COMS 3134 Homework 3

Submission instructions

All programs must compile and run on CUNIX to receive credit. Submit your electronic files via `http://courseworks.columbia.edu`. We prefer electronic submission of theory, though you will not be penalized for paper submissions. (Do **not** print out your programs.) I recommend learning LaTeX for typesetting math. Include a README file that explains exactly what each file in your submission is. Place all the files you want to submit into a submission directory with the following naming scheme.

`<your_uni>_hw<number>`

So if my UNI is `uni1234` am submitting homework 6, my directory would be `uni1234_hw6`. Archive your submission directory using

```
tar -czvf uni1234_hw6.tar.gz uni1234_hw6
```

and upload `uni1234_hw6.tar.gz` to courseworks. (You will probably need to first download the file to a local directory using an FTP program. See CUNIX tutorial for more info.)

Multiple Submissions: You can submit multiple times, but we will only consider the latest submission based on the timestamp in courseworks. Please give at least 1-2 minutes between two submissions so we can tell which is the newest submission.

I recommend that you also keep a pristine copy of your submission folder in case there is any submission error.

Theory Problems

Make sure your solutions are clear. Diagrams and math are often insufficient to convey exactly what you mean, so supplement with some text. Either pseudocode or Java are acceptable when asked to provide algorithms. Nevertheless, clear, concise English is often preferable.

1. (6 points) **Weiss 4.6** A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.
2. (4 points) **Weiss 4.9**
   (a) Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.
   (b) Show the result of deleting the root

3. (4 points) **Weiss 4.19** Show the result of inserting 2, 1, 4, 5, 9, 3, 6, 7 into an initially empty AVL tree.

4. (4 points) **Weiss 4.27** Show the result of accessing the keys 3, 9, 1, 5 in order of the splay tree in Figure 4.71.

5. (6 points) **Weiss 4.41** Write a routine to list out the nodes of a binary tree in level-order. List the root, then nodes at depth 1, followed by nodes at depth 2, and so on. You must do this in linear time. Prove your time bound.

6. (6 points) **Weiss 4.49** Suppose we want to add the operation `findKth` to our repertoire. The operation `findKth(k)` returns the kth smallest element item in the tree. Assume all items are distinct. Explain how to modify the binary search tree to support this operation in $O(\log N)$ [i.e., $O(d)$] average time, without sacrificing the time bounds of any other operation.

Programming Problem

1. (30 points) **Autocompletion via Tries**

   Write a program that prompts the user for the beginning of a word and outputs all the possible words that can complete what the user typed in alphabetical order. For example, one possible user interaction would be:

   ```
   $ java AutoCompleter dictionary.txt
   Loading Dictonary. Standby...
   Dictionary loaded!
   Start typing a word and hit enter ('quit!' to end)
   algori
   Possible completions:
   algorithm
   algorithmic
   ```

   Perform the autocompletion lookup by storing a dictionary of words in a trie. Load the dictionary when your program starts from a text file of words. Use the included Scrabble dictionary file `TWL06.txt`[1].

   Write your own `MyTrie` class for Strings that performs insertions, lookup and preorder toString() conversion (no deletion is necessary). At the cost of memory usage, simplify your code by storing the full word at each leaf. The lookup method should take the beginning of a word input string and return the sub-trie of words that start with the input string. Calling toString() on this sub-trie should return the desired output (i.e., put the newlines in the string conversion to create the kind of output in the above example).

   Extra credit 5 points: Modify your code a bit to search for Scrabble-style completion. Instead of finding words that start with the entered letters, find words that contain the entered letters, in any order. (This is pretty tricky; make sure you finish your required work before working on this.)

[1] I downloaded this file from `http://www.isc.ro/en/commands/lists.html`