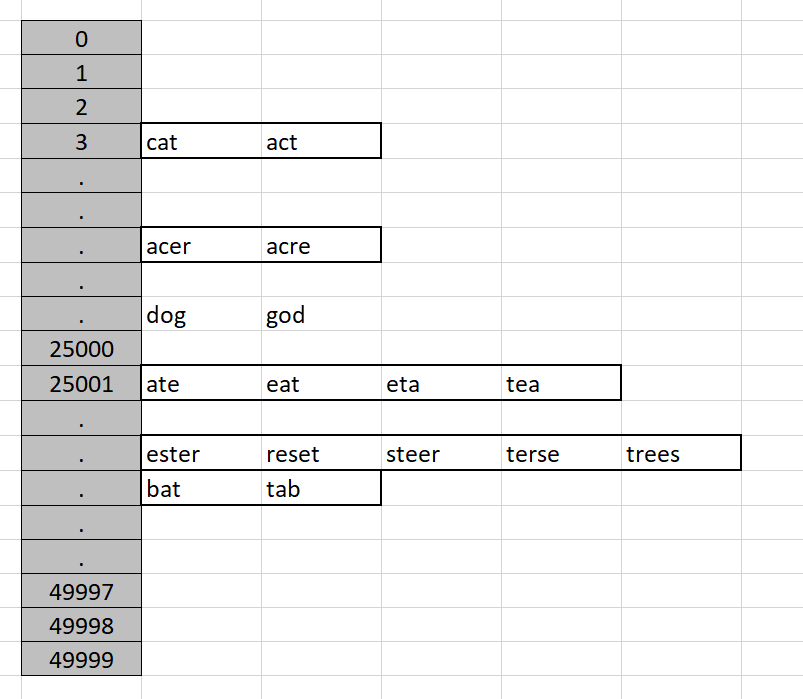
Angel Jimenez

CS 32

Spring 18

604-667-905



**Description:**

My method creates an array of vectors. The vectors hold the word data and the array address is the unique key generated by a series of prime multiplications mod 49999 (because of the array limit). The hash table has a limit of 50,000 buckets as per the spec. I chose a vector because there is no set size to them and they can become as large as they need to be, so when choosing what data structure to place inside the array. It was either a vector or a list for this reason. Retrieval rates weren’t noticeably different.

The spec suggests we can use sort from <algorithm>, this is what allows me to look through the vectors and find the words I need to print. To start, I did some reading on anagrams, I learned about anatree’s and prime tables. There was an interesting article about multiplying the prime associated with a letter in the alphabet. The most common letters being smaller primes. However, I felt that it didn’t make it a difference due to the method I went about implementing my anagram solver. I generate a unique key by multiplying several primes together then taking the remainder after performing a mod 49,999. This is to prevent collision. Also, I chose a hash table because we just learned about it in class and several websites suggested it was an extremely fast method for retrieving data. I read about open hashes as well.

What my program does differently than the inefficient method is it generates hash keys to given words, then pushes them into the hash table. When you want to look them up, you generate the key for that particular string of characters, then you call the hashtable to that key. You loop through the values inside the vector (using a const\_iterator) and compare them to find the ones to print. If it meets the criteria, you print.

**Pseudo-Code:**

hashFunction(string word) // generates hash keys  
{

keyGen prime constant  
for loop until word length

{

Update keygen by multiplying old value to new value

}

Return final keygen mod bucket\_limit

}

DictionaryImpl::Insert(string word)

{

Remove non alphabet letters

If the word isn’t empty

{

Generate a hashkey

Push word into the array at the hashkey address

}

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

{

If callback is nullptr, return

Remove non alphabet letters from string

If the string is empty, return

Generate hashkey

Create vector iterator

Loop through the vectors stored at the hashkey until you reach end of vector

{

Generate two strings storing both string objects

Use the <algorithm> sort function (begin(), end())

If the words are equivalent, callback the object the iterator points to

Increment the iterator

}

**Bugs, Inefficiencies, Notable Problems:**

I chose vectors because it’s the only one I could get to work efficiently. I tried creating a vector of vectors, a list of vectors, a vector of lists (which worked out nicely, but some of the implementation was costly), a vector of node pointers which more vectors and information inside of it. I’m not sure if my implementation is as fast as it can be given the data structures I used and created. I know where the complexity is getting worse, but I don’t have the time to fix it. The complexity can be improved in the lookup function. I had to call the sort algorithm, which increased my complexity, but it was an easy solution given a short amount of time.

I had a lot of problems doing this, all approaches seemed simple until you got to the part where they deviated in theory. For example, iterating through a vector of vectors or a list of vectors etc. It got tricky and ultimately I started over like 5 times. It was quite frustrating because I ended up commenting so many lines of code because of certain implementations had strange restrictions or they weren’t efficient enough.