CIS 313, Intermediate Data Structures
Fall 2018

CIS 313 Lab 4

Due: November 30th, 2018 at 4:59pm

This lab involves implementing a Red-Black Tree

Overview

You will Construct a Red-Black Tree of numbers. You will extend your code in programming assignment 2 to now include a balancing operations. If you prefer, we have posted a working BST and Node class for you to build off of. Note: BST.java is there as a reference. No need to turn it in along with your code.

A BST is a Red-Black tree if it satisfies the following Red-Black Properties:

1. Every node is either red or black
2. Every leaf node counts as black
3. If a node is red, then both its children are black
4. Every simple path from a node to a descendant leaf contains the same number of black nodes
5. The root node is always black

Fill out all of the methods in the provided skeleton code.
You may add additional methods, but should NOT add public fields. You also should not change the name of any of the classes or files.
The program should continually accept instructions until exit is entered.

In particular, you will implement the following functionality for your Red-Black Tree:

insert
Preform BST insert
Check to see if the tree breaks any of the Red-Black Properties
If so, preform the appropriate rotations and or re-colorings

delete
Preform BST delete
Check to see if the tree breaks any of the Red-Black Properties
If so, preform the appropriate rotations and or re-colorings
Note: You will not be asked to remove the most difficult nodes
(ie) you won't need to recursively rebalance
See Extra Credit

search
Return the node corresponding to the given number
Print "Found" if the corresponding key is in the tree.
Otherwise, print "Not Found"

leftRotate
If x is the root of the tree to rotate with left child subtree T1 and right child y, where T2 and T3 are the left and right children of y:

- x becomes left child of y and T3 as its right child of y
- T1 becomes left child of x and T2 becomes right child of x

rightRotate
If y is the root of the tree to rotate with right child subtree T3 and left child x, where T1 and T2 are the left and right children of x:

- y becomes right child of x and T1 as its left child of x
- T2 becomes left child of y and T3 becomes right child of y

Additionally, implement the following instructions in the main() function in lab4.java

traverse
Preform the preorder traversal of the Red-Black Tree

exit
Exit the program
Print "Successful Exit"

Input Description

The input will be a text file, for example *inSample.txt* below will be provided. Each of the lines (unknown how many) will contain a different set of words specifying a task along with a number (if applicable). You should create an empty Red-Black Tree (with root null), and then perform a sequence of actions on that tree.

Note: You should implement your Red-Black tree with generics, but create a Red-Black tree that takes in integers.

```
insert 10
insert 5
insert 20
insert 3
traverse
insert 2
traverse
search 17
insert 1
delete 20
traverse
exit
```

Output Description

Using the sample input above, your program should output:

```
10 5 3 20
10 3 2 5 20
Not Found
3 2 1 10 5
Successful Exit
```

Testing Protocol

We will test your program in the same fashion as previous labs. We strongly suggest you test your program in the following ways:

- While creating it
- With inSample.txt
- With the provided test.sh file found in `/home/users/smergend/public/cs313/lab4` on ix-dev.

Grading

This assignment will be graded as follows:

Correctness 40%
Your program compiles without errors (including the submitted files NOT containing package names at the top. Delete the package name from the top of the files before you submit them): 10%
Your program runs without errors: 10%
Your program produces the correct output: 20%

Implementation 40%
Insert, delete, search, traverse, and exit all preform in the correct time complexity for the RBT: 10%
leftRotate, and rightRotate are also preform in the correct time complexity: 30%

Documentation 20%
To earn points for implementation, your code must be clear enough for us to understand

Extra Credit 20%

Each part of the extra credit will be worth 10%
We will test the extra credit similarly to how we test the rest of the assignment.
Add an additional public function isRBT() to your RBT.java that checks if the current tree matches RBT rules. Add an extra command in lab4.java that is called "test"

1. Instantiate a new object from the given class WrongTree.java at the beginning of lab4.java
2. Instantiate a new RBT from your RBT.java at the beginning of lab4.java
3. Set root on your RBT to the root returned in WrongTree.getRoot()
4. Determine if your current RBT is in fact a RBT
    - Don't change WrongTree.java, write code in RBT.java
5. Print the private time field in WrongTree.java using the getTime() method
6. Print true or false depending on if you think the tree is a RBT

For the other half, we will simply run a more in depth test input file.