Justin Scott Lee

CS 32 – 804603677

June 3, 2018

Project 4 Write – Up

My implementation of the DictionaryImpl class tried to change as little as possible from the one given in the specifications. In my implementation, I replaced the list that held all the words with an “open” hash table. The reason I did this was due to the “open” hash table not being limited by a max size and is almost always more efficient than closed hash tables in terms of the average steps needed to find an item within the hash table. The hash table was constructed using an array of buckets, a class that is just a list of strings. In addition to making an “open” hash table, I changed the insert and lookup functions to account for the change in container, and added a hash function.

**Pseudocode for non-trivial function**

hashFunc

given an input of type string

initialize unsigned int total = 1

loop through the string

total = total \* each character in the string

return total % the table size

insert

given an input of type string

remove all non-letters using the given removeNonLetters function

if the word is not empty after that function is called

calculate the bucket to put it in using the hashFunc

place the word in the bucket using the push back function of the bucket, which is just a list

lookup

given an input of type string and a callback function

if the callback function is a nullptr, stop the function by returning early

remove all non-letters using the given removeNonLetters function

if the word is empty after that function is called, return early

calculate the bucket number of the inputted string using the hashFunc

for loop through that bucket using an iterator

if the inputted string and the string the iterator is pointing to are a permutation of one another, then the callback function is called with the word the iterator is pointing to

I thought that the fastest way to find anagrams would be to have all anagrams in one bucket and then when the bucket number was calculated using the hash function, the program would be able to get all the anagrams by checking if there were any words in that bucket number. I realized that just adding the characters would be inefficient use of the hash table, so instead I decided to multiply the character values because it would allow me to fully use the hash table and not cause egregious degrees of clustering to occur. However, this did not fully separate all possible anagrams into one bucket. In order to deal with that, in the lookup function, once the bucket number was computed, I had an iterator that went through the bucket, and I used a handy algorithm called is\_permutation to determine if what the iterator was pointing to was a permutation of the inputted string. This way, the lookup function would only have to search through one bucket’s values instead of all and I would not have to generate permutations but rather just check if the inputted string and the word in the correct bucket are permutations of one another.

I did not have any known bugs, serious inefficiencies, or notable problems with my program.