Raymond Chen

David Smallberg

CS 32 Spring 2023

Project 4 Report

I decided to implement a hash table using a vector. Each index of the vector stores a list. During my implementation, I thought that a vector of lists seemed to be the most structured way to store anagrams while also allowing for efficient access. Because we are dealing with anagrams, I realized that the hash function would map the anagrams to the same bucket (indexes of the vector in this case) which would result in many collisions. I thought chaining would be the best way to resolve this problem hence why I created a vector of linked lists. Thus, when a user inputs a string of letters, the letters will be passed through the hash function, and the hash value should immediately map us directly to the index of the vector that contains the list of anagrams. We then traverse the list and check if the string at a particular node is an anagram. If it is then we invoke the callback function on it.



**Pseudocode**

void DictionaryImpl::insert(string word)

*remove nonletters before using hash function*

*store the word in some string variable so we can modify it*

*if word isnt an empty string*

*Sort the characters of the word in alphabetical order*

*Get hashvalue of the sorted word*

*Store the unsorted version of the word in vector using hash value*

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

*If callback function is invalid then return*

*remove nonletters from ‘letters’ string*

*If ‘letters’ is empty return*

*sort characters of ‘letters’ in alphabetical order*

*Get hashvalue of ‘letters’*

*Create a copy of the list stored at the hashvalue of the vector*

*for loop to iterate through list*

*Dereference iterator*

*Sort the string stored at each node in alphabetical order   
 If sorted string is equal to ‘letters’*

*Invoke callback function*

**Problems, Bugs, Inefficiencies**

A problem I had was that words that were not anagrams would be printed. For example I would input diet, and the output would be “tide diet edit rummage”. I realized that some words could be chained to the same list despite them not being anagrams of one another. The way I resolved this was by sorting the string inputted by the caller in alphabetical order and comparing it to the sorted version of all the strings in the lists. If the sorted strings matched, then that means they are anagrams of one another, so I would then invoke the callback function. This extra check prevented the program from printing out words that had the same hash value as one another, but were not anagrams of one another.