Tommy McDermott
C212 Stage 01
10/5/2020

1. A tree is a hierarchical data structure composed of nodes and including their root node and children nodes. Each node can have more than one child.
2. A binary tree is a tree where each node has 2 maximum possibilities. A good example used by the book was a *decision tree* that asked yes or no answers and each node either went to yes or to no.
3. Binary search trees are made up of the characteristic that all nodes to the left are less than the current node, and all to the right are greater than it.
4. A red-black tree is defined by the rules that every node is either red or black, the root is black, red nodes can't have red children, and all paths from the root to a null have the same number of black nodes. The "equal exit cost" put this problem in perspective very well for me, saying if the black nodes were toll booths costing the same price, before arriving at any null, you should be paying the "same price".
5. Searching for a value on a RBT is the same as that on a binary tree. The user searches left and right, and moves in the direction of their target value (whether up or down).
6. To insert data a user can use .add(), as in the context *var*.add("Lorium Ipsem"), and the method for *add* essentially creates a new Node *newNode* and ascribes to its left and right nodes a null value. It also makes sure that the root has a value before adding *newNode* to the root, otherwise if the root is null the method will make *newNode* the root.

```
public void add(Comparable obj)
{
        Node newNode = new Node();
        newNode.data = obj;
        newNode.left = null;
        newNode.right = null;
        if (root == null) { root = newNode; }
        else { root.addNode(newNode); }
}
```

After adding this, the color comes into play. The user assumes black and makes any new node red, and if left with a double red (implying three total nodes) the user will make the middle node value the red, and it will expand into that greater than and less than it, both black.

7. Begin removal by searching the left and right nodes and progressing the appropriate direction until you reach the node whose value you want to remove. With one child, removal is as simple as modifying the child to the parent node of that being removed. However with two children, the user must search down the tree to find the closest value to that being removed, copy and store its value in the node being "removed", and reroute the node on the bottom that could possibly be isolated in the process.