CS 127: Data Structures	**Winter 2006**
Handout 7: Problem Set #5 (Due: Thursday, 16 February 2006, 11:59p)	*10 February 2006*

The mind boggles.
— Charles Lederer (1910–1976).[1]

This is a group assignment.

In this assignment, you'll build a Boggle-playing program to end all Boggle-playing programs. *Boggle* is a game that works as follows:

- An 8-by-8 grid of 6-sided dice is arranged. The faces of the dice have letters on them, with the percentage of the $8 \cdot 8 \cdot 6 = 384$ faces that have letter x on them roughly proportional to the frequency of the letter x in the English language. (So there are a lot of vowels, for example, and not so many `J`'s and `Q`'s and `X`'s.)
- A timer starts. You have a minute to produce as many English words that are contained in the grid as you can. A word is contained in the grid if you can find a line of dice containing its letters in order. You may use diagonals, but you may not repeat the same die twice.

For example, you might find the word `REBOOTED` in this grid:

B	E	Q	X	O	O	B	R
T	A	N	V	T	F	A	E
A	E	F	U	E	E	A	U
T	A	A	B	D	A	T	E
X	Z	T	T	W	Z	B	D
A	N	V	T	A	N	V	T
E	F	U	E	E	F	U	E
A	A	B	D	A	A	B	D

To write a Boggle-playing program, you'll have to some clever search techniques to find candidate words, and use a dictionary of words of the English language to figure out when a candidate word is an actual word.

0. Estimate the amount of time you spent on this problem set, and write it at the top of your `ps5.txt`.
1. Grab your BinarySearchTree class from the last assignment. Let me know if you had trouble getting it working. Modify it so that each node in the tree contains only a String (we're just going to store a list of words, without any satellite data).

   Write a class Boggle that stores a grid of characters, and also contains the following methods:

   - a constructor that builds a random board of characters. You'll want to do something to ensure that you have a reasonable number of vowels. You can do something very clever if you'd like, or you can do something less clever, like I did. (Whenever I generated a non-vowel, I threw it away and tried again, and did the same thing one more time. That roughly doubles the proportion of vowels in your board.)
   - `public String toString()`, which prints out the board.
   - `void findWords(BinarySearchTree tree)`, which print out all words *of fewer than 9 letters*[2]

[1] Screenwriter for "The Thing from Another World" (1951), the movie better known as "The Thing." Previous line: "An intellectual carrot." Apparently a character has just been told that the alien is an intelligent vegetable life form. The mind boggles indeed.

[2] That's to save you time; longer words will take a *long* time to find!

that are contained in the board (as defined above) and are also present in the BST. You'll need to think carefully about how you want to do this! (Remember 8 queens?) I recommend that you use backtracking, and that you carefully consider what parameters your algorithm needs. Don't print out the same word more than once. See below for some tips.

- A main method that loads a dictionary as specified by a command-line argument into a BST, then prints out the Boggle board and all the words that can be found in the board.

  On the mathcs machines, you'll find a file called (intuitively enough) `/Accounts/courses/cs127/dlibenno/twl98.txt`.[3] This should serve as your dictionary throughout this assignment.

2. You'll notice that loading the dictionary takes a *long* time. Actually, you probably thought that your machine crashed, or that you were in an infinite loop, or something. (I did.) In your `ps5.txt`, explain why (a sentence or two is fine), and then fix it. You can fix it in either of the following ways:

   - Write a new BST constructor that accepts as input the name of a file containing one word per line, and produces a well-balanced tree if the file is sorted.
   - *(Harder—not required, but you'll get extra credit for doing it.)* Implement a balanced BST scheme—either AVL trees or red/black trees.

   Actually, you might want to do this problem before you do problem #1, because #1 will be a lot faster once you can load the dictionary quickly!

Hints and tips:

1. If you wondering why you don't just use an array to store the dictionary and binary search to do lookups, the answer is: because that wasn't the assignment. It *would* make more sense to do it that way, because the dictionary is already sorted, but I want you to have the experience.
2. As with the last problem set, this assignment has a lot of pieces. Start early!
3. A good way to test everything: start with a smaller board—maybe four by four.
4. Another good way to do testing: start with a predefined board that you hardcode, and a small dictionary that includes some words that are found in it.
5. Think about how to efficiently store the words that you've found already.
6. There's a sample run of my code on the website. Again, it's worth comparing to what you're getting.
7. If you want to do #1 before you do #2, you can *temporarily* use the built-in `TreeSet` class instead of the BST class that you've written. (TreeSets actually are red/black trees.)
8. A good way to test your backtracking search: print out every word that you try to find.
9. Start early!
10. Ask questions!
11. Keep in touch!

[3] It's "the word list" (twl), as in The Official Scrabble Word List, from 1998.