Binary Search Tree Algorithm

Homework #8

By

John Kiyak

CS303

October 23, 2018

**Problem Specification**

The goal of this assignment was to write a program that would implement a red black tree algorithm recursively.

**Program Design**

For this program, 4 different classes were made. BinaryTree.java, BinaryTreeNode.java, RBNode.java, and RBTree.java. The BinaryTree.java had the insert method, which inserted values into the tree, inorderTreeWalk, which printed out the tree values, search, which searched for a specific key value within the tree itself, and the main class, which had the scanner class for getting the input from the UPC.csv and input.dat file. The BinaryTreeNode.java class was used to instantiate the nodes, calculate the height of the tree, and order the output of the data. The RBNode.java class extended the BinaryTreeNode class, and had the left and right rotate methods. The left and right rotate methods took in the trees, and compared the left and right sides of the tree to each other, and rotated depending on which one greater. For the RBTree.java class, it exnteded the BinaryTree.java class. It had the RBInsert method, which inserted values into the RBTree, and it also has an insertfixup method to fix any problems that arose in the insert method,

**Testing Plan**

For my testing plan, I would make sure that the arrays for both the ‘input.dat’ and ‘UPC.csv’ files were printed and stored correctly. I would then make sure that the key is printed and different for every test. Afterwards, I would make my node, and insert different values using the insert command, make sure that the inorder function works, and verify that the search function works as intended. Afterwards, I would test to make sure my left and right rotate methods would work as expected, and make sure the output for my main method would deliver in a good length of expected time complexity.

**Analysis and Conclusions**

A red black tree differs from a regular binary search tree in that its average and worst case are both log(n) time complexity. It also differes from a binary search tree in that its insertion, search and delection methods are both O(log(n)) for a red black tree, but for a binary search tree, it is O(n). In our case, this would case a big time discrepancy, considering that the input file had over 150,000 entries.