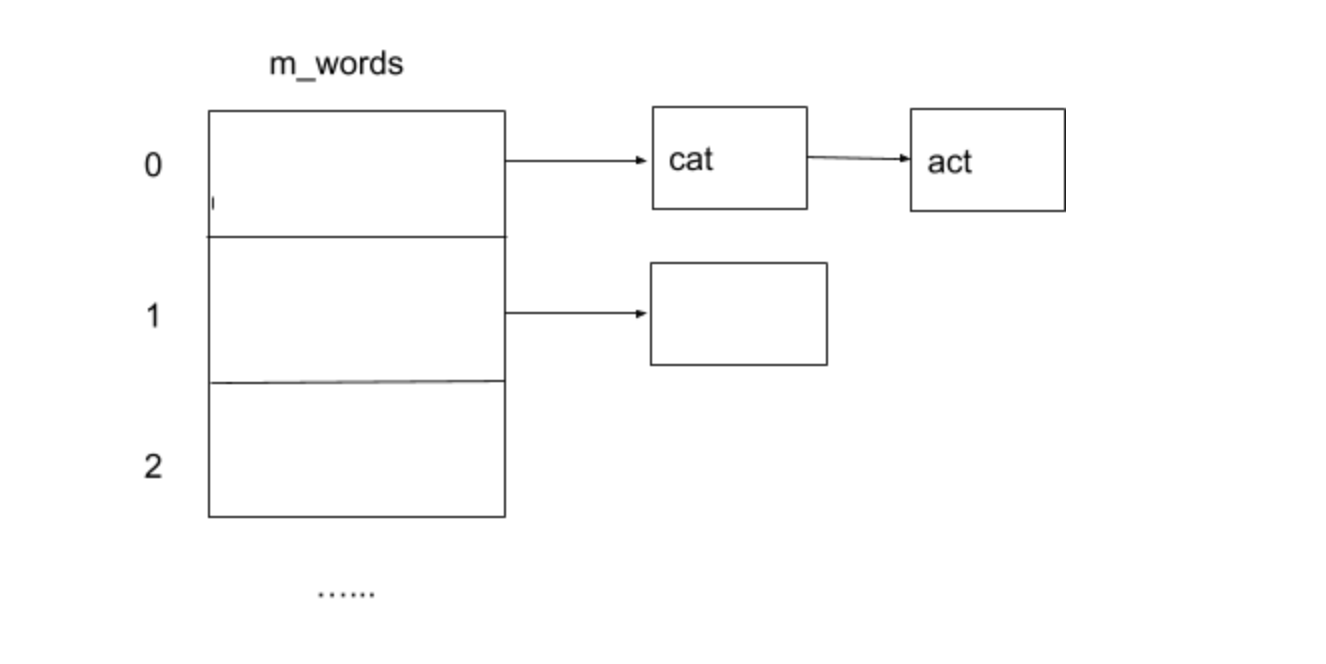
**Project 4 Report**

Casey Taylor

1.A description of your algorithms and data structures 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.

For this project, I chose to implement a hash table in order to minimize the time complexity of lookups in my data structure. I chose this because the time for a lookup in a hash table is constant time, or O(1). Once the algorithm identified the proper bucket within which to search, the lookup within my reasonably long linked list is very efficient.



The photo above shows a very crude photo of my implementation, in which I decided to work with an array of linked lists (using the C++ std::list type). I allocated an array with 49,999 slots, the largest prime number allowed by the spec. I used prime numbers whenever possible through my code to minimize collisions.

Each linked list is populated with the contents of its corresponding “bucket,” or slot in the array. My hash function does the job of splitting up each word into its proper bucket based on a unique hash value, which I obtain by assigning each character in the string a prime number and multiplying those values together. I chose to use both prime numbers once again, as well as multiplication (rather than addition), to reduce collisions and ensure that my distribution is uniformly distributed. I then rely on the mod % operation to scale this value down to the size of my array.

2. Pseudocode for non-trivial algorithms.

void DictionaryImpl::insert(string word)

{

Remove the non letters from the word

If the word is not empty

Use the hash function to map the inputted word to its corresponding bucket

Insert the word into the linked list located at that bucket

}

unsigned int DictionaryImpl::myHashFnc(const string &toHash) const

{

Create an array containing the first 26 prime numbers

For each character in the inputted word

Calculate its hash value by assigning each character a prime number and multiplying them together

Compute the hash value % size of the array to get the proper bucket number

Return the bucket number

}

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

{

If callback function is null

Return

Remove non letters from the inputted word

If the word is empty

Return

Use the hash function to map the inputted word to its corresponding bucket

Access the linked list held in that bucket of the array and store in local variable

For each word in that linked list

If the length of the current word equals the length of the user inputted word

Call the callback function on that word

}

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

There are no known bugs in my program as far as I can tell.