Lab 01 Report

In our first lab were instructed to write a program that receives a word and returns ever possible anagram for that word. Specifically, our program needed to read a text file containing four hundred thousand words and needed to store those words into a set. Following this, our program needed to contain a recursive function that needed to find every anagram of a word given by a user. The program would display the anagrams in alphabetical order and contain no duplicates. The program also needed to display how long it took the program to find every word.

To start with, I decided to use part of the method that was given to us in class. The function given to us returned every permutation of a given list of characters recursively. The function first checks I if the length of the word I equal to zero (base case), otherwise the function will set a variable ‘b’ to the empty list that was passed through. After this, the a for loop is initialized to iterate through the length of the given list of characters. In the for loop another variable ‘c’ will be initialized and set equal to the given word and the given word is concatenated to the empty list ‘left’ an example of this line is shown here ([] + ['a']). The next line creates a variable ‘d’ and sets it equal to the list ‘left’ and adds one to the entire list. The following line is the recursive call, the recursive call is doing n-1 work and is passing in the variables c, d, and ‘words’.

To make this function work with the set of words that has been read in, I made another parameter in the function that passes in the new set that was read from the file. To convert the set into a string in the method I used another method “convert to string” which creates an empty string and sets it equal to the new variable ‘x’. In the method I set a variable ‘word’, equal to the method that coverts to string. This will change the chosen word to the string. After this I will need to compare the word that is now a string and check if it is in the set of words that has been passed in and will be printed once the recursive.

In the main method, I simply used a while loop to make the program run until the user wanted to stop running the program. I set a condition equal to true and proceeded to run the while loop with the method recursive function method call inside the loop as well as the timer to check the time it took to run the method.

The graph above represents the time it took to run different words with different amounts of letters. According to the data it appears that the run time is above n^2, as I began to input bigger and bigger words, the runtime began to increase at a big rate. I used the program with 10 characters and it seemed to crash.

I also ran the program with multiple of the same letters, such as the letter ‘a’ represented in the graph. I started by only using two, and increasing by one. The time worked similarly to when I ran the program with regular words that increased as I did in the first graph. It seems as if the biggest problem in the code is that the more letters that are being checked, no matter what letters they are they are all being checked. The best solution to get the runtime of the program down is to change the recursive function. I would change the function to check only the letters in a smaller manner. For example, if there are three letters right after one another like ‘xvl’ the function should exit because there are no further cases where the function will return a word. this is why the test for all of the a’s took almost the same time as if I was using a regular word. no matter what, my function is still checking each one of the characters in the string.

Overall, we can see that the recursive function is still running at the (n^2 ) amount of time because the amount of work going on outside of the recursive call. To improve most recursive functions, I would make less work happen by creating better cases in which the program will do less work like when checking all of the letters.

#Daniel Solorzano

#lab01

#06/19/19

#Data Structures summer 2019

import time

#converts from list of chars to string

def convertToString(s):

# initialization of string to ""

new = ""

# traverse in the string

for x in s:

new += x

# return string

return new

def print\_p(chosen, left, n, words):

if n==0:

# convert the word, check if it is a word against set

# print if it is in the set

word = convertToString(chosen)

if word in words:

print(word)

# print(chosen)

else:

b = len(left)

for i in range(b):

c = chosen+[left[i]] # [] + ['a']

d = left[:i]+left[i+1:] # ['a'] + ['b','c']

print\_p(c, d, n-1, words)

def main():

condition = True

newSet = set(line.strip()for line in open('words\_alpha.txt')) #open the text file back and read the content

while condition:

word = input("enter a word or an ampty string to finish: ")

if len(word) == 0:

condition = False

print('Word: ' + word)

print("The word " + word + " has the following anagrams:")

# start timer

start = time.time()

print\_p([], word, len(word), newSet)

# end timer

end = time.time()

total = end - start

print('it took' , total , 'seconds to find the anagrams')

print("Bye, thanks for using this program")

if \_\_name\_\_ == '\_\_main\_\_':

main()

“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.”

X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_