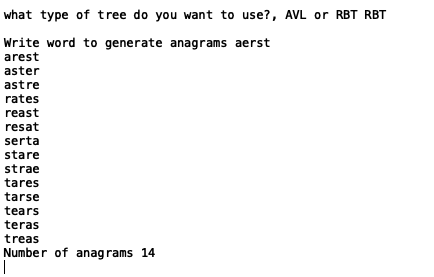
The objective in this lab was to utilize a binary tree, either AVL or Red Black Tree, to find the anagrams of a given English word. To populate the tree, I utilized a file called word.txt that contains over 300,000 English words. I attempted to solve the problem by implementing the zyBooks code for both AVL and Red Black Trees to create my tree. During the execution of my program, I initially asked the user which kind of tree they would like to use. Since the .txt file contains over 300 thousand words I decided not to provide the user which a visual representation of the tree. In my code however, I commented two lines of code that could be implemented if the user would like to print the entire AVL tree or would like to know the total height of the Red Black Tree created. I also asked the user for what word they would like to know the anagrams. After the user answers, I implemented the three methods required for our assignment, print\_anagrams, count\_anagrams, and greatest\_anagrams.

For the first method, print\_anagrams, I utilized the binary search tree created by the user as a global variable, called engish\_words. The method simply printed the anagrams found in the whole tree. I did not modify this method, I utilized the one provided by the instructor. It is important to note that during the instructions I was provided with the execution of the method with the word “spot”. However, my output was not similar to the one expected since some of the exact anagrams where not found within the tree. For my second method, count\_anagrams,I used the counterL, the tree name, and the word as my parameters. I used a variable counterL to keep track of the number of anagrams present in the tree. This counterL was returned and was printed for the user to identify the number of anagrams found within a tree,which in this case was engish\_words. Within this method I utilized the method .search() found within both the RedBlackTree and NodeAVL classes. I was able to use this method with the original words.txt file provided. This method is completely similar to the one provided by Professor Aguirre, I just added 4 lines of code to do the desired operation. The running time of this algorithm was based on the height of the tree, making it O(log(n)).

For my method greatest\_anagrams, which counts the greatest number of anagrams in a file, I only the name of the file and the tree\_type provided by the user in the initial question as parameters.I utilized a counterL to keep track of the number of anagrams returned by the method count\_anagrams(). The count\_anagrams method was the same one previously mentioned.This counterL was then compared to max\_anagrams and the greater of the two was saved along with the word for which it belonged. This method traversed the whole tree looking for the greatest number of anagrams of a given word. At the end it returned the last word with the greatest number of anagrams along with the total number of anagrams not including itself. For this method, my time complexity was dependent on the height of the tree, but also on the number of words in the given list. The method checked the tree for every word contained in the file. The time complexity is O(nlog(n)).

 For my test cases, I first implemented the program using the complete 300 thousand word list. I searched for the anagrams of the word aerst. My greatest anagrams method never finished executing in my computer, since after 10 min the computer aborted the execution. I believe this was due to the file size since for a smaller file the method worked. This was the output for this test case.

A close up of a tree

Description automatically generated For my second test case, I used a smaller file containing 28 anagrams for the word aerst. This .txt file contained over 100 words. This time, the method greatest\_anagrams() was correctly executed. This was the output:

For the output above, I decided to use an AVL tree. I used a RBT tree as well, but noticed no significant difference in the actual execution time, only in the time spent creating the actual tree.

For this lab I learned how to implement AVL trees and RBT trees to search for a word. In this case we were searching for a word, but I assume it can be any time of input. I initially read the instructions for the lab wrong a created a list to find the word with the greatest anagrams without implementing a tree first. My program never finished executing and my computer aborted the execution. I was able to compare how more efficient the creation of the tree was for this case by mistake. There was a small variation in creation time for the kinds of trees. The creation of the RBT tree was faster. For my appendix I only included my main file where I executed the methods described above. The classes I utilized to create the trees I uploaded on GitHub.

Appendix

﻿"""

Created on Fri Oct 26 08:53:43 2018

@author: Diana Ramirez 88604827

Professor: Diego Aguire

Class: CS 2302 TR 1:30PM

Lab 3B

"""

from NodeAVL import Node,AVLTree

from RedBlackTree import RedBlackTree

#########################################################################################

def print\_anagrams(word, prefix=""):

if len(word) <= 1:

str = prefix + word

if engish\_words.search(str) == str:##str in engish\_words:

print(prefix + word)

else:

for i in range(len(word)):

cur = word[i: i + 1]

before = word[0: i] # letters before cur

after = word[i + 1:] # letters after cur

if cur not in before: # Check if permutations of cur have not been generated.

print\_anagrams(before + after, prefix + cur)

###########################################################################################

def count\_anagrams(counterL,treename,word,prefix=""):

if len(word) <= 1:

str = prefix + word

if treename.search(str) == str : ##str in engish\_words:

counterL +=1

else:

for i in range(len(word)):

cur = word[i: i + 1]

before = word[0: i] # letters before cur

after = word[i + 1:] # letters after cur

if cur not in before: # Check if permutations of cur have not been generated.

counterL = count\_anagrams(counterL,treename,before + after, prefix + cur)

return counterL

##########################################################################################

def greatest\_anagrams(file,tree\_type):

counterL = -1

max\_anagrams = -1

max\_word = ''

##reading the file and creating a list

fileT = open(file)

list\_w = fileT.readlines()

for i in range(len(list\_w)):

list\_w[i] = list\_w[i][:-1]

if tree\_type == 'AVL':

new\_tree = AVLTree()

for ln in list\_w:

words = Node(ln)

new\_tree.insert(words)

if tree\_type == 'RBT':

new\_tree = RedBlackTree()

for ln in list\_w:

new\_tree.insert(ln)

for word in list\_w:

counterL = count\_anagrams(0,new\_tree,word)

if(counterL > max\_anagrams):

max\_anagrams = counterL

max\_word = word

print('The word with the most anagrams is',max\_word,'it has',max\_anagrams,'anagrams')

###########################################################################################

##first I saved all the lines in the file in a list

g= open("words.txt")

word\_list = g.readlines()

tree\_type = input('what type of tree do you want to use?, AVL or RBT ')

##based on the input of the user I created a an AVL or RBT called engish\_words

if tree\_type == 'AVL':

engish\_words = AVLTree()

for ln in word\_list:

ln = ln.replace('\n','')

words = Node(ln)

engish\_words.insert(words)

##print(engish\_words)

if tree\_type == 'RBT':

engish\_words = RedBlackTree()

for ln in word\_list:

ln = ln.replace('\n','')

engish\_words.insert(ln)

##print( 'tree has height ' + str(engish\_words.get\_height()))

word = input("Write word to generate anagrams ")

print\_anagrams(word)

count = count\_anagrams(0,engish\_words,word)

print("Number of anagrams",count)

##checks for the last word with the greatest number of anagrams

greatest\_anagrams('words.txt',tree\_type)

“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.”

