Homework #7 Data Structures Due: November 24 (Wednesday at 11:59 PM)

Word Concordance

Objective: To learn how to implement the dictionary ADT using BST and AVL trees, and to gain an understanding of the relative performance of the BST and AVL trees.

To start the project: Download and unzip the file hw7.zip. You'll also need to copy your BST implementation from Lab 9 and your AVL implementation from Lab 10.

The Assignment: This assignment has several parts -- a comparison of dictionary ADTs (section 19.2.2 of the text) and a concordance-production application using the dictionary ADTs. A Webster's dictionary definition of concordance is: "an alphabetical list of the main words in a work." In addition to the main words, I want you to keep track of all the line numbers where these words occur. NOTE: You do not need to implement the `myDictionary.pop(key)` method for the AVL tree implementation.

WORD & LINE CONCORDANCE

The goal of this assignment is to process a textual, data file to generate a word concordance with line numbers for each word. A dictionary ADT is perfect to store the word concordance with the word being the dictionary *key* and a list of its line numbers being the associated *value* with the key. Since the concordance should only keep track of the "main" words, there will actually be two files of textual information: the data file to be processed and a stop-word file. The stop-word file will contain a list of stop words (e.g., "a", "the", etc.) -- these words will **not** be included in the concordance even if they do appear in the data file. The stop words are to be stored in a second dictionary which will be accessed to ensure its contents do not appear in the concordance. Words in the data file are to be extracted, compared with the stop words, and, when appropriate, added to the concordance list along with the line number of the current occurrence of the word. Often, a word will be encountered several times--each line of encounter is to be recorded in the concordance list, but each word is to appear only once. The output of the program should be a text file containing the concordance words printed out in alphabetical order along with their corresponding line numbers.

NOTES:

a) Words are defined to be sequences of letters that are delimited by any white space, punctuation, brackets, parentheses, dashes (two hyphens in a row), double quotes, etc. but not an apostrophe or single hyphens. For example, "it's" and "end-of-line-characters" should be considered words.

b) There is to be no distinction made between upper and lower case characters, i.e., "ADT" is the same word as "adt".

c) Blank lines are to be counted in the line numbering.

d) The word-concordance application should not reference the dictionary ADT implementation, except through the dictionary operations.

e) It is strongly suggested that the logic for reading words and assigning line numbers to them be developed and tested separately from other aspects of the program. This could be accomplished by reading a sample file and printing out the words recognized and the lines they appeared on with no effort to avoid duplicates or associate words with more than one line.

DICTIONARY ADT

I want you to implement the dictionary ADT two ways:

1) use your binary-search tree (BST) implementation from Lab 9 as the underlying data structure, and

2) use your AVL tree implementation from Lab 10 as the underlying data structure. (you do not need to implement the dictionary `pop` operation since that would require an AVL tree delete operation. You may need to modify the ADT `add` operation so that it replaces the old item by the new item if it already exists in the tree.)

Time the word-concordance application with both dictionary ADT implementations, and complete the following table:

Word-concordance Program	Execution Time (seconds)
Dictionary ADT implemented using a single BST	
Dictionary ADT implemented using a single AVL tree	

DATA FILES

The stop words are in the file `stop_words.txt` contained in the `hw7.zip` file.

The textual information to be examined is in the file `hw7data.txt` contained in the `hw7.zip` file.

HINTS:

0) The word-concordance application code and each dictionary ADT should be developed and tested separately.
1) A built-in Python list or a linked implementation of the queue ADT may be useful to store the lines-of-occurence for each word.
2) You might want to use a regular expression to define a word. See the HOWTO at:
http://docs.python.org/dev/howto/regex.html

Implement AND fully test both your BST and AVL tree versions of your word-concordance program. You are to submit:

- the timings of both versions of your word-concordance program,
- a one page overview of the design of your program, directions for running your program,
- all of your program files, and
- the output files containing your word/line concordance produced by running each version of your program.

Submit all program files as a single zipped file (called hw7.zip) electronically at:
https://www.cs.uni.edu/~schafer/submit/which_course.cgi