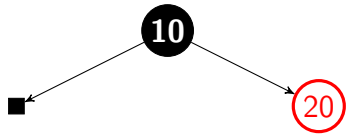
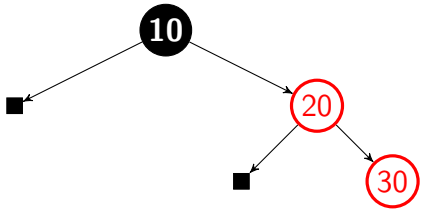
Question 1: Does inserting a node into a red-black tree, re-balancing, and then deleting it result in the original tree?

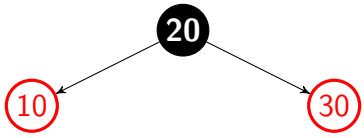
We will show that inserting, rebalancing, and then deleting a node *can* result in a new tree. Consider the following tree:



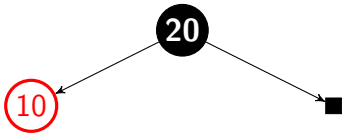
Adding 30 to this tree will result in the following:



This tree is not balanced because it contains a red node with a red child. From the RBInsert Algorithm line 11, we perform a left rotate on 20 (parent of 30). This will make 20 the root, and by line 17, the root will be colored black. This results in this tree:

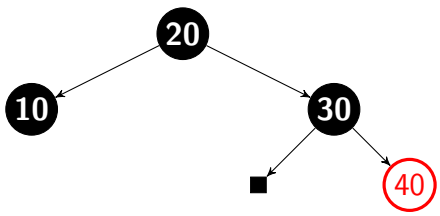


Now deleting 30 from the tree we are left with a tree that is different from the original:

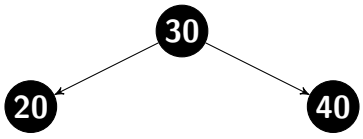


Question 2: Does deleting a node with no children from a red-black tree, re-balancing, and then re-inserting it with the same key always result in the original tree?

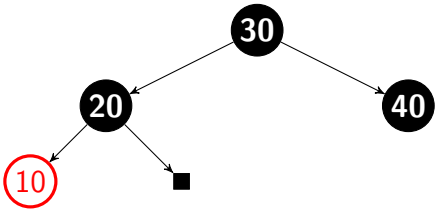
We will show that by deleting a node with no children, re-balancing, and then re-inserting the same node can result in a different tree. Consider the following tree:



Deleting 10 will result in an unbalanced tree. The RBBalance Algorithm will be called since x will be a null node black child of 20. This will result in a left rotate on 20 (line 21 of RBBalance) and the re-coloring of 40 (line 20 of RBBalance). The resulting tree is:



Now, re-inserting the 10 node we get the following:



This tree needs no rebalancing after insertion and is a valid Red Black Tree. Comparing it to our original tree, we can see that the two differ in several areas. Thus, proving that deleting a node with no children, re-balancing, and re-inserting the same node *can* result in a different tree.