Nathan Aun

6/17/19

Lab #1 - Recursion

CS2302 – Data Structures

Summer 2019

Introduction

The problem posed for this lab was to write a program that, when given a word, will output all the anagrams of the given word and tell the user how long it took to do so. Anagrams are essentially when the letters in a word can be rearranged to produce an entirely new word. An example of this would be the words “cat” and “act”. The strict requirements given for this lab were that there must be a recursive function used to find the anagrams, the anagrams must be displayed in alphabetical order, and that there cannot be any duplicate words within the list of anagrams.

Proposed Solution

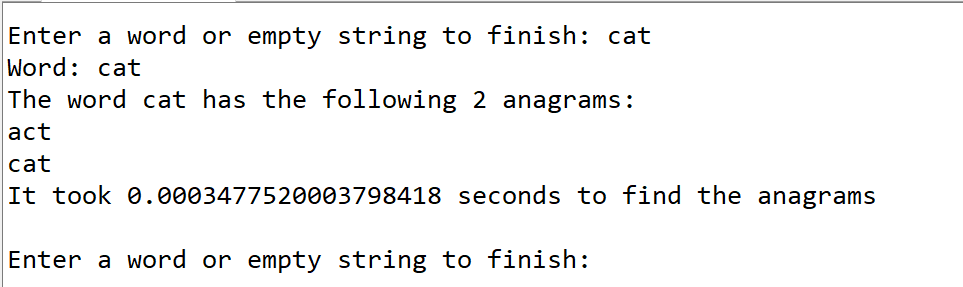
The solution that I decided to go with involved splitting the program into two distinct methods, one being called “permutations” and the other called “output”. The “permutations” method is a recursive method that takes two strings, the first being the initial word to scramble and the second being the letters after they have been taken from the initial word. This allows the method to eventually work its way through all the possible permutations of the given word and it will add every word that is generated to a set called “permutationSet”. Having this in a set is important because it will allow us to compare with other sets with a smaller runtime than what would be possible with something like an array or list. The base case for the recursive method is basically that when there are no letters left in the initial string, it will add the stringSoFar to the set. The recursive call happens whenever there are letters still left in the string, and it will do n recursive calls every time it reaches the for loop, where n is the length of the string. Every recursive call should have a different group of letters for the initial string and a different group of letters for stringSoFar and this will allow the program to permutate all the different words. As far as runtime is concerned, this is not a perfect solution because just the permutations part of the program alone has a time complexity of O(n!), where n is the length of the word. At the same time, however in the context of this particular problem it might not be as big of a deal as it could in other cases, since most words are not incredibly long, but this is definitely a less than ideal runtime for the program and words 11 letters or greater take a fairly unreasonable amount of time to find their permutations.

A possible solution to this problem could be to implement our word bank using a hash map, with an alphabetically sorted string of all letters as a key and with all words that use those letters as a list in the data side of the hash map. This way, the only actual things we would need to do while the user is using the program would be to sort the letters in their given string in alphabetical order and compare it against our key, which would significantly reduce runtimes and make the program much better. There are a few problems with this solution, including the fact that it would make starting the program take much longer since every word in the word bank would need to be sorted, compared, and put into a hash map but even that wont take O(n!). The main reasons I did not implement this solution is because we have not formally been taught hash maps and this solution is non-recursive which does not follow the restrictions given for this lab.

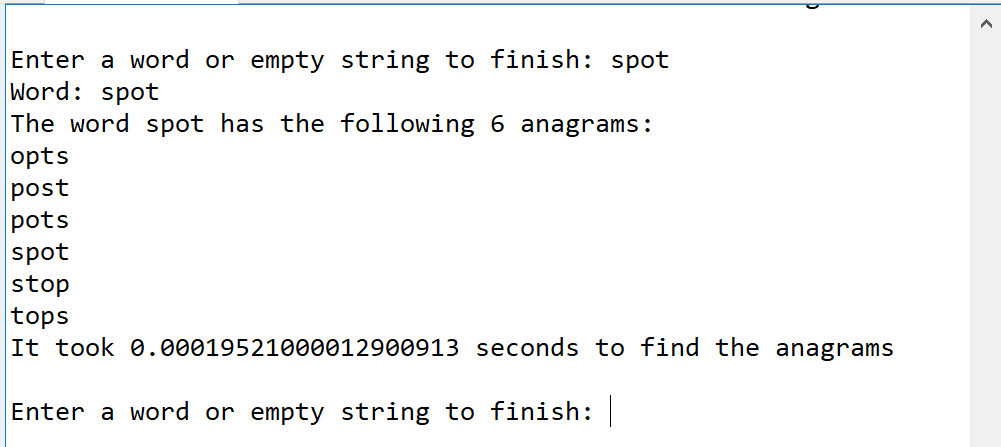
The “output” method handles all the input and output of the program. It first asks the user to input their word and checks to make sure there is a word there. If there is, it records the time when it first gets the word, then it takes this word and puts it in the “permutations” method. After this, it runs an intersection on “permutationSet” and our “wordBank”, which was declared outside of these two methods, and stores where these sets intersect as a new set called “realWords”. “realWords” is then sorted to be in alphabetical order, then printed out. It also records the time, finds the difference between the first time call and now to get the actual time it took for the program to run and prints that out. It then clears the “permutationSet”, clears the realWords set, and asks the user again for a word and will keep doing this until the user inputs an empty string. Once it detects an empty string it will print the “goodbye” statement that is written, and the program will end.

Experimental Results

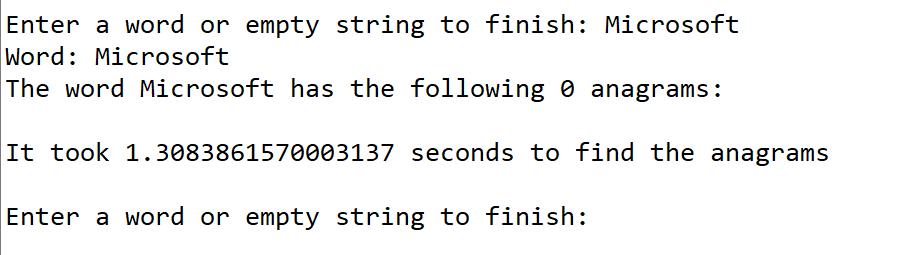
When running my code I decided that my different test-cases would be words of varying length. The first word tested was “cat” which has 2 anagrams, “act” and “cat”. The output the program gave was as follows:



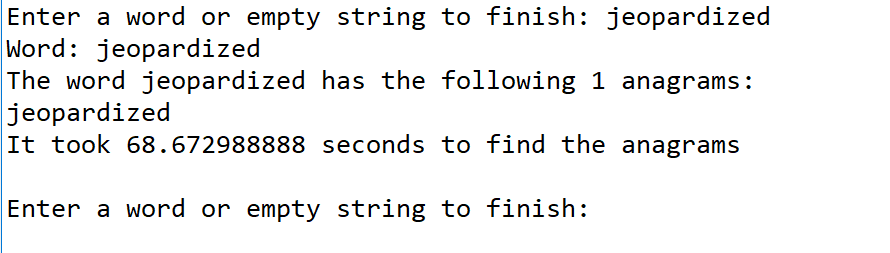
The second word I decided to test was “spot” since it was one of the examples given to us. The output was:



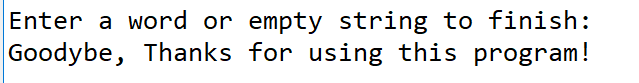
The third word I decided to test was “Microsoft”, which technically isn’t a word, so the output was:



At this point time was starting to take dramatically longer the more letters that were in a word. This output was thousands of times slower than the 4-letter word spot that we just checked and this word was only 9 letters long. The last word I decided to test was jeopardized, which is a 11-letter word, in order to demonstrate how poor the runtime can get with longer words. The output was:



I did attempt to run jeopardizing, a 12-letter word, through the program but it seems like either my computer or the program cannot handle words of this size because after 2 hours the program hadn’t finished but by my estimates it shouldn’t have taken longer than 20 minutes. Finally, this last test just shows that the program actually does finish when an empty string is input



Conclusions

I feel like this lab has really helped me have tighter grasp on both recursion and programming in python as a whole. It’s also pushed me to learn about hash maps on my own, since sets have a similar functionality to them and I got curious. This was a really interesting lab and I think on my own time I will look into implementing my other solution and compare it to this one to see which one would be better.

Appendix

**Source code is as follows:**

# -\*- coding: utf-8 -\*-

"""

Course: CS2302 Data Structures

Author: Nathan Aun

Assignment: Lab 1 - Recursion

Instructor: Dr. Fuentes

TA: Ismael Villanueva-Miranda

Date of Last Modification: 6/18/2019

Purpose of Program: This program will, when given a word, return all possible anagrams of that word

"""

import time

#adding all the words in the text file to a new set named "wordBank"

#we need to split at the end in order to make our words comparable to

#one another, since they would otherwise have extra spaces and \n at the

#end

wordBank = set(open('words\_alpha.txt').read().split())

#create an empty set so we can later fill it with all the

#possible permutations of the given input

permutationSet = set()

#This method recursively adds all possible combinations of a

#given input to our set named "permutationSet"

def permutations(l, stringSoFar):

if len(l) == 0:

permutationSet.add(stringSoFar)

else:

for i in range(len(l)):

permutations(l[:i]+l[i+1:], stringSoFar + l[i])

def output():

userIn = input("Enter a word or empty string to finish: ")

while userIn != '':

print('Word:',userIn)

start = time.perf\_counter()

permutations(userIn, '')

#realWords is our final set that contains all real word combinations

#of our given input

realWords = permutationSet.intersection(wordBank)

print('The word',userIn,'has the following', len(realWords),'anagrams:')

wordList = sorted(realWords)

print(\*wordList, sep = "\n")

timeElapsed = time.perf\_counter() - start

print('It took', timeElapsed, 'seconds to find the anagrams')

permutationSet.clear()

realWords.clear()

userIn = input("Enter a word or empty string to finish: ")

print("Goodybe, Thanks for using this program!")

output()

Statement of Academic Honesty

“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.”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_