CIS 313, Intermediate Data Structures Fall 2020

CIS 313 Lab 4

Due: Wednesday, November 25 at 11: 59 pm

This lab involves implementing a Red-Black Tree building on your knowledge from Binary Search Tree.

Overview

You will Construct a Red-Black Tree of numbers. For your convenience, some of the BST operations have already beed written for you in the skeleton code. You are also given a working traverse, insert and delete methods for BST. You simply need to extend the functionality to support balanced insert and delete operations.

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

print_with_colours()
It's the same as the print_tree() in the skeleton code but prints the color code along with the node, 'B' for black and 'R' for red. It also prints in preorder traversal format just like print_tree()

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.

You will implement the following functionality for your Red-Black Tree:

Note: The BST functionality for insert, delete has already been provided.

left_rotate()

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

right_rotate()

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

insert()

Preform BST insert

Check to see if the tree breaks any of the Red-Black Properties. If so, perform the appropriate rotations and or re-colorings. That is, implement the __rb_insert_fixup() function.

delete()

Preform BST delete

Check to see if the tree breaks any of the Red-Black Properties. If so, perform the appropriate rotations and or re-colorings. That is, implement the **__rb_delete_fixup()** function.

Additional Test Cases

1. Test left rotate of intermediate node
    a. Insert the following elements using bst_insert(): [7,5,9,3,6,8,10,1,2]
    b. Left rotate node 9
    c. Test if the rotation was performed correctly
2. Test colours post insertion
    a. Insert the following elements using insert(): [7,5,9,3,6,8,10,1,2]
    b. Test the color of every node after the above insertions
3. Add additional 3 test cases

Submission

Compress the rb_tree.py and test_lab4.py files and upload in Gradescope similar to the previous programming assignments. The test cases file may contain additional test cases that you have written to test your code.

Grading

70 points - AutoGrader
10 points - Style
15 points - Additional Test Cases
5 points - DocStrings

To earn points for style, your code must be clear enough for us to understand. The rubrics for different

functions will be provided in Gradescope.