**CS 2302 Data Structures**

**Spring 2019**

**Lab Report #3**

Due: Sept. 6, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab, we used sets and recursion to find the anagrams of a word in a given word set. It is important to focus on recursion theory and practices and to consider the runtime for this recursive solution.

**Proposed solution Design**

I decided to define a “recursive\_anagram” method that will take in three parameters: a list of ordered letters, a word set, and a list of anagrams. This method will check if the word set is empty, and If it is not, it will compare the ordered letters to a sorted letter list of a popped word from the word set. If the two letter lists are the same, that word from the word set is an anagram of the parameter letters and it is added to the anagram list.

In the main method, a file of a list of words is read and saved as a set. The user is then prompted to enter a word for which they want to find the anagrams. Then, that input is sorted into a list of letters alphabetically. The recursive solution is called, and the return value should be printed. I also added a few lines to calculate the amount of time passed in the recursive call.

**Experimental Results**

The results return an error stating “maximum recursion depth exceeded in comparison.

**Appendix:**

import time

def recursive\_anagram(letters, wordset, anagram):

if len(wordset) == 0:

return anagram

else:

word1 = wordset.pop()

if letters == sorted(word1):

anagram.append(word1)

return recursive\_anagram(letters, wordset, anagram)

file = open

with open("alpha.txt", "r") as file:

lines = [line.strip() for line in file]

Wordset = (set(lines))

print(Wordset)

input\_word = input("Enter the word you want to try")

word\_letters = sorted(list(input\_word))

print(word\_letters)

anagrams = list()

start\_time = time.time()

print(recursive\_anagram(word\_letters, Wordset, anagrams))

print("--- %s seconds ---" % (time.time() - start\_time))