Laurence Labayen

9/7/2019

Lab #1

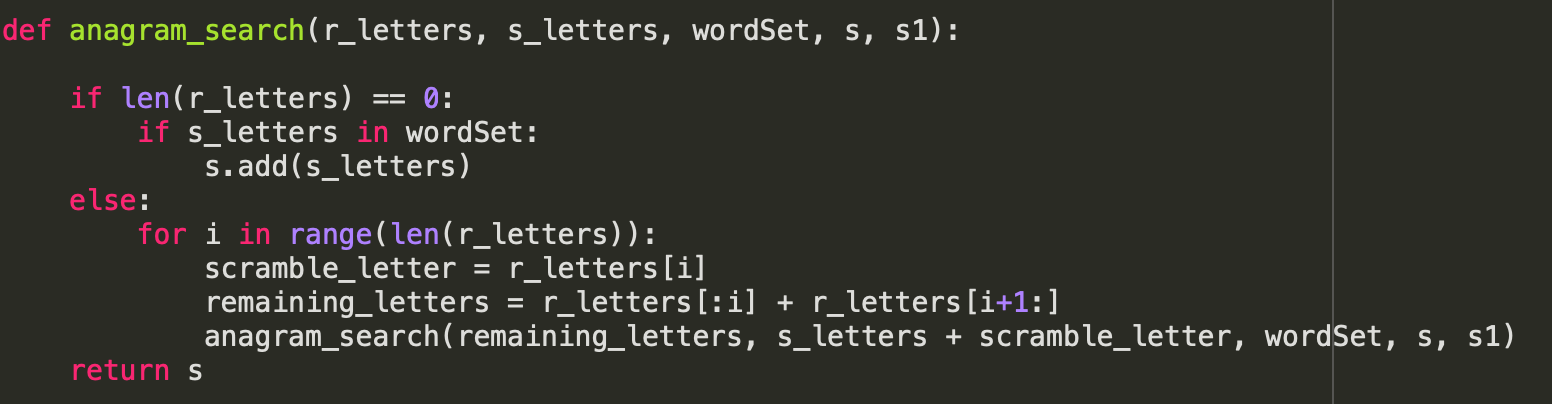
CS 2302 10:30

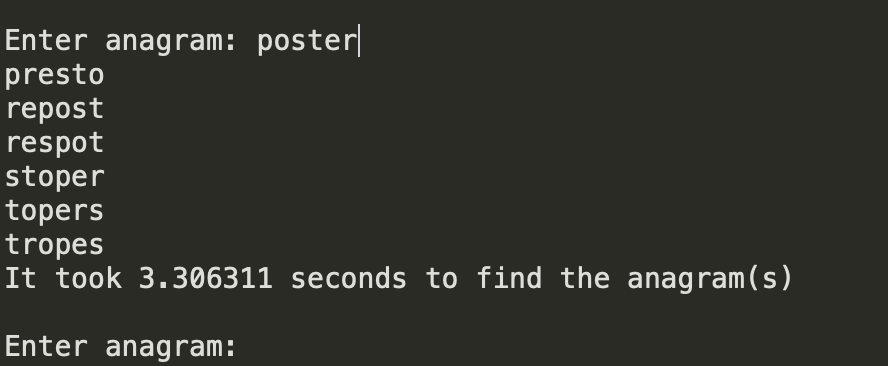
Description: For this lab, we were asked to take an input string of any word and permute the given string to all possible combinations and find anagrams of the given string from a text file with 4666000 words from the english language.

First, I needed to open and read a file using open() and read() functions, along with using splitlines() to get each word in a list. Then, I defined a recursive function called anagram\_search to permute the input string to all possible combinations, compares the matching combination and stores it in a set that is returned . Inside the function, I set a base case to where if the length of r\_letters is 0, and if s\_letters is in the word list from the text file, it adds that word to the set of words that are an anagram for the input string. Otherwise, it goes into a for loop that recursively permutes the word until it gets to the base case. The function search is then called to get an input string from the console and to start a timer. The first conditional statement handles if the input is not valid, such as if the input is blank or starts with a space. If the conditions are met, one of the anagram search functions is called which returns a set of words that are matched with the permutations to the list of words/dictionary. Another conditional statement is applied to check if there are any anagrams in the set. If it has 2 or more words in the set, the first one is omitted (input string) and the set is printed line by line using a for loop. Lastly, the timer is stopped and the difference of end and start is printed out.

For the second part of the lab, I needed to optimize the speed of finding the anagram. I created a function called optimized\_anagram\_search. Similar to the original anagram\_search function, the input word is permuted to get all possible combinations but with added conditional statements that check if there are any duplicate characters in the string. I also had to create a new function that gets all the prefixes in the word\_alpha.txt and puts it in a set. Using this new set of words with all the prefixes, I added another conditional statement to check if the remaining letters are in the prefix set before calling the recursive function.

Experimental results:





Conclusions:

This was my very first Python program and transitioning from Java to Python was very difficult, especially with this lab assignment. Although I feel like I didn’t feel like my optimized version of the program was correct, I learned a lot from this lab. I’m familiar with solving recursive problems from my previous class but having to learn a new programming language added more barriers to solving this lab.

Appendix:

"""

Created on Wed Sep 4 12:20:27 2019

@author: Laurence Justin Labayen

"""

wordSet = list(open("words\_alpha.txt").read().splitlines())

import time

def prefix():

set2=set()

for j in range(len(wordSet)):

w=wordSet[j]

for i in range(len(w)):

set2.add(w.strip(w[i:]))

return sorted(set2)

def anagram\_search(r\_letters, s\_letters, wordSet, s, s1):

if len(r\_letters) == 0:

if s\_letters in wordSet:

s.add(s\_letters)

else:

for i in range(len(r\_letters)):

scramble\_letter = r\_letters[i]

remaining\_letters = r\_letters[:i] + r\_letters[i+1:]

anagram\_search(remaining\_letters, s\_letters + scramble\_letter, wordSet, s, s1)

return s

def optimized\_anagram\_search(r\_letters, s\_letters, wordSet, s, s1):

if len(r\_letters) == 0 and s\_letters in wordSet:

s.add(s\_letters)

else:

for i in range(len(r\_letters)):

scramble\_letter = r\_letters[i]

remaining\_letters = r\_letters[:i] + r\_letters[i+1:]

if scramble\_letter not in remaining\_letters and remaining\_letters in set2:

optimized\_anagram\_search(remaining\_letters, s\_letters + scramble\_letter, wordSet, s, s1)

return s

def search(anagram\_finder):

set1 = set()

word = input('Enter anagram: ')

start = time.perf\_counter()

if len(word) > 0 and word.startswith(' ') == False:

anagram\_list = sorted(list(anagram\_finder(word, '', wordSet, set1, set2)))

if len(anagram\_list) >= 2:

anagram\_list.remove(word)

for i in range(len(anagram\_list)):

print(anagram\_list[i])

else:

print('No anagrams for ' + word)

end = time.perf\_counter()

print('It took ' + str(round((end - start), 6)) + ' seconds to find the anagram(s).\n')

else:

print("Bye! Thanks for using the program!")

set2=prefix()

search(anagram\_search)

search(optimized\_anagram\_search)

Academic Honesty Statement

“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”