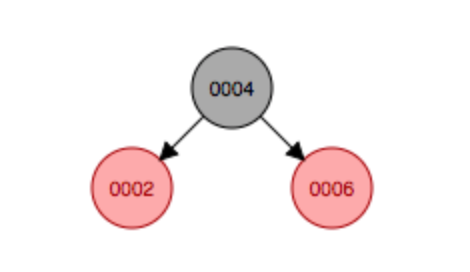
John Furlong

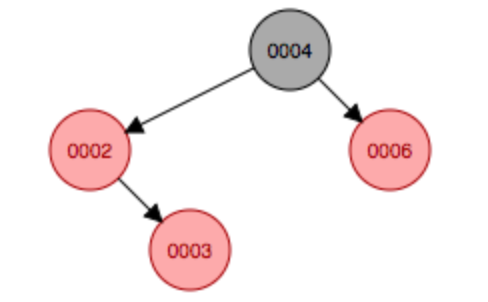
Assignment 5

CSCI 2270  
July 15, 2018

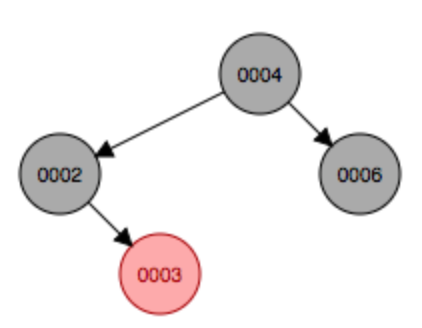
**Question 1:** Does inserting a node into a red-black tree, re-balancing, and then deleting it result in the original tree?

Inserting a node into a red-black tree, re-balancing, and then deleting it does **not** necessarily result in the original tree every time. For example, if we insert the values {4, 6, 2} respectively, our red/black tree will look like the following:

If we go to insert a node with the value {3} into our tree, the new node and its parent will both be red, as well as the uncle node to the new node we are inserting. Before the tree gets balanced, the tree looks like this:



The parent node and the uncle node of the new node get colored black after they violate our conditions, and the root node changes back to black from red. After the tree is rebalanced it will look like this:

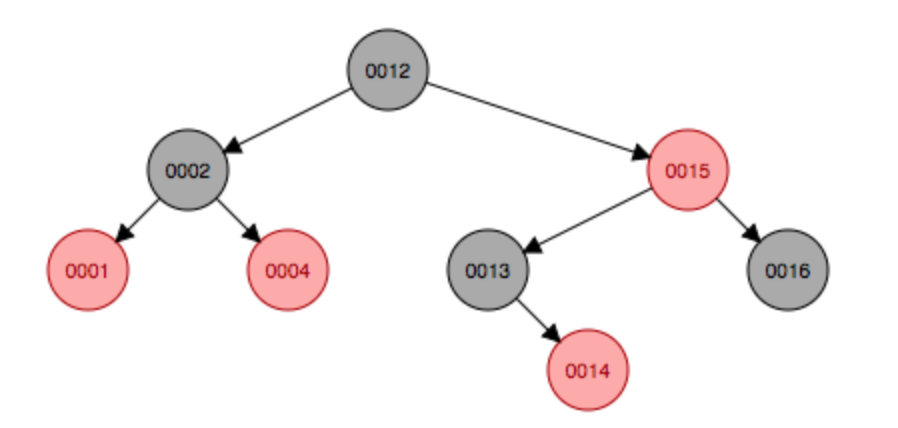


If we now go to delete the node containing 3, we know that this is a leaf node so it gets deleted and our final, balanced red/black tree looks like this:

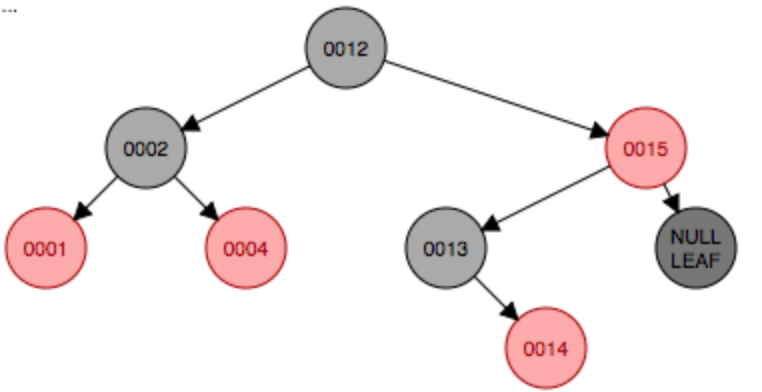
This is not the same as the tree we initially had, where the nodes containing {2} and {4} were red.

**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?

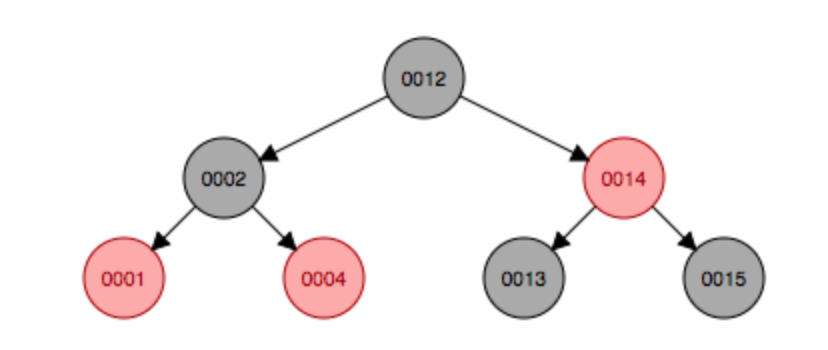
Suppose we inserted the numbers {12, 4, 15, 1, 2 ,13, 16, 14} into our red/black tree, respectively. Our red/black tree would look like this:

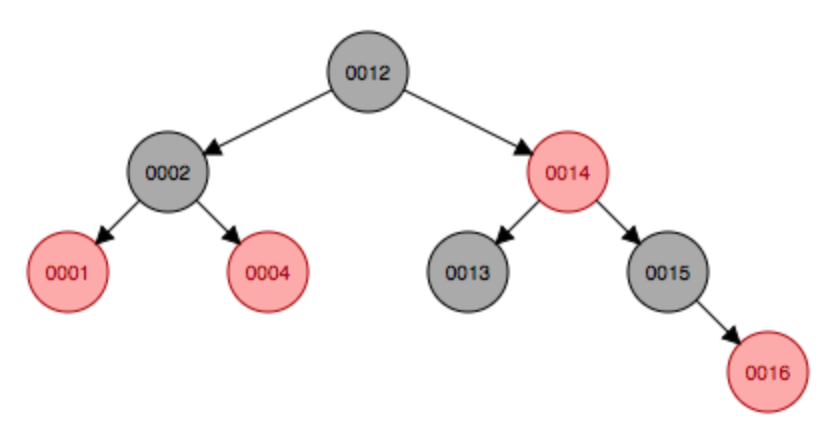


The node containing the value 16 has no children so we will delete this node from our tree to determine the answer to this question. Once the node is deleted, the double black node has a sibling but the double black node is a right child and the left nephew node is black.



Therefore, we single rotate left to make the opposite nephew node red. After we’ve rotated, the double black node has a black sibling so we must single rotate right to fix the double blackness. Once the tree is rebalanced we have:



Now, we insert a node with the same value as the one we’ve just deleted (16) to determine whether the resulting tree will be the same as our original. No violations occur while inserting 16, which ends up as the right child to the node with key 15. The resulting tree is:

We can see that the resulting tree after reinserting the node with key 16 after we deleted it was different than our original tree. While adding the node with key 16 did not trigger any violations, deleting the “16-node” is what forced our tree to rotate twice and change colors which changed it from our original. So, it must be the case that the resulting tree will not *always* be the same as the original after deleting a node with no children, rebalancing and then reinserting a node with the same key.