Lab Assignment 1

Ismael Villalobs

Professor Fuentes

TA: Dita nath

Introduction

For this assignment our program should then use a recursive function to find the anagrams. The anagrams should be displayed in alphabetical order and contain no duplicates. Your program should also display the time it took to find the anagrams. For part 2, you will implement the following two optimizations:

• If the string has duplicate characters, only make recursive calls the first time the character appears. This will avoid generating the same anagram multiple times

• Stop recursion if the partial word you have is not a prefix of any word in the word set. To do this, you’ll need to create a set containing the prefixes of all the words in the word set.

Proposed Solutions

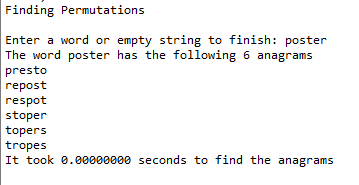
Part 1 - To begin I read in the text file provided ‘words\_alpha.txt’ which was ­ saved into a set. For my recursive call I would create all permutations of the of the word chosen by recursively iterating through the loop that was the length of the remaining letters that needed to be permuted. Once the word was scrambles it would be compared to the set that was created with initial words to see if the scrambled word appears in the word set. If it does appear in the word set it would be placed into a set that contained only permutations that are words in word set.

Part 2 – For this part I used the same methodology as part one but I also had a parameter that included the prefix set for the word. For example, if your word set is {’data’, ’science’}, your prefix set should be {”, ’d’, ’da’, ’dat’, ’s’, ’sc’, ’sci’, ’scie’, ’scien’, ’scienc’} this was to optimize the permutation finder to avoid generating the same anagram multiple times if the string has duplicate characters. In addition it would only allow the recursive call to continue if the partial word is not a prefix of any word in the word set.

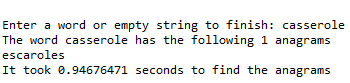
Experimental Results

Part 1

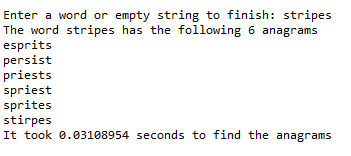
Using Input – “poster”



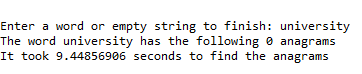
Using Input – “casserole”



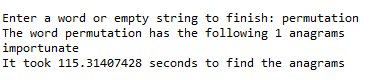
Using Input – “stripes”



Using Input – “university”

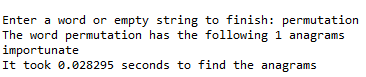


Using Input – “permutation”

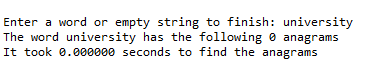


Part 2

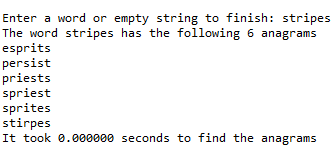
Using Input – “permutation”



Using Input – “university”



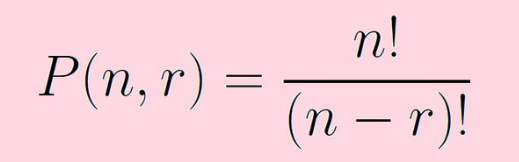
Using Input – “stripes”



As pictured by the previous figures the first program had a much longer runtime for similar words using the second method that used prefix sets to avoid long runtimes with no permutations.

Big O Notation

Since BigO represents an upper bound then you can safely say that there are O(k\*n!)  strings where n is the length of the longest input word (since in the worst case all the strings will have the same length then there will be exactly k\*n! permutations, in a better case with variable lengths the number will be <k\*n! but the notation O(k\*n!) still holds). Thus, has a Big O of O(n!)



Conclusion

In conclusion I was really amazed at how finding permutations of words can be done so easily using recursion. This lab opened my eyes to how it finding permutations can be optimized by validating the prefixes of the words.

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.”

Name: Ismael Villalobos

Appendix:

import time

#Takes a file name, creates a set of words from the given file, and returns the Set

def readFile():

try:

wordList = set(line.strip() for line in open(r'words\_alpha.txt'))

return wordList

#Prints Error message if the file cannot be found

except IOError:

print("Error File Not Found")

#Finds all prefixes of the word inputted

def getPrefixes(string):

prefix = set()

#Creates prfixes by adding x characters from the length of the word

for x in range(1, len(string)):

#Adds the list of prefixes into the empty set

prefix.add(string[:x])

return prefix

def anagramFinder(remainingL, scrambledL, word, wordSet, anagramSet):

#Base Case if there are not letters remaining

if len(remainingL) == 0:

#Adds the word to a set then compares itself with fileSet to see if the

#word is an anagram and if it is it is then added to the anagramSet

tempPre = {scrambledL}

tempSet = set.intersection(tempPre, wordSet)

if len(tempSet) > 0:

anagramSet.add(scrambledL)

#Otherwise it beings making premutations of the inputted word and checks if

#they are within the fileSet

else:

for i in range(len(remainingL)):

scramble\_letter = remainingL[i]

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

anagramFinder(remaining\_letters, (scrambledL + scramble\_letter), word, wordSet, anagramSet)

def anagramFinder\_2(remainingL, scrambledL, word, wordSet, anagramSet, prefixSet):

#Base Case if there are not letters remaining

if len(remainingL) == 0:

#Adds the word to a set then compares itself with fileSet to see if the

#word is an anagram and if it is it is then added to the anagramSet

tempPre = {scrambledL}

tempSet = set.intersection(tempPre, wordSet)

if len(tempSet) > 0:

anagramSet.add(scrambledL)

#Otherwise it beings making premutations of the inputted word and checks if

#they are within the fileSet

else:

#First Optimization

#Creates a new epmty set to hold letters already used

duplicatedL = set()

for i in range(len(remainingL)):

scramble\_letter = remainingL[i]

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

#Uses the set of prefixes to stop the recusrion if the partial word

#is not found in the set

if len(remainingL) > 1:

if(not((scrambledL + scramble\_letter) in prefixSet)):

continue

#Checks the set duplicateL for duplicated Letters

if scramble\_letter in duplicatedL:

continue

#If letter has no been used yet and it is in the prefix set then it

#is added to duplicatedL

duplicatedL.add(scramble\_letter)

anagramFinder\_2(remaining\_letters, (scrambledL + scramble\_letter), word, wordSet, anagramSet, prefixSet)

def permutationsPart1(wordSet):

#Runs the program untill the user enters an empty string

while True:

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

#If an empty sting is entered the program prints a goodbye message and closes

if word == '':

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

break

#If a word is not inputed then it will prompt the user that that there input

#is invalid

elif(not(word in wordSet)):

print("Input invalid")

else:

#Sorts the inputted word alphabetically

sortedList = sorted(list(word))

sortedWord = ''.join(sortedList)

#Adds the word to the set of anagrams to start the set so the word

#is already in the list when we try and remove it

anagramSet = {word}

start = time.time()

anagramFinder(sortedWord, '', word, wordSet, anagramSet)

end = time.time()

runTime = end-start

anagramSet.remove(word)

anagramList = sorted(list(anagramSet))

print("The word", word, "has the following", len(anagramSet), "anagrams")

for anagram in anagramList:

print(anagram)

print("It took", "{:.8f}".format(runTime), "seconds to find the anagrams")

print()

def permutationsNoDuplicates(wordSet):

#Runs the program untill the user enters an empty string

while True:

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

#If an empty sting is entered the program prints a goodbye message and closes

if word == '':

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

break

#If a word is not inputed then it will prompt the user that that there input

#is invalid

elif(not(word in wordSet)):

print("Input invalid")

else:

#Sorts the inputted word alphabetically

sortedList = sorted(list(word))

sortedWord = ''.join(sortedList)

#Adds the word to the set of anagrams to start the set so the word

#is already in the list when we try and remove it

anagramSet = {word}

prefixSet = set()

#Creats and Adds prefixes for all words in the fileSet

#and adds them into the set prefixSet

for setWord in wordSet:

prefixSet.update(getPrefixes(setWord))

start = time.time()

anagramFinder\_2(sortedWord, '', word, wordSet, anagramSet, prefixSet)

end = time.time()

runTime = end-start

#removes the original word from the anagramSet

anagramSet.remove(word)

#Converts anagramSet to a list, sorts them alphabetically, and

#then prints them alphabetically

anagramList = sorted(list(anagramSet))

print("The word", word, "has the following", len(anagramSet), "anagrams")

for anagram in anagramList:

print(anagram)

print("It took","{:.6f}".format(runTime), "seconds to find the anagrams")

print()

wordSet = set(line.strip() for line in open('words\_alpha.txt'))

print("Finding Permutations")

permutationsPart1(wordSet)

print()

print("Finding Permutations without Duplicates")

permutationsNoDuplicates(wordSet)