Zheng Wang (404855295)

COM SCI 32

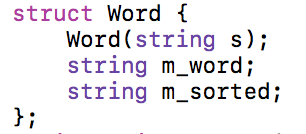
Professor David Smallberg

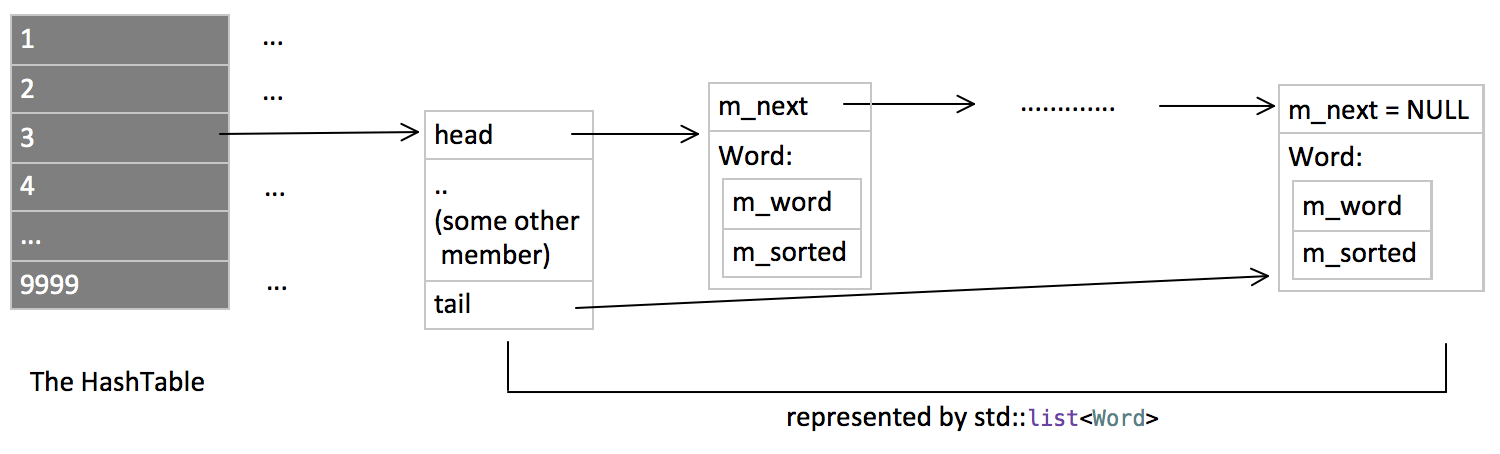
June 4, 2018

**Project 4 Report**

Design and Description:

The data structure I choose to store the value for the dictionary is hash table of size 10000. The hash table is represented by an array of list<Word>\* (a pointer to a list of Word). The index to place or look for a Word is determined by a hash function named DictionaryImpl::KeyGen. The Word is a private structure in DictionaryImpl that store the words obtained in dictionary or through DictionaryImpl::insert function. Below the implementation of the structure Word:



Word(string s) is the constructor of Word. string m\_word is a string that store the actual word (e.g. if you call Word(“cba”), then m\_word is set to “cba”). string m\_sorted is the string that store the sorted word (e.g. if you call Word(“cba”), then m\_word is set to “abc”). So overall, the data structure looks like this:

For my hash table, I decide to use a size of 10000, which is sufficient and less than the size limit specified in the spec. As mentioned above, each bucket is a pointer to a list that will actually hold the words. (AKA type list<Word>\*). Inserting word to the dictionary will call list::push\_back() function to add the word to the list.

Pseudocode and Algorithm Complexity:

unsigned int DictionaryImpl::KeyGen(string& word) const

sort word

initialize unsigned int h to be 2166136261U

for each char in word

h is assigned h + the ASCII code of that char

h is assigned h \* 10012879

return h mod 10000

(complexity: O(N logN), where N is the length of the word)

void DictionaryImpl::insert(string word)

remove non letter from word

if word is not empty

copy word to temp

go HashTable’s KeyGen(word)

if it's a nullptr, create a new list<Word>

push back word to the list

else if word is empty

leave the function

(complexity: O(N logN), where N is the length of the word, due to calling KeyGen function)

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

if there is no callback

leave the function

remove non-letters from letters

if letter is empty

leave the function

copy the pointer at HashTable’s KeyGen(letters) position

if that pointer is nullptr

leave the function

from the beginning of the list to the end

if the sorted word is same as sorted letter

call function callback(unsorted word)

(complexity: O(N logN), where N is the length of the word, due to calling KeyGen function)

Constructor and Destructor

DictionaryImpl::Word::Word(string s)

using s to initialize m\_word

sort s

assign s to m\_sorted

DictionaryImpl::DictionaryImpl()

for all elements in HashTable array

assign it to nullptr

DictionaryImpl::~DictionaryImpl()

for all elements in HashTable array

delete it if it is not nullptr

Other Non-member Function

void removeNonLetters(string& s)

assign an iterator “to” points to the beginning of s

for all elements in string

if the char is a letter

force it to lower case and assign it to “to”

increment “to”

erase anything after “to”

Issues and Difficulties

The most difficult part is to find the hash function KeyGen. My original hash function is not very efficient and cost a long time to finish search. This problem is solved after Week 10, Monday’s lecture. Using a modified version of Professor Smallberg’s general hash function, which is slightly more effective than the original version during the test, the program is able to pass the test and finish the search within 10 ms. Since the hash function requires sort, the program may take a much longer time when dealing with particularly long letter sequence (say 10000-letter-long string).