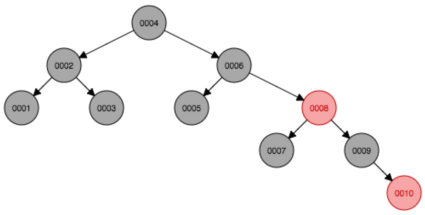
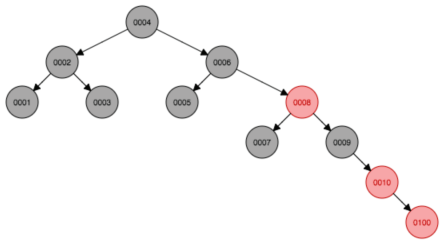
Thomas Payne
Assignment 7

Question 1:

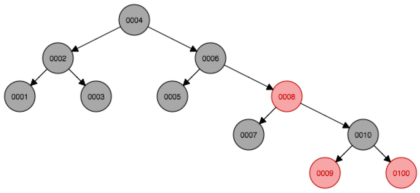
Original Tree Ex1: Add 1 - 10



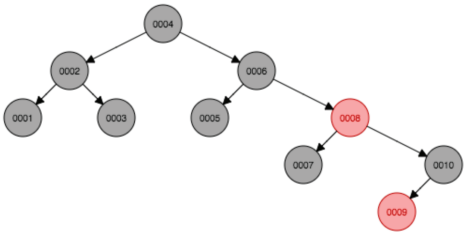
Adding 100: unbalanced



Adding 100: balanced



After deleting 100:



Explanation: Insert

1. set node(100) to be red, evaluate if greater or equal to root.
2. because 100 > root, traverse right and continue to evaluate the next right node
3. node.parent.color == red
4. if : node = node.parent.right
5. then : node = node.parent
6. The fact that 10 and 100 are both red, right children breaks a rule. We must accomidate and preform a leftrotate around 10, now 100's sibling is 9, and parent is 10. This rebalances the tree.
7. make node(100).parent.color = black and node(100).parent.parent.color = red

Explanation : Delete

1. Find the node(100)
2. assign node.parent to null and delete the node

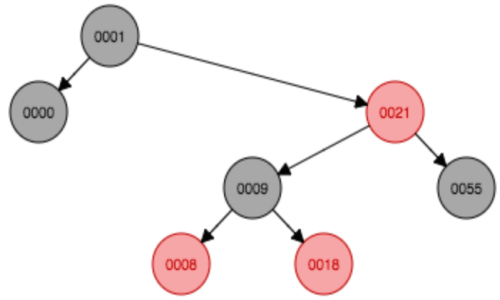
3. 8 remains red, and 9 remains as 10's left child. The final result does not look like the original because we never rearranged the 9.parent to point to 8, and 9.right to point to 10. This was not be necessary because after deleting 10, the rules remain intact.

To Recap:

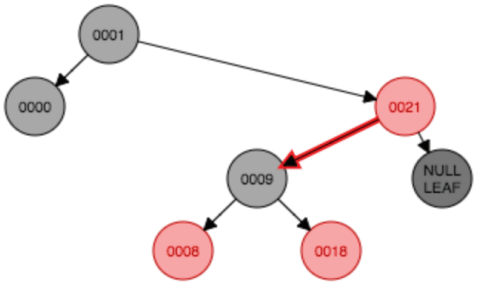
Adding 100 to the tree relocated the right child of 8 to be 10, and made 10.left = 9, and 10.right = 100. After deleting 100, the final result was not the same as the original. The final tree was rebalanced and resulted in 9 being a red child of 10, where as in the original, 10 was a red child of 9.

In Conclusion:
When you insert a node into a RB tree, rebalance it, and delete that same node, the resulting tree is not the same as the origional. This is due to the rebalancing function making necessary changes to pointers and node colors to account for breaking the RB tree 'rules'. These pointers and colors are not necessarily reinstated.

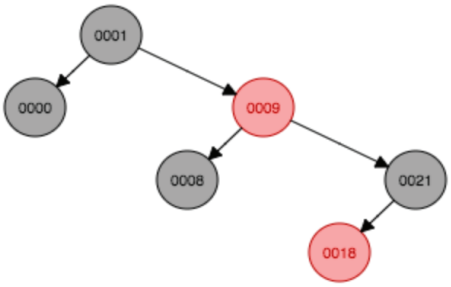
Question 2:



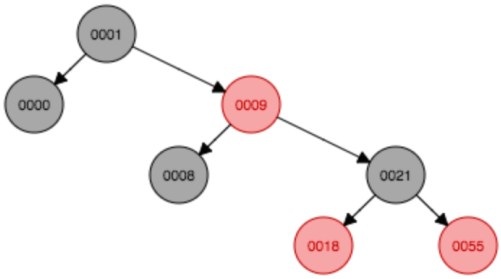
In this case deleting a node with no children and reinserting it does not result in the origional tree



Explanation: Delete 55
1. node does not equal root, enter case with no children
2. parent node is 21, sibling is 9, nodeColor = black
3. delete the node
4. This creates a violation, because the path from the null Leaf(shown on far left of tree) has 1 black node, where the path from 18 or 8 have 2 black nodes.
5. Rebalance the tree around 9
this essentially..
- assign the root.right to be 9
- assign 9.right to be 21
- assign 21.left to be 18
6. Tree is now reblanced with 9 being right red child of root, 9 leftchild remains 8, 9 right child changes from 18 to 21.



Explanation: Insert 55
1. Traverse far right and insert.
2. No rules are broken. We are finished.



To Recap:

After deleting, rebalancing, and reinserting the node, it is clear that we do not have the orginal tree. After deleting, the rule that states every path taken must have the same amount of black nodes is broken, thus we must rebalance. This causes 9 to become red, and have two children.
Reinserting 55 gives 21(9's right child) a right child. This breaks no rules, thus we do not have to further rebalance, nor rearange any pointers.