CS 2302 Data Structures

Fall 2019

Lab Report #1

Due: September 6th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Programmer: Miriam Olague

Introduction

For this lab we were asked to find the anagrams of a word using recursion. The purpose of this lab is to get familiarized and practice recursion. The main objective of this lab is to learn how to manipulate sets and arrange the words the sets contain to obtain the anagrams of the word the user entered recursively.

Proposed Solution Design and Implementation

Part #1:

For this operation, I used the code from Chapter 2.6.1 on Zybooks as a reference. With this code I iterated through every word in 'words_alpha.txt' (I created it into a set). I first saved the first character of the word into a temporary variable and the remaining of the word was saved into a different variable. After I iterated through the entire word, I saved the word into a temporary set and then compared it to the original set. I then updated the set by comparing the words (all the anagrams) with the original set and see if the anagrams of the words existed inside of the original set. I also called the method (scrambled is a recursive method) and iterated through all the word set I created.

Part #2.1:

For this operation, I approached it in a very similar way as **Part #1**. I iterated through every word in 'words_alpha.txt' (I created it into a set). Inside of an if statement, I compared the first character of the word with the rest of the characters inside that word to prevent it from making recursive calls if the character repeats itself. If the conditions are true for this word, I save the first character inside of a temporary variable. Then, I saved the rest of the word inside of another temporary variable. After iterating through the entire word, I then saved the word into a temporary set and compared it to the original set. I continuously updated the set each time if the new anagram existed inside of the original word set.

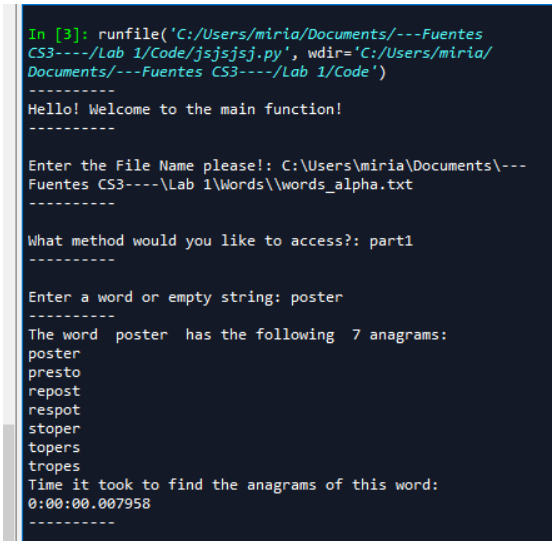
Part #2.2:

For this operation, I approached this problem in a similar way as Part #2.1. I iterated through all the words that are inside 'words_alpha.txt' with a set. I created an if statement, in which I allowed it to make recursive calls if and only if the word the user inputs is inside of the set I created, in which it contains all the prefixes of the original word set (words_alpha.txt). I then saved the first character of the word inside of a temporary variable. Then saved the rest of the characters of the word into a temporary variable. After iterating through all the word, I saved the word into a temporary set and compared it to the original set (words_alpha.txt). I repeated this step for all anagrams that were created so I could compare it and then update it if the anagram existed inside of the original word set.

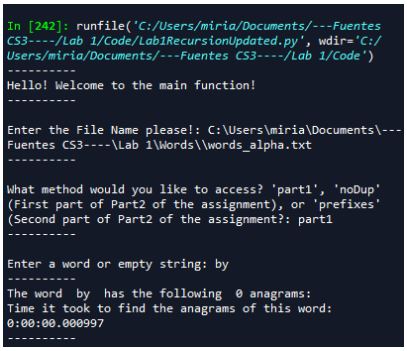
Experimental Results

Part #1:

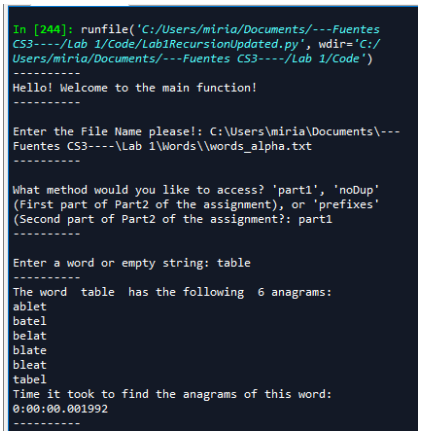
For this operation, I created an empty set, in which all the existing anagrams of the user's input were added into. I first thought that I had to start the time inside of the recursion method but then realized that the time would reset. I attempted to print all the existing anagrams inside of the recursion method, but then decided it would be better to put them in a set and then print them outside of the recursion method. I tried printing the anagrams inside of the recursion call, but I found it cleaner to print them outside.



User's input = poster



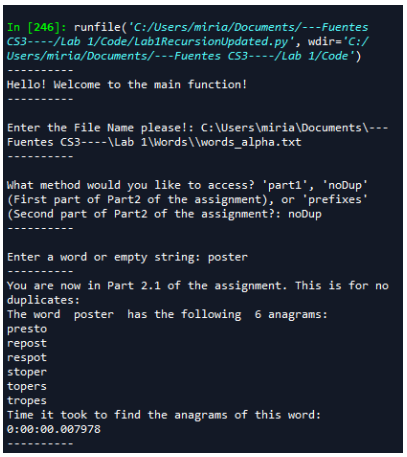
User's input = by



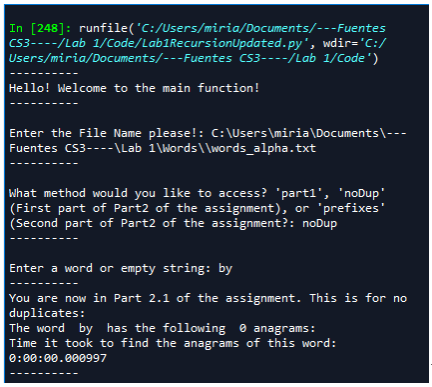
User's input = table

Part #2.1:

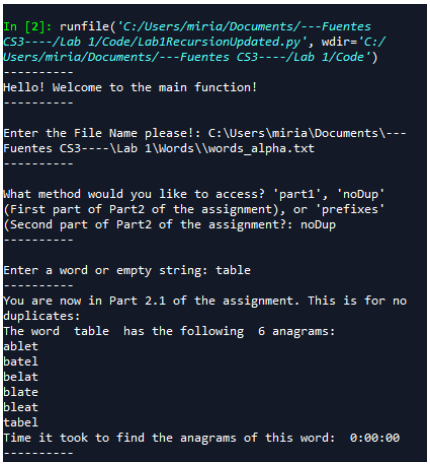
For this operation, I had a hard time understanding how I could compare a character with the rest of the characters. All the experiments I made were the same ones as **Part #1**. The only different experiment I had to do was how to compare the characters with the rest.



User's input = poster



User's input = by

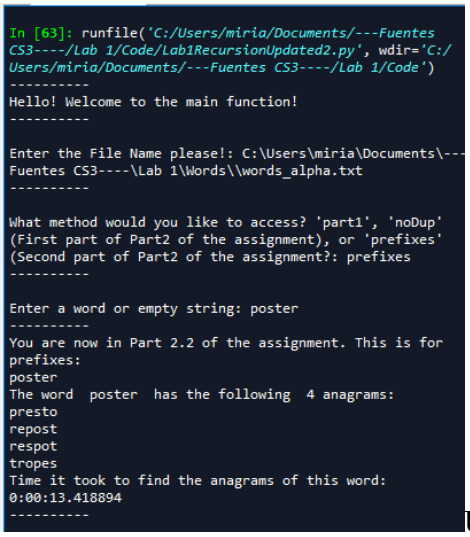


User's input = table

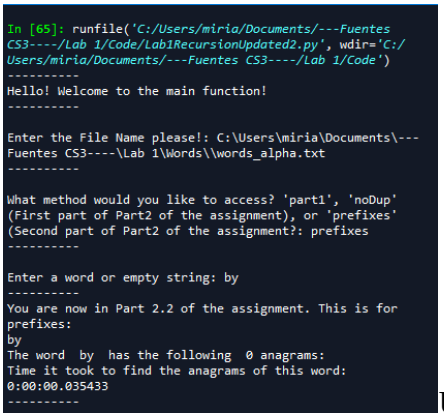
Part #2.2:

For this operation, I approached this problem similarly as **Part #2.1**. The way I created a set with all the prefixes of the original set (words_alpha.txt) was with a list. I made a for loop in

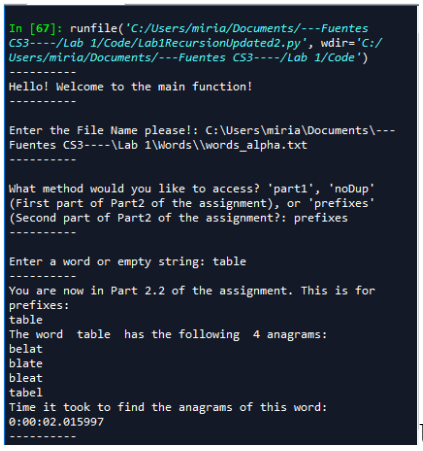
which I added the prefixes of the set and then converted it to a set in order to compare it to the original set (words_alpha.txt), however, all of the anagrams of the word were not found inside of the word set. Since many of the words that were in the list were added incorrectly into the set, it did not find all the existing anagrams inside of the original set (words_alpha.txt).



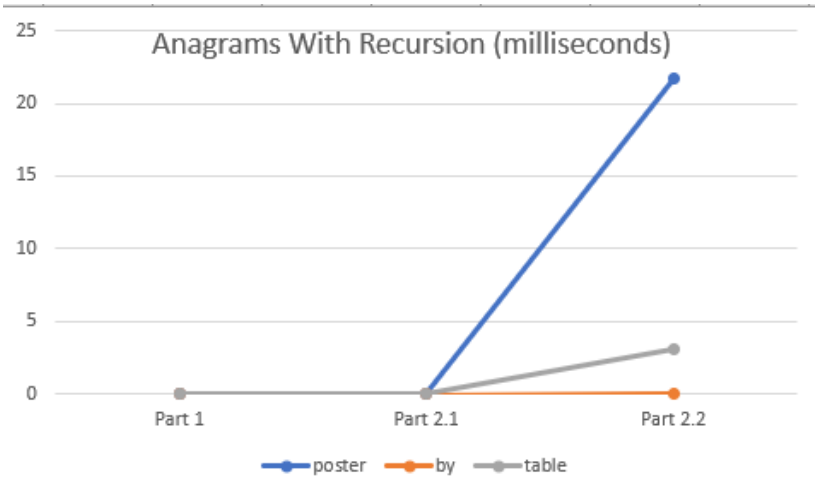
User's input = poster



User's input = by



User's input = table



	poster	by	table
Part 1	0.006953	0.000841	0.000997
Part 2.1	0.004986	0.000264	0.001963
Part 2.2	21.75485	0.048872	3.13166

As the results show, finding the anagrams of a word takes more time iterating through every word in the word set rather than finding the anagrams of a word by only calling the recursive method if no characters repeat. Since I separated **Part #2.1** (no duplicates) and **Part #2.2** (prefixes), if the word is found, it takes more time to iterate through the prefix set and iterating through the original word set (words_alpha.txt).

Conclusion

This lab helped me reinforce my knowledge of recursion. It helped me understand and practice the way lists and sets can be used. This lab was challenging but managed to learn a lot from it the more I spent time on it. I enjoyed it since I am barely learning Python and I find it very interesting. I learned how to be a little bit more organized with my code and learned to give meaningful names to my variables so that way I don't get lost. I learned how to use recursion with python and how to scramble words in every possible way (anagrams).

Appendix

```python
#Course: CS 2302 Data Structures
#Programmer: Miriam Olague
#Lab 1 Recursion
#Date of last modification: September 8th
#Professor: Olac Fuentes
#Purpose: The purpose of this lab is to find the anagrams and display the time it took
#   to find the anagrams.
#TA: Anindita Nath
#---------------------------------------------------------------------------------------
from datetime import datetime
import sys

def scrambled(letters, _scrambled_, _creating_set_, _newSet_):
    #This is part 1 of the assignment
    if len(letters) == 0: #base case)
        _newSet_.add(_scrambled_) #adding the word into the new set
        _newSet_ = _newSet_.intersection(_creating_set_) #re-writing the set with the
        #words that actually appear inside of words_alpha
        if len(_newSet_) == 0:
            print("This word has 0 anagrams. ")
    else:
        for i in range(len(letters)): #letter at i will be scrambled
            _scram_letters_ = letters[i]
            _remain_ = letters[:i] + letters[i+1:] #letter will be removed from remain letters list
            scrambled(_remain_, _scrambled_ + _scram_letters_, _creating_set_, _newSet_) #calling method

#---------------------------------------------------------------------------------------

def noDup(letters, scram, _creating_set_, _newnewSet_):
    #this is part 2 no duplicates of the assignment
    if len(letters) == 0: #base case
        _newnewSet_.add(scram) #adding the word into the new set
        _newnewSet_ = _newnewSet_.intersection(_creating_set_) #re-writing the set with the
        #words that actually appear inside of words_alpha
    else:
        for i in range(len(letters)): #letter at i will be scrambled
            if letters[i+1:] != letters[:i+1]: #making sure that the partial word is a prefix of any word
                #in the word set and that every other character does not repeat itself
                _scr_l_ = letters[i] #saving character
                _rem_ = letters[:i] + letters[i+1:] #letter will be removed from remain letters list
                noDup(_rem_, scram + _scr_l_, _creating_set_, _newnewSet_) #calling method

#---------------------------------------------------------------------------------------
def prefixes(a, scrams, _creating_set_, _newSet3_, prefs_set):
        #this is part 2 of the assignment
    if len(a) == 0: #base case
        _newSet3_.add(scrams) #adding the word into the new set
        _newSet3_.update(_newSet3_.intersection(_creating_set_)) #re-writing the set with
        #the words that actually appear inside of words_alpha
    else:
        for i in range(len(a)): #letter at i will be scrambled
            if a in prefs_set: #making sure that the partial word is a prefix of any word in the word set
                _scrs_l_ = a[i] #saving character
                _rems_ = a[:i] + a[i+1:] #letter will be removed from remain a list
                prefixes(_rems_, scrams + _scrs_l_, _creating_set_, _newSet3_, prefs_set) #calling method
#---------------------------------------------------------------------------------------


print("----------")
```

```
print("Hello! Welcome to the main function!")
print("----------")
fileName = input("Enter the File Name please!: ")
print("----------")
_creating_set_ = set(open(fileName, 'r').read().split()) #This is splitting the words and putting them into a set
size = len(_creating_set_)


_method_name_ = input("What method would you like to access? 'part1', 'noDup' (First part of Part2 of the assignment), or 'prefixes' (Second part of Part2 of the assignment?: ")
print("----------")

_word_input_ = input("Enter a word or empty string: ") #word will be utilized
a = _word_input_
letters = _word_input_
letters2 = sorted(_word_input_)
print("----------")


        #*********************************************

if _method_name_ == "part1": #calling scrambled()
    _newSet_ = set() #This is where I will store the anagrams that were found inside the document
    startTime = datetime.now() #I am starting time as we go inside part1

    scrambled(letters, '', _creating_set_, _newSet_) #calling method
    _newSet_ = sorted(_newSet_.intersection(_creating_set_))
    length_of_newSet = len(_newSet_)

    if len(_newSet_) == 0:
        print("This word has 0 anagrams. ")
    else:
            print("The word ", _word_input_, " has the following ", (length_of_newSet)-1, "anagrams: ")


    _newSet_.remove(_word_input_)

    for i in range(length_of_newSet):
        if i < (length_of_newSet)-1:
            print(_newSet_[i])


    print ("Time it took to find the anagrams of this word: ", datetime.now() - startTime) #I am stopping time as we finish with method part1
    print("----------")

        #*********************************************

#*************************************************************************************************************************************************************
elif _method_name_ == "noDup": #calling noDup()
    print("You are now in Part 2.1 of the assignment. This is for no duplicates: ")

    _newnewSet_ = set()
    startTime = datetime.now() #I am starting time as we go inside part2.1

    noDup(letters, '', _creating_set_, _newnewSet_)  #CALLING METHOD---

    _newnewSet_ = sorted(_newnewSet_.intersection(_creating_set_)) #I am saving the created set by the method noDup
    length_of_newSet2 = len(_newnewSet_)

        #*********************************************
```

```
119
120     if len(_newnewSet_) == 0:
121         print("This word has 0 anagrams. ")
122     else:
123         print("The word ", _word_input_, " has the following ", (length_of_newSet2)-1, "anagrams: ")
124
125
126     _newnewSet_.remove(_word_input_)
127     for i in range(length_of_newSet2):
128         if i < (length_of_newSet2)-1:
129             print(_newnewSet_[i])
130
131
132     print ("Time it took to find the anagrams of this word: ", datetime.now() - startTime) #I am stopping time as we finish with method part2
133     print("----------")
134
135         #*********************************************
136 #=============================================================================================================================================
137 elif _method_name_ == "prefixes":
138     print("You are now in Part 2.2 of the assignment. This is for prefixes: ")
139     _newSet3_ = set() #creating empty set
140     prefs_list = list() #creating empty list
141     prefs_set = set() #creating empty list
142     creatings_list = list(_creating_set_) #converting set to list
143     creatings_list = sorted(creatings_list) #sorted
144
145
146     for i in range(len(creatings_list)): #iterating through every word
147       temps_string = '' #blank
148       temps_word = creatings_list[i] #saving a word temporarily here
149       for j in range(len(temps_word)): #iterating through every character but the last one
150           temps_string += temps_word[j] #adding next character
151           prefs_list.append(temps_string) #adding prefix to the list
152
153     for i in range(len(prefs_list)):
154         temp = prefs_list[i]
155         prefs_set.add(temp) #I am adding every element of the list into the set
156
157     prefs_set = sorted(prefs_set)
158
159
160
161
162     startTime = datetime.now() #I am starting time as we go inside part2.2
163     prefixes(a, '', _creating_set_, _newSet3_, prefs_set) #CALLING METHOD ---
164
165
166     print(a)
167     _newSet3_ = sorted(_newSet3_.intersection(_creating_set_))
168     _lenSet3_ = len(_newSet3_)
169
170     if len(_newSet3_) == 0:
171         print("This word has 0 anagrams. ")
172     else:
173         print("The word ", a, " has the following ", (_lenSet3_)-1, "anagrams: ")
174
175     _newSet3_.remove(a)
176     for i in range(_lenSet3_):
177         if i < (_lenSet3_)-1:
177         if i < (_lenSet3_)-1:
178             print(_newSet3_[i]) #I am printing the anagrams
179
180
181     print ("Time it took to find the anagrams of this word: ", datetime.now() - startTime) #I am stopping time as we finish with method part2
182     print("----------")
183
184 #=============================================================================================================================================
185
186 else:
187     print("Since you did not choose a valid method, you were kicked out of the program. Bye! Thank you for using the program. ")
188     sys.exit() #exits program
189
190 #---------------------------------------------------------------------------------------------
```

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.