Sagar Doshi

UID: 604901376

TA: Abani, N.

Dis 1A

Project 4 Report

**General Strategy**

The general approach I used to solve this problem was to use an open hash table.

I made this choice because I realized it was possible to relate the ASCII values in the characters of a string by some mathematical formula to the index of an array. Using the random access functionality of an array would then reduce search time to O(1), and this has the added benefit that anagrams all generate the same key as they have the same letters, so to get all the anagrams, all you have to do is iterate through the linked list at the specified key.

To implement the hash table I created a node class to serve as my linked list (I felt that the STL library list was overkill).

*//Linked list for each bucket*

*//Each node has a value and key (in case we want to work backwards and get the key of an anagram without utilizing the "Secret" hash function.*

*struct Node {*

*Node(string str, int k) : value(str), key(k)*

*{}*

*Node \* next;*

*string value;*

*int key;*

*};*

To make the actual hashtable, I made an array of Node pointers and a private hash function.

*Node\* hash\_table[MAX\_HASH\_SIZE];*

*int hashfunction(string & str) const;*

Because I changed the data structures being used I wrote a custom constructor and deconstructor

DictionaryImpl() – *Set all the nodes in the hashtable array to nullptr*

~ DictionaryImpl() – *Go through every index of the array, and at each index delete the linked list by iterating through the nodes with pointers*

**There are no known bugs in my program, and when I ran it with the cs32.seas tester, it passed both correctness tests and was running at about 10-12 ms.**

**Pseudocode for nontrivial Functions**

**bool matchingCharacters(const string & s1, const string & s2 )**

*Create an integer counter and empty string called already\_checked*

*Iterate through s1, looking at the character at each index*

*Check to see if the character is in s2, and increment the counter if it’s in s2*

*Return false if the counter is 0*

*Add the checked character to the already\_checked string, so on later iterations, can skip if a character has already been checked*

*Reset the counter and continue*

*Return true as no extraneous chracters were found.*

*//This hash function is somewhat quick, and does an OK job of distributing the string evenly through the buckets, I’m sure it can be improved.*

***int DictionaryImpl::hashfunction(string & str) const***

*Remove the non letters of the passed string*

*Initialize an array of the first 26 primes*

*Initialize an integer key to 1*

*Iterate through the string, and subtract each character from the character ‘a’ to obtain the corresponding index in the primes array*

*\*= the key by this prime number at that index*

*Return the key % MAX\_HASH\_TABLE\_SIZE*

***void DictionaryImpl::insert(string word)*** *Use the hash function to get the key of the word*

*If the bucket at that corresponding key is empty,*

*simply add a new node with the passed string as its value, and the key of the passed string as its key. Set the node’s next to nullptr.*

*If the bucket at that corresponding key is not empty*

*Iterate to the end of the linked list, and add a New Node with value being the passed string and key being its corresponding key at the end of the list. Set this new node’s next to null ptr*

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

*If the function passed is a nullptr or if the passed string is empty*

*Return*

*Go to the Node at the index corresponding to the key obtained from calling the hash function on the passed string*

*If the size of the passed string equals the string in the linked list, and matchignCharacters() returns true*

*Callback(passed string)*

*Otherwise just go to the next node in the list as you have found one of the few cases of two strings w/differing size or characters that happen to have the same hash key*