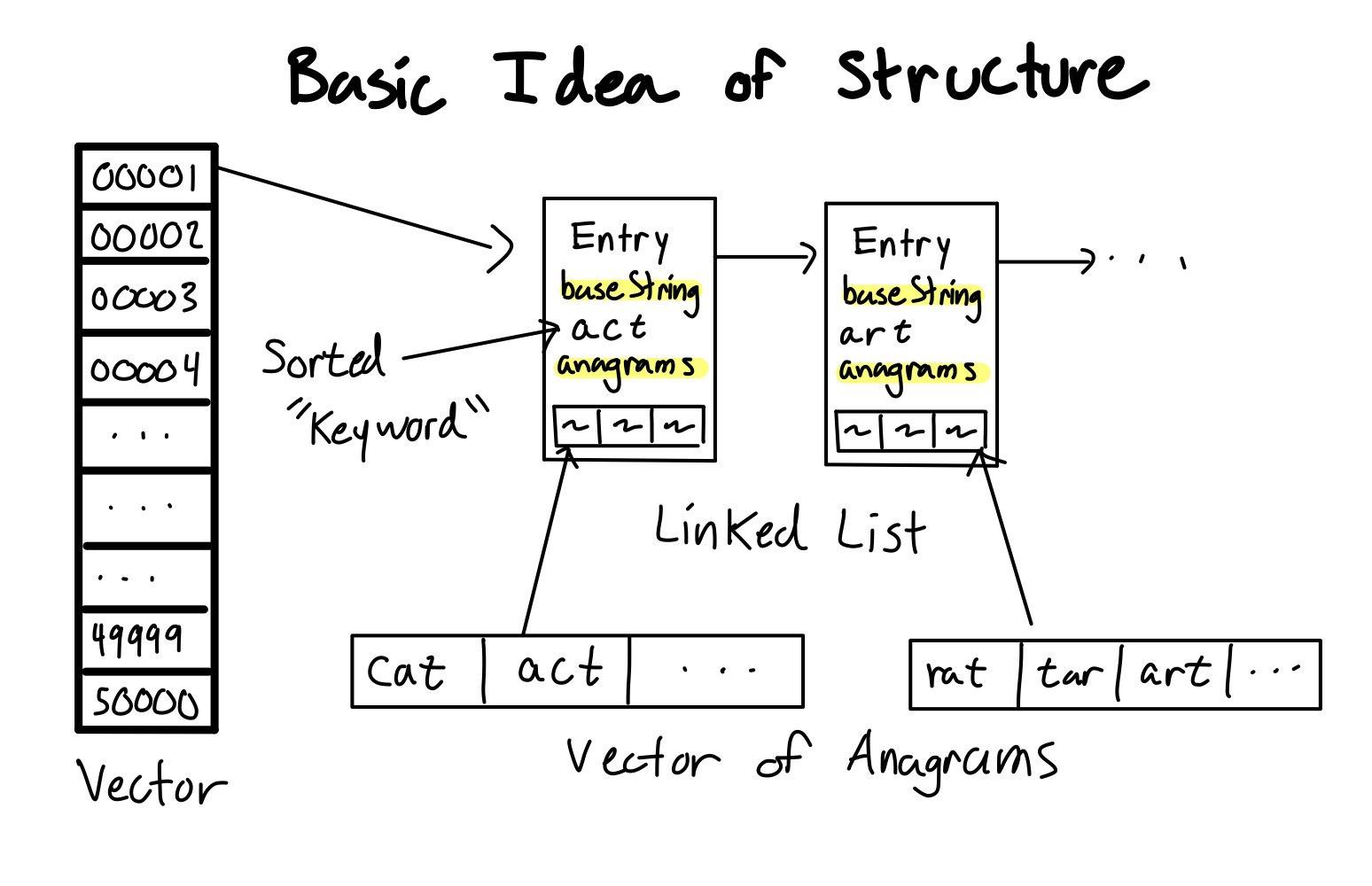
Brian Chiem

705973390

CS32 Project 4

1. Drawings, Data Structure, and Algorithms

The basic idea behind my data structure is a hash table built using linked lists of a structure I created called Entry, where the linked lists are spread out in a vector using hashing. The Entry structure consists of a base string and a vector of all anagrams that correspond to that base string.

Essentially, instead of searching through the whole library for possible anagrams which takes a long time, instead, we sort possible anagrams together when the dictionary is initially inserted, meaning that when given a string we look for the corresponding bucket, and then the corresponding entry and then we just output the anagrams in from the entry, instead of having to go through the whole dictionary.

The major concept and reason this works is based on how we can sort strings. In conjunction with the already written removeNonLetters function, we can sort a string from least to greatest, meaning that for EVERY string that has the same letters, it will ALWAYS be sorted from least to greatest the exact same way every time. For example, “rat,” “art,” and “tar” all share the same sorted string of “art.” This means that we can use “art” as a base string for our entry because it has a common relationship with its anagrams.

1. Pseudocode

void DictionaryImpl::insert(string word)

{

remove any non letters from word

keep a copy of the original word

sort word from least to greatest

if the word isn't empty

get the hash of word

get the bucket from this hash

for every entry in the bucket

if word matches a baseString of an entry

put this word into the entry's anagrams

otherwise create a new entry and put the original word in

the entry's anagrams, and put word in the baseString

}

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

{

check if callback is a valid function

remove all non letters from letters

sort letters from least to greatest

get the hash from letters

get the bucket from this hash

for every entry in the bucket

if letters matches a baseString from an entry

call callback on every anagram in the anagram vector of that entry

}

1. Note

At first, I had trouble thinking about where to even begin. My first attempt was to just convert the Dictionary into a hashtable purely on the unmodified strings. This improved time by a bit, but was definitely still really slow. I kept on trying to utilize generateNextPermutation, however it wasn’t until a long time later that I realized the reason it was so slow was because of this function. I realized this when I tried one of the examples “Veronica Snot.” It was so slow because the string is so long that there are so many different ways to order the string. That was also where I realized that I should sort it by least to greatest, because that is just one ordering that is consistent for all anagrams.