Data Structures (CS 1520) Homework #6 Due: 12/9/17 (Sat.) at 11:59 PM

This assignment has several parts -- a comparison of dictionary/map ADTs (Lecture 15 at: http://www.cs.uni.edu/~fienup/cs1520f17/lectures/lec15_questions.pdf) 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 main words occur. (e.g. like an index at the back of a textbook, but with line #s instead of page #s)

WORD & LINE CONCORDANCE APPLICATION

The goal of this assignment is to process a textual, data file (`WarAndPeace.txt`) to generate a word concordance with line numbers for each main word. A dictionary ADT is perfect to store the word concordance with the word being the dictionary *key* and a Python 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 a second stop-words file (`stop_words.txt`). The stop-words 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. Sample files might be:

Sample `stop_words_small.txt` file

```
a
about
be
by
can
do
i
in
is
it
of
on
the
this
to
was
```

Sample `hw6small.txt` file

```
This is a sample data (text) file to
be processed by your word-concordance program.

The real data file is much bigger.
```

Notes:

1) Words are defined to be sequences of letters delimited by any non-letter. (e.g., white space, punctuation, parentheses, dashes, double quotes, etc.)
2) There is to be no distinction made between upper and lower case letters. (e.g., "CAT" is the same word as "cat")
3) Blank lines are to be counted in the line numbering. (e.g., line 3 above is blank)

Sample output file

```
bigger: 4
concordance: 2
data: 1 4
file: 1 4
much: 4
processed: 2
program: 2
real: 4
sample: 1
text: 1
word: 2
your: 2
```

The general algorithm for the word-concordance program is:

1) Read the `stop_words_small.txt (or stop_words.txt)` file into a dictionary (use the same type of dictionary that you're timing) containing only stop words, called `stopWordDict`. (WARNING: Strip the newline ('\n') character from the end of the stop word before adding it to `stopWordDict`)

2) Process the `hw6small.txt (or WarAndPeace.txt)` file one line at a time to build the word-concordance dictionary (called `wordConcordanceDict`) containing "main" words for the keys with a list of their associated line numbers as their values. The main loop is something like:

```
lineCounter = 1
for each line in the data file do
        processLine( lineCounter, line, wordConcordanceDict... )
        lineCounter += 1
```

3) Traverse the `wordConcordanceDict` alphabetically by key to generate a text file containing the concordance words printed out in alphabetical order along with their corresponding line numbers.

The general algorithm for the processLine (lineCounter, line, wordConcordanceDict ...) function is:

```
wordList = createWordList(line)
for each word in the wordList do
        if the word is not in the stopWordDict then
                if the word is in the wordConcordanceDict then
                        look up the line-#-list value associated with the word in the wordConcordanceDict
                        append the lineCounter to the end of the line-#-list
                else
                        add the word with an associated [lineCounter] list value to the wordConcordanceDict
```

(Note: I 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 with their corresponding line numbers without any other word processing.)

Data Structures (CS 1520) Homework #6 Due: 12/9/17 (Sat.) at 11:59 PM

DICTIONARY ADT COMPARISON

We have 5 dictionary ADT implementations from lab 7: `ChainingDict`, `OpenAddrHashDict` with linear probing, `OpenAddrHashDict` with quadratic probing, and 2 tree-based dictionaries from labs 9 and 10:

- BST-based dictionary implementation, **and also**
- AVL-based dictionary implementation

None of these should need to be modified much. You just use their dictionary operations.

Time your word-concordance application using all five dictionary ADT implementations to complete the following table: (FYI, for WarAndPeace.txt there are about 2,700 stop words and less than 20,000 non-stop words). Both the stop words and non-stop words should use the same type of dictionary (e.g., both are different ChainingDict's). Have just one word-concordance program with 5 pairs of dictionaries with all but one pair commented out, i.e., the dictionary pair you are timing is the only pair uncommented.

Dictionary ADT Implementation Used	Word-concordance Program Execution Time (seconds)
`ChainingDict` (hash table sizes 2**15 = 32768)	
`OpenAddrHashDict` with linear probing (hash table sizes 2**15 = 32768)	
`OpenAddrHashDict` with quadratic probing (hash table sizes 2**15 = 32768)	
Dictionaries implemented using BSTs	
Dictionaries implemented using AVL trees	

DATA FILES - Download hw6.zip file at http://www.cs.uni.edu/~fienup/cs1520f17/homework/ it contains:

- `ChainingDict` in the file `chaining_dictionary.py` and `OpenAddrHashDict` in the file `open_addr_hash_dictionary.py`
- the sample data files for testing your word-concordance program: `hw6small.txt` and `stop_words_small.txt`
- the "real" stop words are in the file `stop_words.txt`
- the "real" data file to be processed by your word-concordance program is in the file `WarAndPeace.txt`

EXTRA CREDIT POSSIBILITIES:

1) Use a better definition of a word that allows words to contain an apostrophe or single hyphens. For example, "it's" and "end-of-line-characters" should each be considered words.

2) Modify `OpenAddrHashDict` dictionary ADT to allow double hashing as a rehashing technique.

3) Modify `OpenAddrHashDict` dictionary ADT to allow the capacity of hash table to double if the hash table load factor exceeds 0.7.

4) Modify the `ChainingDict` dictionary ADT to use a BST or AVL-tree at each hash table "slot" for storage.

SUBMISSION

Submit **ALL necessary files** to run your concordance-production application using the dictionary ADTs as a single zipped file (called hw6.zip) electronically at

https://www.cs.uni.edu/~schafer/submit/which_course.cgi

Include in your hw6.zip file a "results" file (.txt, .doc, .rtf, .odt, etc.) containing the completed table above, i.e., timing results for your word-concordance programming using the various dictionary ADTs.