Word Games

Project Due: 11:59 PM [redacted] Wednesday, May 1

1.0 Objective

This project is meant to help you develop skills at working with hash tables.

2.0 Project Description

2.1 Basic Program

For this project you will design a program that searches a text file looking for key words, keeps statistics on the results of the search, and prints out these statistics. To make this work, you will need to read all the key words from a file, build a minimal perfect hash table, and then start reading the contents of a second text file and look for these key words. The following sections examine each of these steps.

2.2 The Hash Function

For this project you will be using *Cichelli's Method* to build a minimal perfect hash table. How to build this table is discussed in the next section. Here we describe the hash function itself.
h(word) = (length(word) + g*(firstletter(word)) + g*(lastletter(word))) % size
The following table is a description of each term in this equation:

Term	Description
h(word)	This will be the index into the hash table for the word.
length(word)	The number of characters in the word.
g	g is the value associated with a given letter. The value ranges between 0 and some maximum value. The max value is one of the inputs to the *Cichelli Algorithm*.
firstletter(word)	The character that is in the first position of the word.
lastletter(word)	The character that is at the end of the word.
size	The total number of key words. It is also the number of elements in the hash table.

To compute the hash function for any integer is going to require a number of data structures. Obviously, you must have the array that represents the hash table. This array will be filled with the actual key words. Each key word should appear only once in the hash table. Secondly, you will need an array to store the *g value* of each letter that appears in the beginning or end of a word.

2.3 Building a Minimal Perfect Hash Table

This is one of the major components of your project. You will need to open the key word file provided by the user on the command line and read each of the words into a vector. Once you have them all read, you can start to build your hash table and define your hash function. You will accomplish this by using the *Cichelli Method* discussed in class. The vector that all the key words were initially read into will be the initial word list for the algorithm.

2.4 Counting Key Words
Once this minimal perfect hash table is constructed, you are ready to begin reading the text file and counting key words. This should be a fairly simple process. You will read a line from the text file, break the line into tokens (one token for each word in the line), and then examine each token. To examine a token, simply pass it through the hash function and see if the word is currently in the hash table.
A couple of things to watch for. First of all, you should make all of your comparisons case insensitive. In fact, I suggest that wherever you store all of your strings in either all lower case or all upper case. This will make your life a bit simpler. Secondly, you need to be careful of the fact that punctuation will be included in your comparison tokens. For example, if one of the key words is "me" and you read the following line:
This just isn't for me.
You need to realize that the last token returned from this line is "me." Notice the period is included in the token. "me" and "me." are not equal. To get full credit on this project you do not need to worry about this - you would simply say that there are no key words on the above line.
2.6 Statistics
Another major part of this project is recording statistics. A summary of all the statistics you must keep are presented in the following table:

Statistic	**Description**
lines	This is the total number of lines read by your tester program. Blank lines should not be counted in this total. Only lines that actually have at least one word on them should be counted.
words	This is the total number of words checked (key words plus non- key words). A word does not have to be in the dictionary. A word is considered to be any string of consecutive characters with no white space in between them. In other words both "hello" and "sadflk" are considered to be words by this program.
keyWords	This is the total number of key words found in the text. Words should only be counted if they are an exact match to the keyword given in the file. In other words, "Alabama" is not the same as "alabama". Hopefully, this will make things easier.

When the program is finished reading the text file, it should print out a list of statistics. The output should look exactly like the following:

```
*********************
***** Statistics *****
*********************
Total Lines Read: xxx
Total Words Read: xxx
Break Down by Key Word
    key1: xx
    key2: xx
    key3: xx
```

.
.
.
Total Key Words: xxx
The x's should be replaced with actual numbers and key1,2,3,... should be replaced with the actual key words. The number following the key word is to indicate how many times that key word appeared in the ananlyzed text.

4.0 Program Design Tips

The number one rule about writing a program from scratch is not to write a single line of code until you have developed a good design. Developing this good design is probably the hardest thing to learn how to do in programming. Here are a few of my suggestions on how you should go about designing and writing this project.

1stRead the program through once to get an idea of what is expected. Then, read it again. The second time through you may want to take some notes. At this point, you should have a thorough understanding of what is expected of you. If you need to read it a third time, do so.
2ndDo a very rough design. This basically means figuring out what classes you are going to want to create and what the purpose of each class will be. For this project, I can think of at least three classes you should build. A class that contains your *main()* method. A class for the Cichelli hash table. And another class to hold, calculate, and print the statistics. You may also want additional classes for parsing files or for some other kind of work. It is completely up to you; however, be sure you know why you need each class and what its purpose will be for.
3rdOnce you have a list of your classes, go through each class and determine what functions it will need and what each of these functions will do. This should include information about what parameters each function will take and what type of value it will return.
4thThen go through each function and write psuedo-code describing how the function will flow. This is a very critical step because if it is done properly, your code writing will be much simpler. A good piece of advice at this stage is when you are examining a function, assume all the other functions already work - even if you have not written the pseudo-code for a function. If you did Step 3 properly, you will know how each function should work, what arguments it will take, and what values it will return.
5thThe very last step is to go ahead and write your project up. Do not write all the code at once. Write functions one or two at a time, test them and make sure they work as you expect and then go on to the next function or two. It is always best to start with the lowest level functions first and work your way up. In other words, if you have some function that does not call any other of your functions, that is the one you want to start with. You can then test it and make sure it works (this often requires a little creativity). Once you have all of these functions written, you can then move up the chain to functions that only call these lowest level functions. Once you finish all of these, move up another level. Continue to do this until all of your functions are written.

One of the things you will notice about this design is that Step 4 calls for a top-down approach (describe all of the higher level functions assuming the lower level ones are done and then move down to the next level). Where as, Step 5 calls for a bottom-up approach (write and test all of the low level functions before moving on to the higher level functions). I think if you follow this approach, you should find your programs (for this or any other class) get written much more quickly and with far fewer bugs.

5.0 Running Your Program

Once your program has been compiled, it should be run needs to have as inputs keyWordFile and textFile. The key word file should contain the list of words to search for. The text file is the file you will search looking for the key words. A sample of each file can be found below: has been added in canvas.

6.0 Submitting the Project

You should submit all *.java files that you created that are needed to compile your program. You should not submit any *.class files.
Also, be sure to comment your code well so that it can be easily browsed to see what is going on. Code that is excessively difficult to read will be penalized.
DO NOT FORGET TO SUBMIT status.txt FILE