Joanna Liu

405-957-891

CS32 Spring 2023 - Project 4 Report

1. *A description of your algorithms and data structures (good diagrams may help reduce the amount you have to write), and why you made the choices you did. You can assume we know all the data structures and algorithms discussed in class and their names.*

The data structure I chose to use to organize my dictionary was a hash table because hash tables are known to be very efficient for lookup, which was the primary focus of the project.



To represent the hash table, I had a large vector named “dictionaryHash” to represent a series of buckets, with each vector element holding a pointer to a linked list.

The number of buckets (size of vector) was determined by the Dictionary constructor’s parameter, maxBuckets. By using the maximum number of buckets allowed, it ensures that our load factor is as small as possible.

Each linked list was made up of nodes (called “Items”), with the nodes being represented as a struct with two items, a string holding the word being stored, and a pointer to the next node. If the bucket is empty, the pointer points to a nullptr.

The diagram to the right displays a general visual depiction of my hash table representation.

In order to find the bucket that an inserted word belonged in, I translated the word into an integer using the STL hash function provided in the library <functional>, std::hash<string>(). Before hashing the words, I would sort them by their letters so that words consisting of the same letters would be stored in the same bucket, making the lookups for anagrams much quicker, as you would only have to search through one bucket. The bucket that the word would go into was determined by the equation (key) % (total buckets).

1. *Pseudocode for non-trivial algorithms.*

DictionaryImpl::~DictionaryImpl() {

For each element in the vector dictionaryHash,

Set a temporary Item pointer, deleter, to where the vector element points.

While deleter is not pointing to a null pointer,

Set a temporary Item pointer, next, to the Item after the Item deleter points to.

Delete the node deleter points to.

Assign deleter to next.

Once all the Items (nodes) in the linked list are deleted, set the vector element to nullptr.

}

void DictionaryImpl::insert(string word) {

Remove any non-letters from word.

If word is now empty,

Return.

Sort word so that it is in alphabetical order.

Translate the sorted word into an integer key using std::hash.

Find the bucket this word should be sorted into using key % dictionaryHash.size().

Create a new Item and assign it’s m\_word to word.

Insert the new item into the beginning of the linked list at the bucket location found earlier.

}

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

If the function passed (callback) is invalid, return.

Remove any non-letters from letters.

If letters is now empty, return.

Sort letters into alphabetical order.

Translate the sorted word into an integer key using std::hash.

Find the bucket this word would be in using key % dictionaryHash.size().

Set a pointer called current to the bucket location found previously.

While current is not a nullptr,

Get the word from the Item current is pointing to

Sort this word alphabetically.

If the sorted letters (from before the while loop) match the sorted word,

Call callback on the unsorted version of the word in the Item we are at.

Move current to the next Item.

}

1. *A note about any known bugs, serious inefficiencies, or notable problems you had.*

I did not encounter any bugs, serious inefficiencies, or notable problems. Something I did notice though, is that my code has around a 2 ms margin of how long it takes to run when using the [ curl -s -L http://cs.ucla.edu/classes/spring23/cs32/Utilities/p4tester | bash ] command on the seasnet server, ranging from taking ~13 ms to ~15 ms.