EECS 233 Programming Assignment #3: Hash Tables

Due April 7, 2016 (11:59pm)

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

public static String wordCount(String input_file, String output_file)

that reads the input file (document) and prints out (into the output file) all the words encountered in the document along with their number of occurrences in the document. As in project 2, this method returns a string indicating the outcome of its execution, e.g., "Input file error", etc. If the execution completes normally, the resulting string should have the format: "OK; Total words: <number1>, Hash table size: <number 2>, Average length of collision lists: <number3>". Here, hash table size and the average collision list length reflect the state of your hash table at the end of the input file processing. Your main program that invokes this method would then print out this string onto the console.

For the output_file, please use the following format: "(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's method "split" of the String class to extract words from a string to save yourself some programming. 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 (as a separate class with appropriate methods) 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. Create an instance of the hash table to be used for your task
2. Scan in the next word
3. Search for this word in the hash table
4. If not found, insert the new entry with this word and the initial count of 1. Otherwise increment the count.

In your hash table, if a new word is inserted, you must check if the hash table needs to be expanded. Explain in your comments the condition you used to decide when to expand.

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 final size of the hash table and 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. However, before you try your program on this (large) input file, make sure it works by testing it on a small "toy" test file you yourself create for the testing purposes.

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 HashMap 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. 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;
2. Input file;
3. Output file: word counts.
4. Text file where you cut and pasted the console output of the program (indicating the outcome, the number of words in the document, the final hash table size and the average length of the collision lists).
5. The “toy” test file on which you debugged your program; all the outputs (see steps 3 and 4) produced on the test file.

Grading:

- ***Correct hashing: 30***
- ***Resizing/rehashing as needed (with proper explanation in comments): 20***
- ***Application code utilizing the hash table: 25***
- ***Correctly produced output on a real file: 25***
- ***Missing test file or output reflecting the run of your program on the test file: (-5%)***