Ramon Martinez

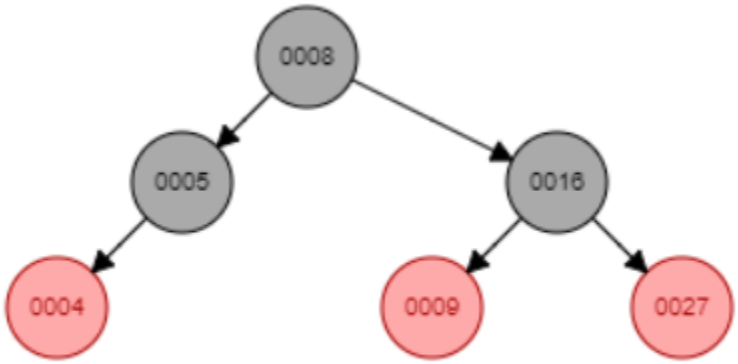
SID:105941151

CSCI 2270

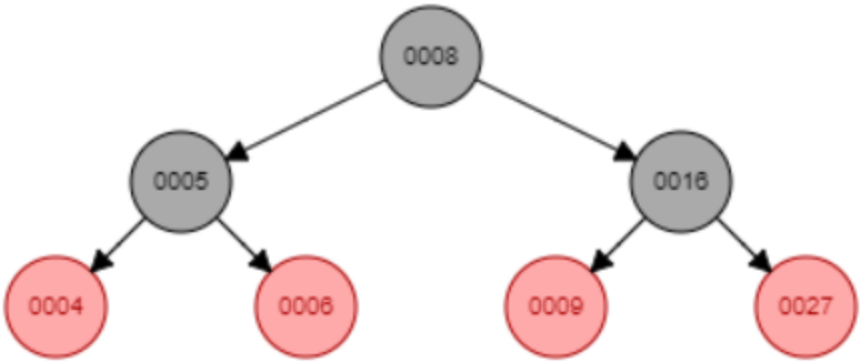
3/18/2018

Question 1:

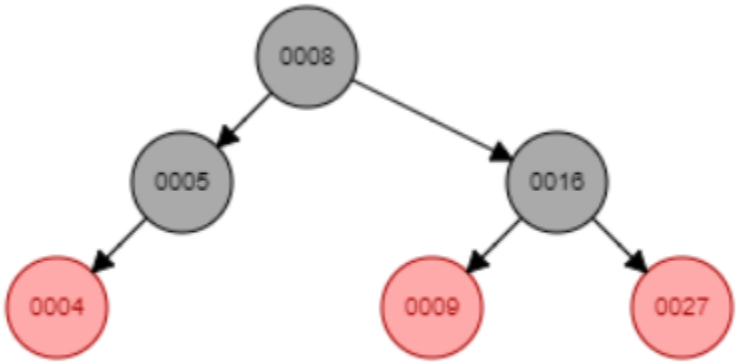
Inserting a node into a red black tree will not always result in the same tree as before. For example, let's say we start with a red black tree with the following values 9,5,8,27,16, and 4. If the values are inserted into the tree in the order given the resulting tree will look like the following:



The only insertion and deletion that would result in the same tree would be if we add a node to the right of 5. For this example, lets insert the number six. For the insertion step, we know that the parent of 6 is going to be 5, the grandparent will be 8 and the uncle will be 16. Now using the insertion algorithm, we will insert the node into the tree and set its color to red. Now we must check for any violations. The first violation that could occur is if x is the root of the tree since the root cannot be red. But in this case the node we wish to add is not going to be the root so this violation will not occur. The next violation would be if 6's parent (5) was the color red. This would violate the property that if a node is red, then both its children are black. However, 5 is black so the violation is once again not a problem, thus no corrections are needed and we can just insert the node. The tree will then look like the following:

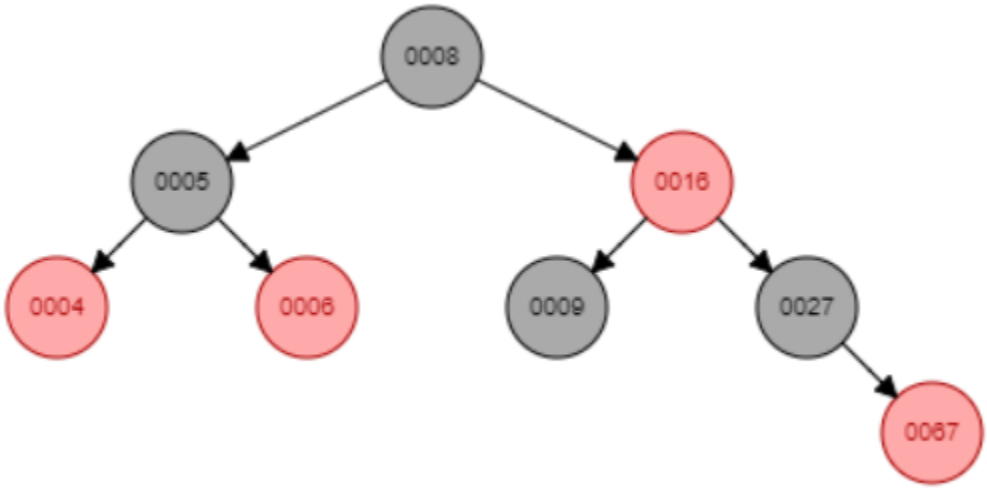


Now that the node has been inserted and the tree has been balanced we can delete 6. Before we can delete the node, we fist must find it in the tree so we will search the tree to find the node. Once the node is found we must find out if the node has 0,1, or 2 children. In this case the node in question (6) has no children so we can make the nodes parents right child null and then we must check if the tree needs to be rebalance. Since the node color is not black no rebalance is needed and we can just delete the node. Thus, the tree will once again look like:

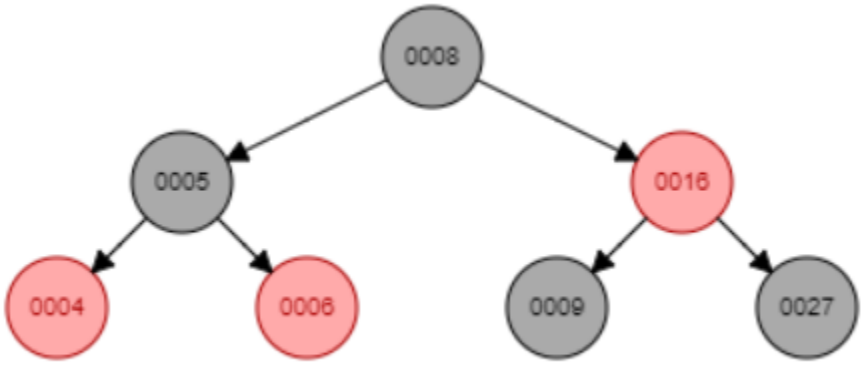


Now let's consider the case of when the tree after inserting and deleting a node results in a new tree. Let's try to add 67 to the tree. Once we insert the node 67 into the tree, the parent of 67 becomes 27, the grandparent is 16 and its uncle is 9. Now looking at the insert algorithm, we know that 67 is not the root of the tree and the parent of 67 is not red so the while loop evaluates to true so we enter the loop. The first if statement evaluates to false since the parent of 67 is parent is not to the left of his grandparent so we evaluate to false and look at the case of where x is to the right of his grandparent. Thus, the uncle is going to be equal to x.parent.parent.left so 67 uncles is going to be 9. Note the color of nine is red so we are going to reassign the color of x.parent to black, thus 27 is going to be black now. Then we change the color of the uncle to black as well so 9 is now black. Next x.parent.parent will become red so 16 is now red. Finally, we assign x to the grandparent of x so x is now 16. Now when we

reevaluate the while loop it evaluates to false since the color of 16 is black so we can skip the loop. Thus, the tree we get the following tree:

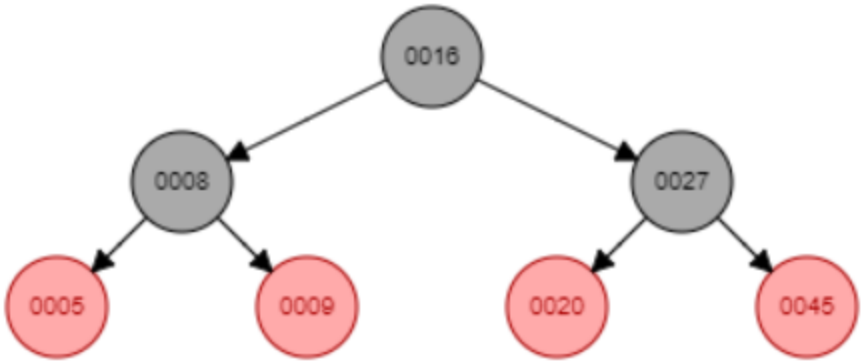


Now we can use the delete algorithm to get rid of 67. In this case the node we wish to delete is not the root and it has no left or right children so the deletion for this case is simple. We need to assign the parent right child to null and then assign x to null. Next, we must see if the tree needs to be rebalanced but since our node color is not black we don't need to do that and we can just delete the node. Thus, the final resulting tree looks like the following and notice how it is different from the original tree:

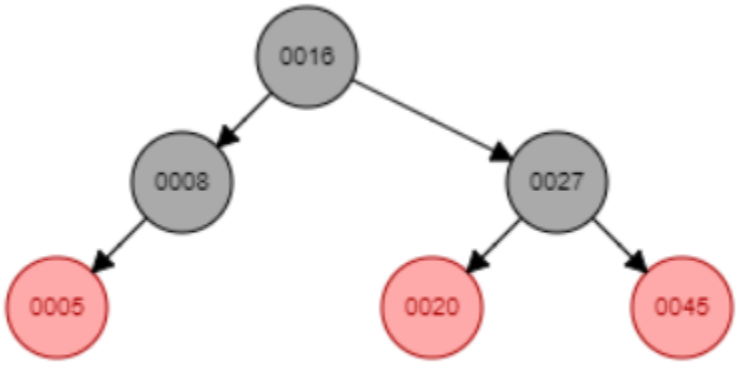


Question 2:

Deleting a node with no children and then re-inserting it will not always result in the same tree. The only time you will get the same tree is when the node you which to delete is at the lowest level of the tree. Take for example the tree depicted below:

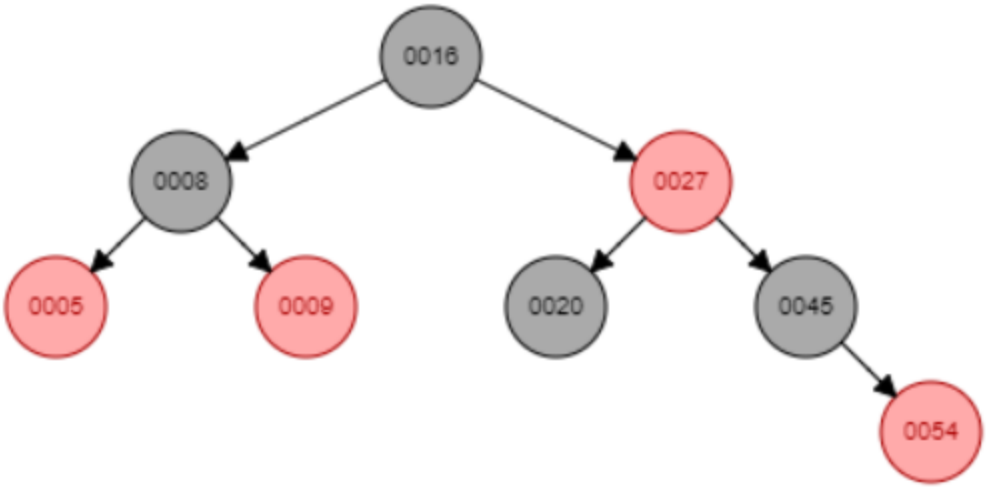


If I wanted to delete 9, we would first look for which case in the deleting algorithm the node matches. In this case the node has no children so we can just set the parent's right child equal to null and delete the node. The parent of 9 is 8 so 8's right child will become null. Resulting in a tree that looks like:

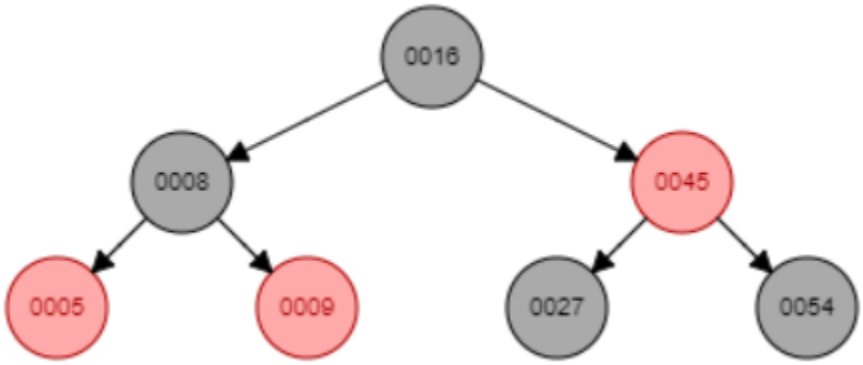


Now we can follow the insert algorithm to see how to properly insert 9 back into the tree. First, we insert 9 and then see if any violation needs to be fixed. Since 9 is not the root of the tree and 9's parent (8) is not red, no violation has occurred and the tree is fine. And finally, we get a tree that is identical to the tree we started with.

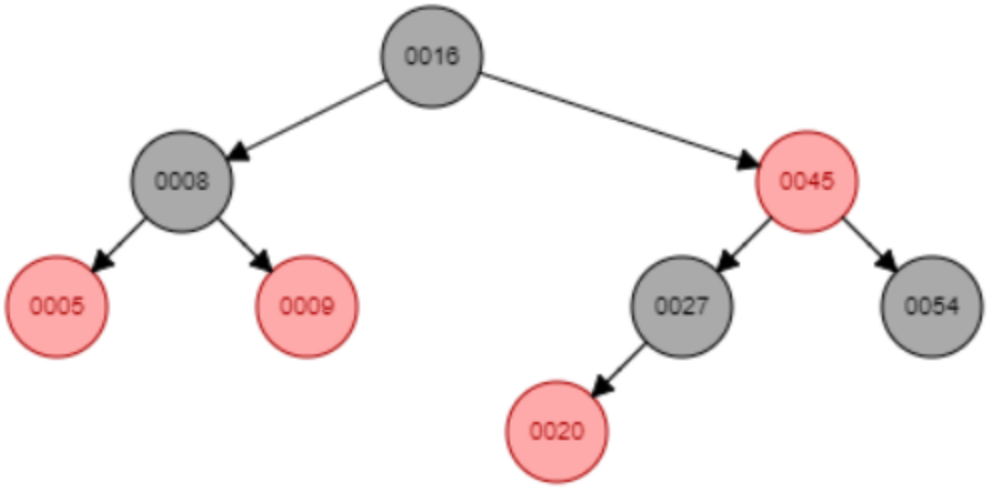
Now lest consider a case when the final tree is not identical to the tree we started with. Let uses the tree we had in the last example but this time add another node, 54. Now following the insertion algorithm, we will get a violation error when we add the node since 54 will be red and 54's parent (45) is red. Also, not that 54's grandparent is 27 and 54's parent is the right child of 27 so we can assign the uncle of 54 to be 20. Now that we know his uncle we can see that the uncles color is red so we change the color of the parent and uncle to black and change the color of the grandparent to red and then we assign x to the grandparent. Now all violations have been fixed and we get a tree that looks like the following:



Now let's delete 20. Now deleting the node is simple, we just needed to assign the parents left child to null and delete 20. Thus 27's left child is now null. However, since the node was black the tree needs to be rebalanced. Now let's look at the rebalance algorithm to see how the tree needs to be rebalanced. In our case x is in fact the left child of his parent so we assign s to the right child. Now s is 45. Note 45's color is black and only has 1 child so the first three cases can be ignored. So, we are in case 4. So, we assign 45 color to the color for 20's parent, thus 45 is now red. We will then assign the parent color to black so 27 is now black. Also, we will change the color of the right child of s to black thus 54 is now black as well. And finally, we will left rotate using the parent of x. Thus 45 is now the new root of the sub tree and 27 becomes the left child of 45 and 54 becomes the right child of 45. So, our new tree looks like the following:



Now let's reinsert 20. Following the insertion algorithm, we insert 20 as a red node. 20's parent is 27 and his uncle is 54 and finally his grandparent is 45. Note, 27's color is black not red so while loop in the algorithm is false so no violation needs to be fixed and the insertion is done. 20 is inserted as the left child of 27. And the final tree we get is not the same as the initial tree. The finally tree will look like the following:


0016
0008
0045
0005
0009
0027
0054
0020