Lab Report 1

CS 2303 Data Structures

Spring 2019

Miriam Ramirez

**Introduction**

This lab consisted of two parts. Part one required us to read a text file and using the text file, read and save each word from the file into a set. With the words we were to come up with a solution (recursively) that would create the anagrams of any given word. The anagrams would have to be displayed alphabetically along with the time it took to run the code. Part two required us to implement two things: 1) avoid creating the same anagram if the word has duplicate characters, and 2) stop the recursive call if the word does not have a prefix from a prefix word set created.

**Proposed Solution Design and Implementation**

I began my code by downloading the word file and reading it. I struggled to create a set given the file. I would print the set and not only would the set print out the words out of order, but they would also print out with a \n along with punctuation. I decided to switch from a set to a list so that perhaps I could manipulate it easier. I was also presented with \n and punctuation when I printed out the list, however, this time it printed out the words in order which is what I wanted.

Since I want the user to enter a word of their choice and I’ll be checking if that word is in the word file, I decided to add a binary search method that given the list and the input, it will search for the word much faster than if I had implemented a for loop to check all the words which are too many! If the word is found, instead of returning an index or just simple a text saying the word is found, I decided to return my makeAnagram method with a parameter being the key word (IF it is found) and another parameter being the index of the first character in the word.

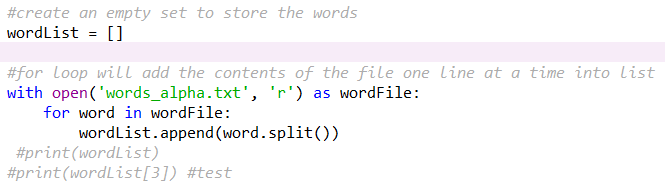
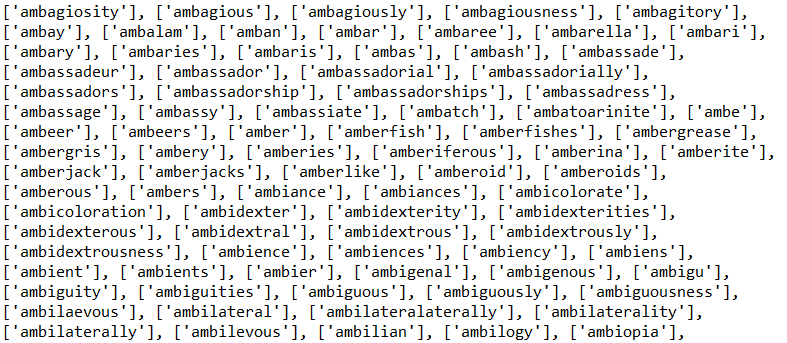
To create the anagrams, I created a method called makeAnagram that took in two parameters, the word we wanted to create the anagrams for and the index of the first letter to create the combinations. The purpose of this method is to take in an input of a word and create n! combinations, n being the length of the string. The base case for my method will be, if the index is less than the length of the word minus one; until index reaches the position of the last character, then print out the word. After the if, a new string variable will be created of only one letter which will have the remaining letters added to it in its different arrangements.

I had no idea how to implement a timer in order to print how long my methods took to run.

**Experimental Results**

As presented in the screenshot above, the words in the list would print out in order, but with a \n along with the word.

Then I added the split() method to try and fix the problem I had been having and that is what seemed to fix it. The words now print out without any addition to them.



**Conclusions**

I cant say that I learned much other than I should definitely not procrastinate and if I am having trouble, I should seek help. However, applying a binary search method for a file of this length made me appreciate and further put into perspective the importance of algorithms that can execute in a short amount of time rather than going through a file with about 300,000 plus words in linear time.

**Appendix (Source Code)**

