math.max` in your code. Make sure you pass the test suite `TestHeight`.

3 Task 3: Range Check

A key invariant for a binary search tree is that, if its left subtree is not empty, then all the keys in that subtree is less than the key of the root node, and similarly if right subtree is not empty, then all keys in that subtree is greater than the key of the root node.

Alternatively, the key in a node is greater than all “left” ancestors (ancestors for which we are in the right subtree) and less than all “right” ancestors. By picking the greatest left ancestor (if
any) and the least right ancestor (if any), the legal range for a key can be written as an open interval \((x, y)\) where \(x\) is the lower bound (greatest left ancestor key) and \(y\) is the upper bound (least right ancestor key). For example, the range for “pabst” is (“leinie”, “riverwest”) and the range for “best” is (nothing, “bud”).

3.1 Step 1: Range for an Example Tree

Now put the range for each node (you can omit “best” and “pabst” to leave more room):

See the TA for checkoff #3. Don’t go on until the TA has checked that you understand the task.

3.2 Step 2: Cases

Now give how to solve the task in general. Since this task is top-down, it has to special case the root.

Range for the root node: ____________________________.

Assuming a node has range \((x, y)\) and key \(k\):

The range for the left subtree is ____________________________.

The range for the right subtree is ____________________________.

See the TA for checkoff #4.

3.3 Step 3: Code

Now implement the function \texttt{checkTree(Set<String> checks)} using the following helper function which should also be implemented:

\texttt{doCheckTree(Node n, String lo, String hi, Set<String> checks)}

Each time \texttt{doCheckTree} is called on a non-null node, it should create a string \(k<-(lo, hi)\) without any spaces.

Make sure you pass the \texttt{TestRange} test suite. See TA for the final checkoff #5, to check your code for the last two tasks. Make sure you have \emph{no} more cases than this sheet says.