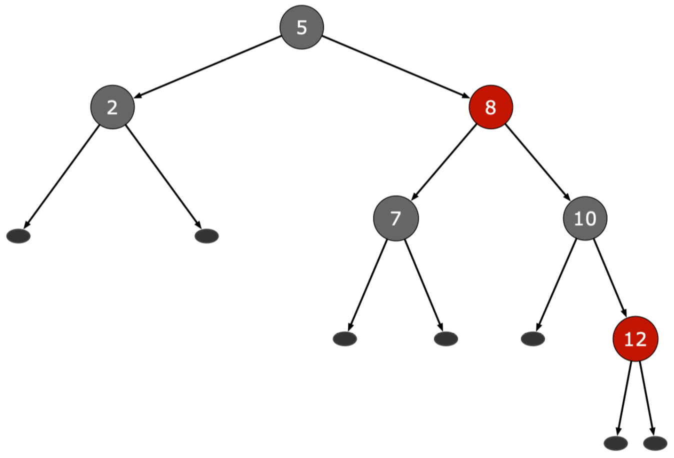
Practice Problem Set 7

.

Problem 1

- Create a BST by inserting the following keys in this order:
  $10, 15, 22, 30, 20, 12, 7, 4, 13, 11, 2, 5.$
  Show the result of deleting node 7. Next show how to delete node 15.
- Show how to insert the new keys 9,4,3, 15, 20 into the RB tree shown below.



Problem 2.

Let T be a pointer to the root node of a BST. Write the pseudo-code for a recursive algorithm called VerifyBST(T) that returns *true* if the tree rooted at T is a valid binary search tree, and *false* otherwise.

Problem 3

- Let T be a pointer to the root of a BST. Write the pseudo-code that returns a reference to the **maximum** element in the BST. Repeat for the **minimum**.
- Given a node x in a BST rooted at node T, write the pseudo-code which finds the **successor** to x in the BST. You may assume that x is not the maximum. Note that you cannot simply assume that x has a right child. What is the runtime of your algorithm?

Problem 4

Let x be a reference to a node in a BST. Let y be the root of another BST. Write the pseudo-code for Replace(x,y) which replaces node x with the subtree rooted at y. You may assume for simplicity that x is not the root of the BST, and that replacing x with y does not violate the BST property.

Problem 5

Let T be a pointer to the root of a BST. Let z be a pointer to a node that is contained in tree T. Write the pseudo-code that deletes z from the tree, and returns a reference to the new updated tree. You may use the functions you wrote in problem 2 and problem 3. Consider all possible cases for node z: it may be the root, it may be a leaf, it may be an internal node.

Problem 6 (**not for problem participation*)

- Give an example of a binary search tree with height $h = 3$ using exactly 15 nodes. Using the same keys, give examples of a BST whose height is 4, 5, and 6.
- Given a RB tree with black-height b, what is the minimum height of the tree? What is the maximum-height of the tree? Give an example of each type when $b = 3$.

Problem 7

Show that the runtime of INORDER is $O(n)$. Use the recurrence from class. Provide the pseudo-code for PRE-ORDER and POST-ORDER and determine their run-time in big-oh notation.

Given any one of the above order types, is it possible to uniquely determine the shape of the BST?

Problem 8

- When we carry out Case 2 of RB-repair does the black-height of the tree change? Explain why the black-height property is not violated after the rotation.
- Under what conditions does the black-height of the whole tree increase after a new node is inserted into a RB tree. Explain your answer.

Problem 9:

- Suppose x is the root of a RB tree. Explain why the black-height of $x.left$ and $x.right$ differ by at most one.
- Suppose a red-black tree has black height 3. What is the maximum difference between the number of nodes in the left subtree and the number of nodes in the right subtree? What is the maximum difference between the *height* of the left and right subtrees?

Problem 10:

Write a recursive algorithm that takes as input a node x from a Red-black tree and determines if all paths from x to a leaf have the same number of black nodes. In this case, the algorithm should return the *number* of black nodes. If it is not the case, the algorithm should return -1. You may assume that the color attribute is stored in $x.color$.

Problem 11: (**not for problem participation*)

The TREE-INSERT algorithm we saw in class was written with a *while* loop. Re-write this algorithm *recursively.*

Problem 12:

Suppose a binary search tree is such that: the difference between the heights of the left and right subtrees of any node x is at most one. These trees are called AVL trees and their heights are $\Theta(\log n)$. Write a recursive algorithm that takes as input the node of a binary tree and returns the height of the tree if the tree is an AVL tree, and some other integer (of your choice) if it is not.

Problem 13:

In class, we defined binary search trees assuming that there were no duplicate keys. Suppose now that we would like to build a BST in such a way where duplicate keys may exist. To handle duplicates easily, we augment each node x with an attribute $x.count$ which indicates the number of occurrences of the key $x.key$ in the tree.

Your job:

- re-write the Tree-Insert(T, z) algorithm from class so that duplicate keys are handled correctly. Note that z is a reference to a node. Is the runtime the same?
- re-write the BST-delete(T, z) algorithm so that duplicate keys are handled correctly. Is the runtime the same?