Lab Balanced Trees

## Introduction

The learning objective of this lab is to understand binary balanced trees. We will be using two types of trees, an AVL tree and a Red Black tree. There is no need to understand the implementation of these trees, just how to slightly adapt them to this project. You’re given a list of words with their respective vector, you will need to modify your tree nodes to carry a key and a list of numbers. You will be asked to get different attributes from your tree like the number of nodes, the height of the tree, write to multiple files, and get the desired depth.

## Design

### AVL and Red Black

My AVL implementation came from a Youtube video that explained the implementation a bit better than zybooks. My red black implementation did come from zybooks, however, I used the same rotation functions in both AVL and Red Black. Both implementations as far as I know work properly.

### Part 1

You’re given a file with words and their corresponding vectors, I created a user interface that will give you the option to read this file with either an AVL tree or a Red Black Tree. The actual function takes two parameters the tree which data will be added, and the name of the file the user would like to open. The type of tree is optional, but the file name must be hardcoded. Once the file is open each line will be read and split by the word and the vectors. Each number is converted to a float type and then added to a list. This list is then set as the nodes data while the word is set as the nodes key.

### Part 2

Here we’re asked to read a file with pairs of keys, check the similarities of these keys by using the given equation on their respective vectors. The function takes two parameters, the tree and a file name. The function opens a file and then takes a pair of keys and calls another function to perform a search of the tree. Once the actual vectors have been received the program performs the equation and prints it to the console.

### Part 3

1. To get all the nodes I just added a class attribute to count as the nodes are being inserted. This was to make the function as efficient as possible while also being readable.
2. Both Red Black and AVL nodes contain a function to get height. This is done using recursion.
3. I transverse through the entire list and add it to a new list. Then write to a new list.
4. To get the depth I used recursion, the function contains three parameters. The desired height, node, and a list of nodes. To get to the desired depth I just recursively call the function while subtracting the desired height. Once the node is null or the desired depth is 0 then you add that node to the list of nodes. This list of nodes then writes to a generated file.

## Results

Test Case 1: For this test case I use the file given. I cap the amount of inserts at 5000 and 1000 for the sake of time, but AVL and Red Black both successfully finish inserting and execute all tasks. AVL does take quite a long time compared to Red Black. To test if the similarities was properly run I used the given examples in the lab document.

Test Case 2: Was used as I was writing the code, check the strings were being split properly. The correct depth was being accessed. The height of the tree was correct. That the tree was being replaced by the words not just numbers.

Test Case 3: Was bad inputs, these files could not be read, not correct file names. The program would crash, but these crashes were built in. Every built in crash has a “Created by @Edd1e234” tag.

N = 5000

AVL Time

* Part 1: 8.820852168999409
* Part 2: 0.0010966919999191305
* Total Time is: 9.21806204999939

Red Black Time:

* Part 1: 0.16481943699909607
* Part 2: 0.0003111180012638215
* Total Time is: 0.1760175759991398

N=10000

AVL Time

* Part 1: 38.20808696746826
* Part 2: 0.0022592544555664062
* Total Time is: 38.23329997062683

Red Black Tree

* Part 1: 0.3522162437438965
* Part 2: 0.0003070831298828125
* Total Time is: 0.37168121337890625

N = 20000

AVL Time

* Part 1: 137.66244220733643
* Part 2: 0.00202178955078125
* Total Time is: 137.7247679233551

Red Black Tree

* Part 1: 0.7109780311584473
* Part 2: 0.0003082752227783203
* Total Time is: 0.7544629573822021

### Time Complexity

When it comes to inserting the program takes advantage of the balance structure. It mostly runs at about **log N**. Searching also takes advantage of this, this runs at about **log N**.

Getting all nodes is **N**, getting the height is **N**, and getting the desired depth is **M**. Relative to the amount asked for.

## Conclusion

What the data shows is that the red black is faster at more operations. However, online I did read red black is better for inserting and removing but AVL is better for searching. Either way, my red black flies through the list. Even with the entire file being read. Perhaps the reason my avl is much slower is because I did not use the zybooks implementation. The reason I opted out of zybooks was because I could not understand what was going on in terms of code, however, the youtube video I saw broke it down much better.