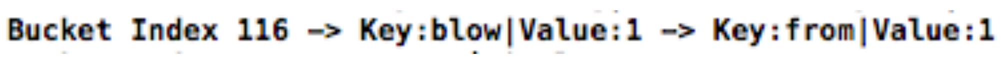
Name: Carlos Carrillo
Date: 02/29/2016
CS 261
Assignment 6

Hash Table Implementation of a Concordance: Written Questions

1. *Give an example of two words that would hash to the same value using stringHash1() but would not use stringHash2().*

   Since stringHash1() literally sums the values of all characters in the input string, the words/strings that produce the same sum will have the same hash. In order for this to happen, the words not necessarily need to have the same collection of letters. For instance, if we assume that a=1, b=2, …, z=26, the words "blow" and "from" should have the same has, and that is exactly what happened in my program:

   ```
   Bucket Index 116 -> Key:blow|Value:1 -> Key:from|Value:1
   ```

   On the contrary, "blow" and "from" are not hash to the same value when using stringHash1(). It is also expected that words with the exact same collection of letters such as "rage" and "gear" get the same hash in stringHash1() but not in stringHash2().

2. *Why does the above make stringHash2() superior to stringHash1()?*

   Because the nature itself of stringHash1() provokes a lower number of buckets with longer linked chains, which may be less efficient in terms of execution time. On the other hand, stringHash2() generates a lower chance for two words with characters that add up to the same value get the same index, which is more efficient when searching for a particular value.

3. *When you run your program on the same input file but one run using stringHash1() and on the other run using stringHash2(). Is it possible for your size() function to return different values?*

   No. The sizeHashMap() function is supposed to return the number that represents the amount of elements added to the table. No matter the way you design your hashing function, sizeHashMap() is always supposed to return the actual number of elements added to the hash table.

4. When you run your program on the same input file using stringHash1() on one run and using stringHash2() on another, is it possible for your tableLoad() function to return different values?

   No. tableLoad() is simply supposed to return the result of the number of hashLinks divided by the number of buckets (ht->count / ht->tableSize). In this case, the hash functions have nothing to do with that result.

5. When you run your program on the same input file with one run using stringHash1() and the other run using stringHash2(), is it possible for your emptyBuckets() function to return different values?

   Yes. It is actually the most important "side effect" of the hashing functions. As mention on the lectures and book, a "poor" hash function, such as stringHash1(), may provoke a lower number of buckets with longer linked chains. So it is completely possible to see different numbers of empty buckets using different hash-functions, even on the same tables with the same size and the same input values. This can easily be seen when comparing the output in our program using stringHash1 (less buckets) versus stringHash2 (more buckets).

6. Is there any difference in the number of 'empty buckets' when you change the table size from an even number, like 1000 to a prime like 997?

   Yes. This behavior is strongly tied to the load factor, which depends on the size of the table. As mention in the book, "*using a prime number decreases the probability of a collision by eliminating the possibility of common factors of the modulo operator resulting in assignment to the same bucket*". Consequently, a prime number should, in theory, decrease the number of empty buckets.

7. Using the timing code provided to you. Run you code on different size hash tables. How does affecting the hash table size change your performance?

   As expected, the time to complete the process of adding keys and updating their values increases, as the size of the table is bigger. This was expected since the more buckets the table has, the more time it takes to find a particular key, which is closer to linear time O(n), rather than constant time O(1).