CMPSC 250
Analysis of Algorithms
Fall 2012
Bob Roos

Lab 5
Tues., 16 October 2012
Hand In by 2:45pm Tues., 23 Oct.

I strongly suggest that you create a new subdirectory called "lab5" or something similar and place all your files from this lab in that directory. At the end of the assignment I'll ask you to create a "zip" file to upload to the Sakai drop box.

Operations on Binary Search Trees

Today you will be working almost exclusively with the class "`BST.java`" from the textbook website. I have modified their code with a few extra comments and placed it on the Sakai site along with this lab writeup.

1. **Make sure it works (nothing to hand in here).** Download `BST.java` and `tinyST.txt` from the Sakai site. Compile `BST.java` and run it, redirecting input from the file `tinyST.txt`. You should see output that looks like:

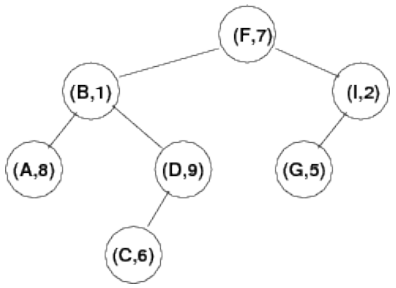
   ```
   S 0
   E 12
   X 7
   A 8
    ...
   ```

   Note: if you are using Dr. Java (and I don't know if any of you are), you can not do "input redirection" from inside the Dr. Java "Interactions" panel—you will have to do it from the terminal line.

2. **Understand it (hand in "tree" picture in a text document).** The `main` program in `BST.java` reads in a file of text and treats each word as a key. The associated value is the position of the word in the text file. For instance, in the file `tinyST.txt`, the first "word" is "`S`", so it is assigned value 0. The second word is "`E`," and it is assigned a value 1. However, "`E`" appears multiple times, and the way the BST is set up, inserting a $(key, value)$ pair for an already-existing key simply updates the value. The last "`E`" in the data file is in position 12, so that is the final value associated with key "`E`".

   On a piece of scrap paper, construct *by hand* the binary search tree corresponding to the input file `tinyST.txt`. There will be 10 nodes in the tree, one for each of the distinct letters in the file. For each letter, indicate the *final* value associated with it next to it in the tree.

If you have not already done so, create a document to hold your answers to this lab. Enter the tree "sideways", one node per line, using indenting to indicate depth. Choose an indentation amount that maked the tree structure clear; 3 or 4 spaces per tree level is probably good enough. All nodes at the same depth should be indented the same amount. For instance, here is a tree and its corresponding text representation (but the data are not the same—this corresponds to input "`F B I D A G C F A D`"):




```
        I(2)
            G(5)
F(7)
            D(9)
                C(6)
        B(1)
            A(8)
```

3. **Traverse the tree (add code to `BST.java` at indicated place).** In class we looked at three different "traversal" orderings for printing out the contents of a binary tree; here is the pseudocode:

```
preorder(Node x):          postorder(Node x):         inorder(Node x):
   if x==null return          if x==null return          if x==null return
   print x                    postorder(x.left)          inorder(x.left)
   preorder(x.left)           postorder(x.right)         print x
   preorder(x.right)          print x                    inorder(x.right)
```

   Each of these can be implemented in Java as a pair of methods—one public, one private. Here's the general plan:

```
public void preorder() { // this is the one the user calls
    preorder(root);
}

private void preorder(Node x) {
     ...
}
```

   In the program `BST.java`, write six methods (I've indicated where to put them using comments) for these three traversals. In the `main` program, add calls to these three methods at the end, e.g.,

```
StdOut.println("Preorder:");
st.preorder();
```

4. **Compute the depth of a key (add code to `BST.java` at the indicated place).** Write a method named `depth` that takes, as its argument, a `Key` and returns an integer representing

the *depth* of that key in the tree. Nonexistent keys have a depth of -1. For instance, in the tree above, F has a depth of 0, G has a depth of 2, C has a depth of 3, and X has a depth of -1.

5. **Figure out complexity.** Assuming the random-key distribution holds, how long does your `depth` method take to determine the depth of a key?

6. **Figure out complexity.** In the worst case, how long does your `depth` method take to determine the depth of a key?

7. Move up one level from your `lab5` directory and type "`zip -r lastnamelab5.zip lab5`" (if you named it something else, use the something else). Now upload this to your drop box.