Introduction

Program that sorts English words into an AVL or Red-Black Tree then has

different functions with anagrams applied to those words.

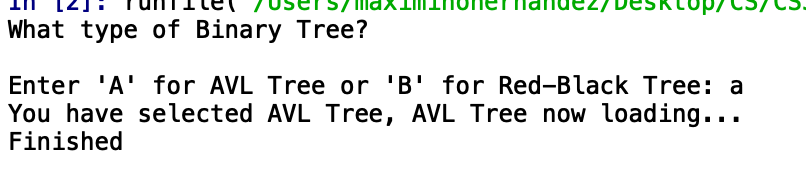
Proposed solution

At first, I had to use the Zybooks implementation of AVL trees and Red-Black Trees. Then I made functions to insert the valid English words into an AVL tree or Red-Black tree. Then, I used one of these implementations to populate Engish\_words. With these methods working, I was able to perform some methods with anagrams that included printing, counting, and getting the largest anagram from a file.

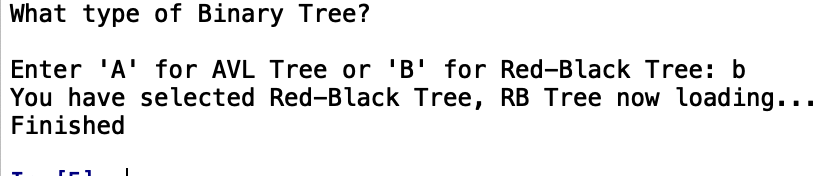
Experimental results

The test cases were not very long and are simply using an AVL tree, Red-Black Tree and anagrams. I tested on a file with a couple of words. My program is able to put all 400,000 English words into a tree in about 3 minutes however, takes a very long computing the anagrams

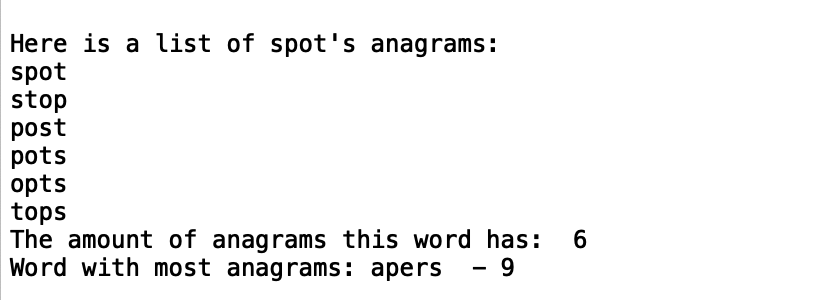
*SAMPLE #1 AVL Tree on testw.txt:*



*SAMPLE #2 Red Black Tree on testw.txt:*



*SAMPLE #3 Test Anagram Functions on testw.txt:*



Conclusion

I learned how to create an AVL and Red-Black tree while also learning how to find, count, and get the largest anagrams from different words.

The time complexity of this program would be O(n^2)

Linked list search takes O(n^2) with bubble sort time while merge sort is O(nlogn) time. The top 20 list takes O(n) time.

Appendix

|  |
| --- |
| #!/usr/bin/env python3 |
|  | # -\*- coding: utf-8 -\*- |
|  | """ |
|  | Jesus Maximino Hernandez |
|  | CS 2302 Data Structures - Diego Aguirre |
|  | TA - Manoj Saha |
|  | Lab 3 - Option B |
|  | Program that sorts english words into a Binary Search Tree and has |
|  | differnt functions with Anagrams |
|  | """ |
|  | from AVLNode import AVLNode |
|  | from AVLTree import AVLTree |
|  | from RBTree import RBTree |
|  | from RBTNode import RBTNode |
|  |  |
|  | fileName = "words.txt" |
|  | testFile = "testW.txt" |
|  |  |
|  | def main(): |
|  | #make sure input is valid |
|  | valid = False |
|  | while(valid == False): |
|  | print("What type of Binary Tree? ") |
|  | answer = input("Enter 'A' for AVL Tree or 'B' for Red-Black Tree: ") |
|  | answer = answer.lower() |
|  |  |
|  | if (answer != 'a' and answer != 'b'): |
|  | valid = False |
|  | print("Error") |
|  | else: |
|  | valid = True |
|  |  |
|  | if(answer == 'a'): |
|  | print("You have selected AVL Tree, AVL Tree now loading...") |
|  | AVL(fileName) |
|  | print("Finished") |
|  | valid = True |
|  |  |
|  | else: |
|  | print("You have selected Red-Black Tree, RB Tree now loading...") |
|  | RBT(fileName) |
|  | print("Finished") |
|  | valid = True |
|  |  |
|  | #Test functionality of other functions. |
|  | # test() |
|  |  |
|  | def test(): |
|  | print() |
|  | testWord = "spot" |
|  | print("Here is a list of " + testWord + "'s anagrams:") |
|  | print\_anagrams(testWord) |
|  | print("The amount of anagrams this word has: " , count\_anagrams("spot")) #should print out six |
|  | print("Word with most anagrams: " + get\_largest(testFile),"-",count\_anagrams(get\_largest(testFile))) |
|  |  |
|  | def engish\_word(word): |
|  |  |
|  | avlTree = AVLTree() |
|  | #opens file and puts into tree |
|  | with open(testFile) as f: #change file to testFile when you want to test |
|  |  |
|  | for line in f: |
|  | if "\n" in line: |
|  | line = line.replace("\n", "") |
|  | lowerCase = (line.lower()) #this line makes every word a lower case |
|  | node = AVLNode(lowerCase) |
|  | avlTree.insert(node) |
|  |  |
|  | if avlTree.search(word): |
|  | #print("FOUND") |
|  | return True |
|  | else: |
|  | #print("NOT FOUND") |
|  | return False |
|  | #function to return the word with the most amount of anagrams |
|  | def get\_largest(file\_name): |
|  | counter = 0 |
|  | with open(file\_name) as f: |
|  | for line in f: |
|  | line = line.replace("\n", "") |
|  | cur = count\_anagrams(line) |
|  | if cur > counter: |
|  | counter = cur |
|  | large = line |
|  | return large |
|  | #counts the number of anagrams a word has |
|  | def count\_anagrams(word, prefix=""): |
|  |  |
|  | if len(word) <= 1: |
|  | str = prefix + word |
|  | #adds one to count when an anagram is a word |
|  | if engish\_word(str): |
|  | return 1 |
|  | return 0 |
|  | else: |
|  | count = 0 |
|  | for i in range(len(word)): |
|  | cur = word[i: i + 1] |
|  | before = word[0: i] |
|  | after = word[i + 1:] |
|  | if cur not in before: |
|  | count += count\_anagrams(before + after, prefix + cur) |
|  | return count |
|  |  |
|  | #function to print anagrams of a word |
|  | def print\_anagrams(word, prefix=""): |
|  |  |
|  | if len(word) <= 1: |
|  | str = prefix + word |
|  | if engish\_word(str): |
|  | 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) |
|  | #function to put words into an AVL tree |
|  | def AVL(fileName): |
|  | avlTree = AVLTree() |
|  | #opens file and puts into tree |
|  | with open(fileName) as f: |
|  |  |
|  | for line in f: |
|  | if "\n" in line: |
|  | line = line.replace("\n", "") |
|  | lowerCase = (line.lower()) #this line makes every word a lower case |
|  | node = AVLNode(lowerCase) |
|  | avlTree.insert(node) |
|  |  |
|  | return avlTree |
|  | #function to put words into a Red and Black tree |
|  | def RBT(fileName): |
|  | rbtTree = RBTree() |
|  | with open(fileName) as f: |
|  | for line in f: |
|  | if "\n" in line: |
|  | line = line.replace("\n", "") |
|  | node = (line.lower()) #this line makes every word a lower case |
|  | rbtTree.insert(node) |
|  |  |
|  | return rbtTree |
|  |  |
|  | 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.”