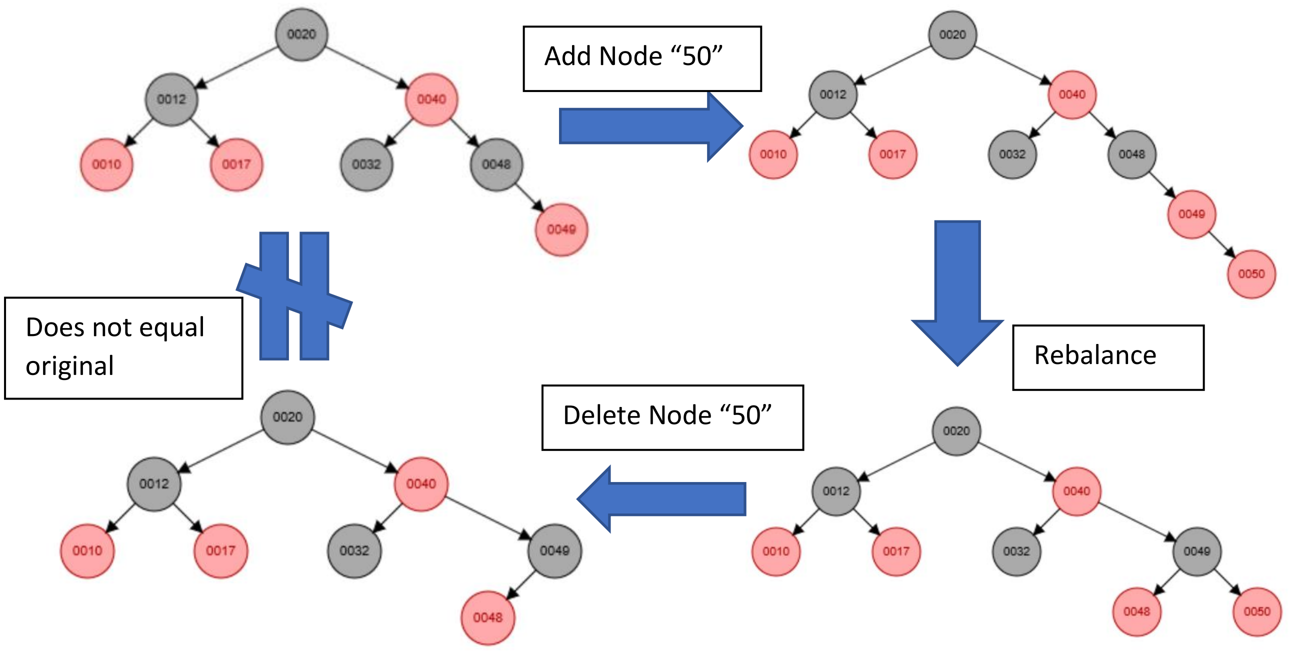
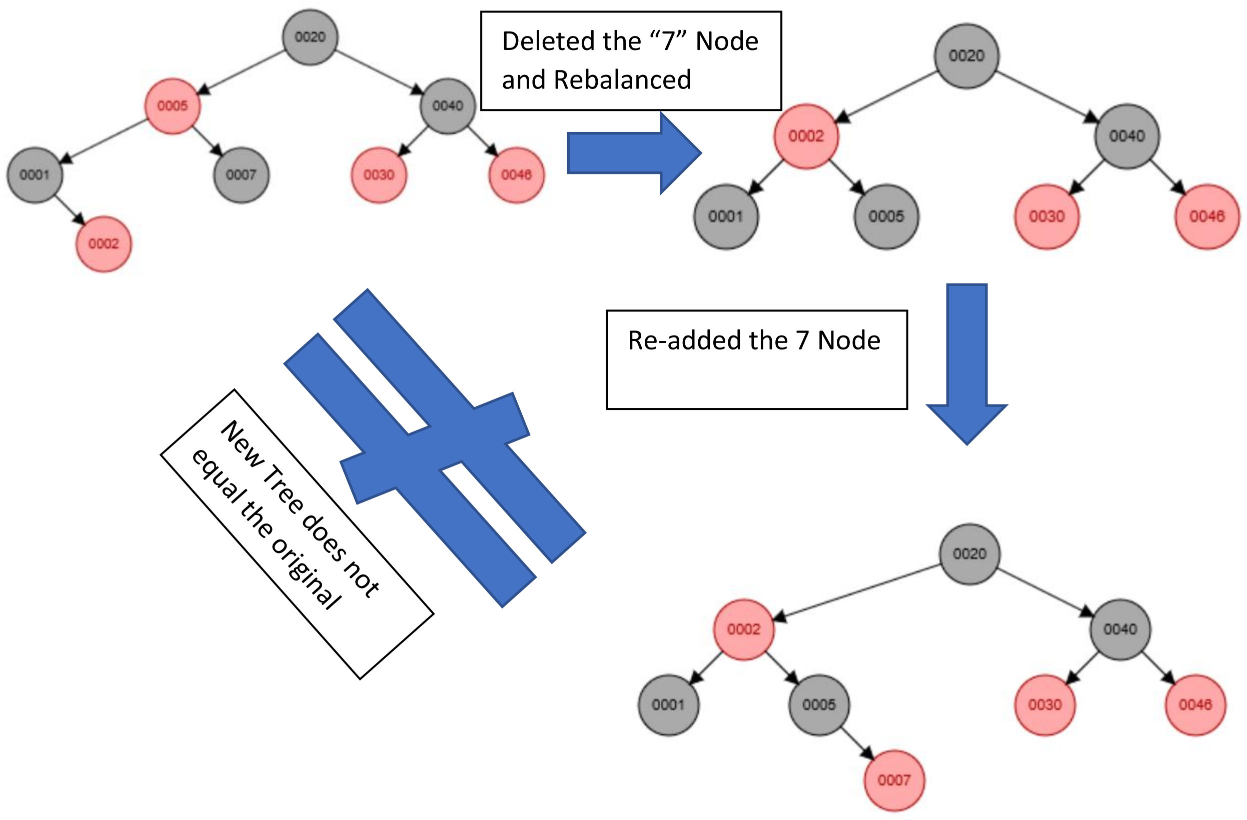
Question 1: Does inserting a node into a red-black tree, re-balancing, and then deleting it result in the original tree? **NO**



We begin by looking at the addition of the “50” node. Before rebalancing, it seems that the new nodes parent is the node”49”, and the grandparent node is “48”. In this case, the new node does not have a uncle node. The issue now lies because the tree is not balanced nor is it following the color rules which prohibits the tree from having two red nodes consecutively (“49” and “50”). Now, the tree must rebalance. We must fix the fact that the tree has two red nodes in a row. The end goal is to make it so the great-grandparent is a black node and to rotate the tree so that it meets the color rules for the tree. We see that since there is no uncle, we just have to preform an easy rotation. In this case, because the right side is heavy we just need to preform a left rotation about the argument node “49”. This rotation then turns the grandparent node into a sibling node and makes the previous great-grandparent into the grandparent. We can check that we are done if the color rules are met (switch the color if necessary) and then see if the great-grandparent (in this case the root) of the new node is black. It is so we are done. Now deleting the node won’t change the tree because it is just a leaf node and so it wont make the tree unbalanced nor break any color rules so you can simply get rid of the node. This, however, does not result in the original tree.

Question 2: Does deleting a node with no children from a red-black tree, re-balancing, and then reinserting it with the same key always result in the original tree? **<u>NO</u>**


0020
0005
0040
0001
0007
0030
0046
0002
Deleted the "7" Node and Rebalanced
Re-added the 7 Node
New Tree does not equal the original

This case is only true if the node that is being deleted is red. This then implies that it is not the uncle of another node and thus won't need to rebalance the tree. In this case, the node "7" is a black node that is being deleted. It is the uncle node of the "2" node. When the "7" node is deleted, it creates a left heavy tree that needs to be rebalanced. This is done in two separate steps. First, a simple right rotation with 1 as the argument can occur. This would make it so that the node "1" is the child of the root node and then the nodes "2" and "5" would be the child and grandchild nodes. This then sets up an issue in color and balance. There would then be two red nodes in a row which is against the color rules for red-black trees, as well as a right heavy tree on the left side. To take care of this a left rotation with argument "2" can occur and some recoloring. The "2" node would become the child of the root and the "1" node would become its left child, and "5" would be its right child. We can now apply the color test and see that the tree is balanced and colored appropriately. So now we must just add back in the "7" node. However, now when the node is added back in, it is added still after the "5" node but now its uncle is the "1" node where previously it was its sibling. Also, the "node" is now red because it is a leaf node succeeding a black parent node. This is different from before because previously it was a black node and not the lowest node in the branch. So, in this case, deleting a node with no children, rebalancing, and adding it back in does not result in the original tree. If we, however, now call the new tree the original, deleted the "7" node, and then added it back in, the trees would look the same.