Kyle Kodani

Project 4 Report

**Description:** I used a hash table that is an array of vectors to represent my dictionary. My hash function creates a number that is a product of prime numbers, which ensures a unique code for a given set of letters (i.e. word). I assigned a prime number to represent each letter, so the key is just the product of the primes associated with the letters. I then modulo the key by the number of buckets (49999) and use that as the position in the array. This ensures that all words with the same key (same letters) are stored in the same bucket (vector). However, certain prime number products (keys) will still have the same value after the modulus operator is applied. This is only a problem when a user looks up a word, because assumedly the lookup function would return all the values stored in the bucket that the word hashes to. To solve this, I sorted each element of the bucket and I sorted the word. I checked these sorted values against each other, and if they were the same, they get printed, if not, nothing happens. This ensures that only anagrams of the input word get printed.

**Pseudocode:**

*void DictionaryImpl::insert(string word)*

remove non-letters from word and convert it to lowercase

hash the word and store the key as an int

use the hash key as the position in the array where the word is stored

*void DictionaryImpl::lookup(string letters, void callback(string)) const*

make sure callback isn’t NULL

remove non-letters from word and convert it to lowercase

make sure string isn’t empty

sort the string, hash the sorted string

for 0 to the size of the bucket that the key corresponds to (loop through bucket)

if the sorted word is the same as the sorted word in the bucket

use the callback function on the word in the bucket

*int DictionaryImpl::hashFunction(const string& word) const*

make a new unsigned long to store the key, set it equal to 1

for 0 to length of word (loop through word)

multiply the key by the prime number associate with the current letter

return the key%size of hash table

**Problems:** The hardest part of this project was choosing a good hash function that would result in few collisions. After learning some basic number theory, I realized prime numbers are very useful when trying to make unique keys, so I used those to hash a string.