EECS 233 Programming Assignment #3

Due Tuesday, Nov. 13, 2012 before midnight

Web search engines use a variety of information in determining the most relevant documents to a query. One important factor (especially in early search engines) is the frequency of occurrences of the query words in a document. In general, one can try to answer a question how similar or dissimilar two documents are based on the similarity of their word frequency counts (relative to the document size). A necessary step the in answering these types of questions is to compute the word frequency for all words in a document.

In this assignment, you will write a method wordCount(String input_file) that reads a file (document) and prints out (into another file) all the words encountered in the document along with their number of occurrences in the document. Please use output format such as "(father 30) (fishing 12) (aspirin 45) …". For simplicity, assume any derivative words to be distinct, e.g., "book" and "books", "eat" and "eating" are all considered distinct. Assume that words are defined to be simply strings of characters between two delimiting characters, which include a space and punctuation characters. Assuming that something like "Father's" is two words ("Farther" and "s", because they are separated by delimiters) is OK for our purposes. You can use java class StringTokenizer or String.split() to extract words from an input string to save yourself some programming. String.split will split the string around matches of a given regular expression, e.g. "[^0-9a-zA-Z]", which is more flexible than StringTokenizer. Do not distinguish words that only differ in upper or lower case of their characters, e.g., "Father" and "father" is one word. You can use appropriate methods of the String class handle this easily (e.g., using toLowerCase method).

In implementing wordCount, please use a hash table with separate chaining to keep the current counts for words you have already encountered while you are scanning the input file. Your general procedure would include the following steps:

1. Scan in the next word
2. Search for this word in the hash table
3. If not found, insert the new entry with this word and the initial count of 1. Otherwise increment the count.
4. **If you inserted a new word, check if the hash table needs to be expanded**.

After you scan the entire file, loop through the entire hash table and print out, sequentially in any order you like, the list of words and their counts. Also, report the average length of the collision lists (across all hash slots, so empty slots also contribute).

Please run your program on the same input file you used for Programming Assignment 2. If you skipped that assignment, please refer to it for instructions on how to obtain a realistic input file.

Additional instructions:

1. In implementing your hash table, you can use Java's hashCode function on strings, so that your hash function will be h(word) = hashCode(word) mod tableSize. But you obviously cannot use built-in hash tables like HashSet in Java.
2. Please use separate chaining to resolve collisions in your hash table. Using separate chaining, you do not need to have tableSize to be prime number. Any number will work as long as it is not a multiple of 31 (see lecture for the reason why). For example, starting with tableSize as a power of 2 and then doubling if you need to expand will ensure you do not have a multiple of 31.

Deliverables:

1. Source code including comments necessary to understand it. You just need to turn in *.java files, do not include *.class files.
2. A brief readme.txt file that provides any information the TAs might need to know.
3. Input file
4. Output result: word counts and average length of the collision lists. Your code should output these to the console.
5. Your code should take two strings as inputs. The first one is the input file and the second one is the output file.

Grading:

- ***Operational implementation of the hash table along with a "toy" test file and output produced on a toy file: 75% of the grade. (Important: DO THIS FIRST!)***
- ***Producing output on a real file: 25% of the grade***.