Lab 07 Fruit Tree

Bradley Grose

# Problem

In this lab, we must make a program that can import a text file, use a scanner to take the data out of it, and create binary search tree with the type and weight of different types of fruit. The program must be able to add, remove, and order the binary search tree.

# Solution

I started this lab by making a basic binary search tree class to Type<T> that was comparable. In this basic class, I created the node and the linked with it. Then I established the root and created the start of the tree. The add function takes in data, if there is no root it makes that root, if not it uses recursion to check which branches to go down to add the data to that spot. The search function works similarly, it takes in a value and recursively searches the branches for that value. Then, the print pre order function recursively prints out each branch in the proper order. The print in order function goes through each branch and prints our in the organized tree in the correct increasing order. The print post order goes through recursively printing out the branches. Then the remove function recursively looks for the needed value by checking branches. Once the value is found, it will find the minimum in the tree that can be used as a replacement value and fixes the tree so the leaves below the removed value are still searchable. Then, in the fruit class, I create an object fruit that has a weight and name. It has constructors, mutators, and accessors. It also has a toString function to print out the result. It also has a compare to function that compares it to other objects if given, this is used to sort the objects. Finally, in the main tester class, we start with user input of finding the file name. Then using IO we split the file and create the objects needed. Once the object is made, tree.add is called adding the fruit to the binary search tree. Then, we have it print out the different traversal orders, as well as showing that deletion works.

# Implementation Problems Encountered

No problems were encountered in this lab.

# Lab Report Questions

1. A self-balancing binary search tree is a tree that follows Big O complexity that has balanced sides and children on each side, creating equal paths amongst the tree. A non-self-balancing might be more weighted on wide side with more values than the other, making some traversals longer than other traversals.
2. For searching a balanced search tree, the complexity is being able to get those values sorted in an even manner to create the less traversal to cover to find the needed value. In a non-balanced tree, the complexity is it will take possibly much longer to find the data because there are more branches the data could in theory be stored under.