CS315 Programming Assignment 2 Writeup- Trees:

Running Code Instructions:

To run the program, change your terminal directory to data, which contains the files needed in the assignment.

My personal example: PS C:\Users\rylee\CS315\data>

Then run the code under that directory. The file names are already inserted within the code, so there is no need for inputting anything.

It most likely will not run correctly since it is full of imperfections.

Binary Search Tree:

For this tree, I first created a node class called Node(), initializing the children and parent to None, and setting an initial key to reference in future functions. Next, I created a class called BST(), which contains the functions needed to implement the binary search tree. In this assignment, I needed to be able to insert nodes, delete nodes, and traverse the tree in order. This requirement resulted in writing functions BSTinsert(), BSTdelete(), transplant(), tree_search(), tree_min(), tree_max(), and inorder_trav().

tree_min() searches for the smallest value in the tree, which happens to reside in its leftmost node. When called, tree_min() starts at the root of the tree and searches every node to the left of it until it reaches the end of those subtrees. tree_max() does the same, except it goes to the right instead. I thought to try to find the height through this, but after a while, I just could not figure it out. Either way, I wanted to use those to find the height of the tree while using a global height function located towards the top of my code.

To insert the nodes, I used BSTinsert, which decides where to put the new value into the tree. It checks the values of the nodes surrounding the current node in the program, comparing them to the new wanted value, eventually finding a place at an empty, in-order position. It then resets the parents and children accordingly.

Red-Black Tree:

To implement my red-black tree, I created a new node class, RBNode, just to keep each tree implementation separate. A node in a red-black tree also requires a color attribute, so I added one in this node class, initializing it to red. This portion also needed insert and delete functions, but with more thorough investigating of the tree. Since red-black trees require every red node to only have two black children, my code attempts to check the color of the current node's relatives –the parent, children, uncle etc. , which can then be used within the rotation function needed to readjust the tree's nodes.

I then created the *RBTree* class, which contains the functions needed to implement the algorithm. These are also slightly incorrect somewhere along the line, unless the issue is in my implementation at the end, which is totally possible. Regardless, based on the assignment guidelines, I

created functions RBinsert, RBinsert_fix, left_rotate, right_rotate, RBtransplant, RBMin, RBdelete, and RBdelete_fix. Looking back now, I believe I should have had an RBmax function too, for calculating height later. The min/max functions probably should have been a global variable for both the Binary Search Tree and the Red-Black tree classes to share.

RBinsert attempts to insert a node just like the binary search tree, but because of the RB-tree's constraints, RBinsert_fix must find and correct any violations caused by the insertion. RBdelete does the same, but it removes nodes and must also have a fixer—RBinsert_fix. I believe I may have had issues within my rotation functions or something that calls them, since the error kept leading me through that function. I might have gotten pointers, as well as left and right mixed up, for a lot of the same text is located in a small area of the code.

This implementation also attempts to traverse the tree in order, to help logic. I spent so much time on height that I ran out of time to debug that part.

Calculating Height:

When compiling, I know I was able to get the logic for inserting into the search tree to work, but while running, I kept arriving at an error regarding the height of either tree. First, I tried to use a counter within the traversal functions to find the height of each respective tree. After a few hours of trying, I moved on to give height its own function, placing its own version within each tree class. This is when I first encountered the errors on my implementation. The issue could be how I implemented the functions in my main code, but I first tried to find the issue within the functions themselves. Since both ended up being extremely similar, I felt that it was redundant to have it down twice, so I just made it a global function and moved it outside of the classes. I still ran into an error after this, so I checked on my main implementation, which, to be frank, did look pretty wonky—I was using too many pointers, so I removed them from in front of where I call my height function. I believe this allowed the program to run even further, but then I kept getting repeated lines after attempting to delete nodes within my main.

(I honestly did work really hard on this for a long while. It takes me so long to figure out small issues that I spend most of my time trying to understand logic that my brain just cannot seem to visualize)