Red-BlackTrees

This is the final balanced binary tree structure we are going to study. Although it won't seem like it at first, these trees are quite similar to 2-4 Trees. Here are the rules for a valid Red-Black Tree:

1) **Must be a valid Binary Search Tree, with each node colored red or black.**
2) **The root is colored black.**
3) **The children of a red node are black.**
4) **Every external node is black. (These are the "null nodes" that are the children of the leaf nodes.)**
5) **All the external black nodes have the same *black depth*. The black depth of a node is defined as the number of black colored ancestors minus one.**

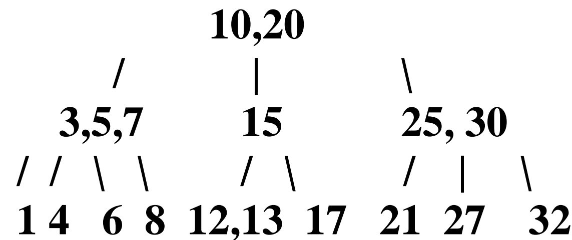
Before we go on, let's go over a method to convert a 2-4 Tree into an equivalent Red-Black Tree:

Each 2-Node in a 2-4 Tree should be colored Black in the analogous Red-Black Tree.

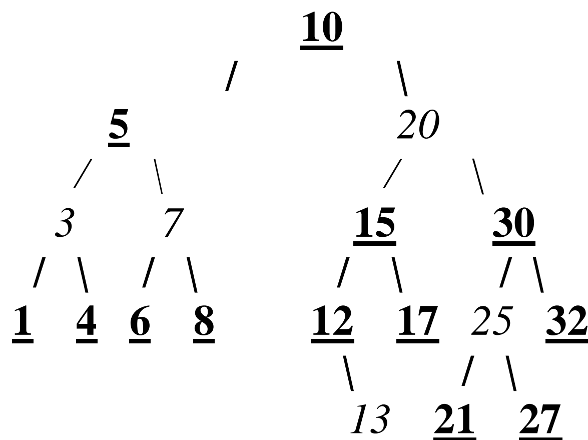
Each 3-Node in a 2-4 Tree should be converted into two nodes in a Red-Black Tree, with the parent node being black and the child being red.

Each 4-Node in a 2-4 Tree should be converted into three nodes in a Red-Black Tree, with the parent node being black and the two children being red.

Example of a 2-4 and corresponding Red-Black Tree



Using the algorithm above, we have (black nodes in bold underline, red in italics):



Height of a Red-Black Tree

In the best case, a Red-Black Tree of n nodes is perfectly balanced and its height is at least log (n+1). For the worst case, we can look at the corresponding 2-4 Tree to any Red-Black Tree of n nodes. Let T' be the corresponding 2-4 Tree for a Red-Black Tree T. The maximum height of T' is log(n+1). Since any path in a 2-4 Tree is expanded by at most doubling the number of nodes on the path in the corresponding Red-Black Tree, it follows that the maximum height of T is 2log(n+1). This shows that the insert, search and delete operations in a Red-Black Tree run in O(log n) time. (This is because each of these operations runs in time proportional to the height of the tree.

Insertion into a Red-Black Tree

Initially, when we insert an element in a Red-Black Tree, we perform a normal binary search tree insert and place the node, coloring it red. (Why is this a better option than coloring it black?) The only exception to this is if the inserted node is the root node. In this case, the node is colored black.

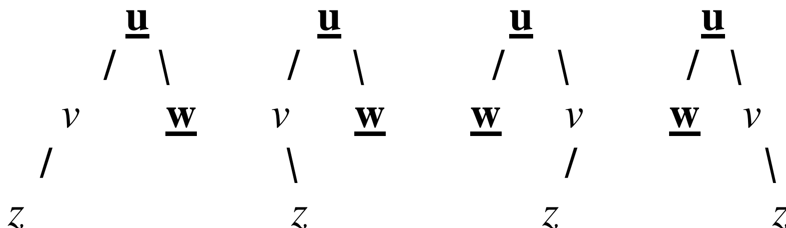
Sometimes, this will cause no problem. First of all, no black depths of external nodes change. Next, as long as the inserted node's parent is colored black, there is no violation of the red nodes must have black children rule.

Thus, we only have problems when the parent of the inserted node is red. Denote the inserted node as z, the parent of the inserted node as v, and the grandparent of the inserted node as u. Finally, denote the uncle of the inserted node as w. We will break up our work to fix this situation into two cases:

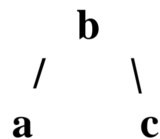
1) w is Black (Initially will be null node.)
2) w is Red

Case 1

Let's deal with the first case. Structurally, we have four possibilities for the portion of the tree with the inserted node, parent node, and grandparent node:



In each of these situations, you can relabel the nodes u, v, and z in their inorder order, a, b, and c, with $a < b < c$.
In this case (for all four of these structures), the restructured tree will look as follows, with w inserted as a child of a or c, as appropriate:



The node storing b is colored black while the nodes storing a and c are colored red. This takes care of the double red problem. The node storing w will be one of the four subtrees that have either a or c as a parent. The reason this works is because previous to the restructuring, all four subtrees had an equal number of black nodes down any path. This number for each external node remains unchanged, as the "root" node of the group remains black, still adding to the black depth of each external node underneath it.

Case 2

This corresponds to the case of an overflow in a node in a 2-4 Tree. The initial way to deal with this problem is as follows:

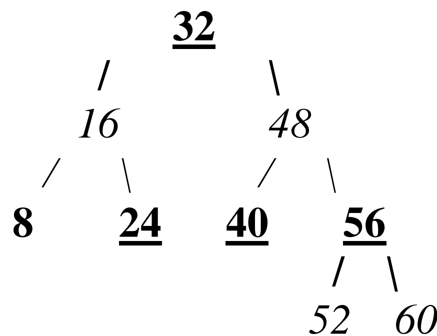
Using the same variable conventions as before, with w being a red node, recolor u, v, and w. In particular, make u red. But in doing this, we need to adhere to the rule that the children of red nodes are black. Thus, both v and w should get colored black. Luckily, this solves two problems: Now, the black depths of each of the external nodes below v and w is restored to its proper value, AND there is no double red occurrence in the subtrees rooted at v or w.

But, the problem that is introduced is that u may change to become a double red node, since it was black previously, and could have had a red parent.

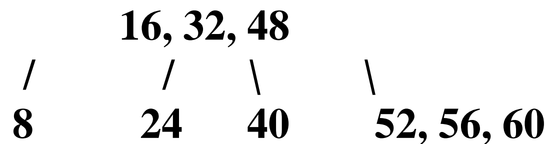
Now, we can deal with this issue at the node u, treating it as the node to restructure/recolor. As long as we continue to fall into case 2, we will just perform recolorings. If we ever fall into case 1, we will perform a restructuring, completing the insertion. (Note: if we ever follow this chain up to recoloring the root node red, we simply stop from doing that. The reason for this is that changing the color of the root node does NOT affect the relative black depth of any node in the tree.)

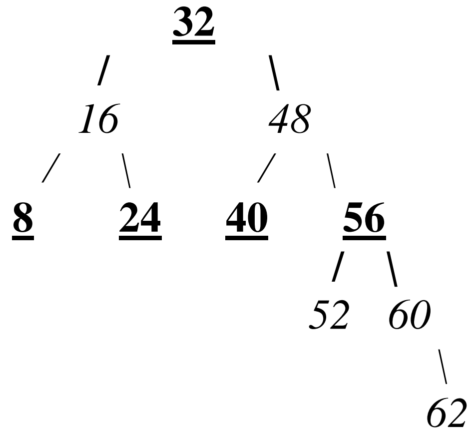
Case 2 Example

Consider inserting 62 into the following Red-Black Tree:

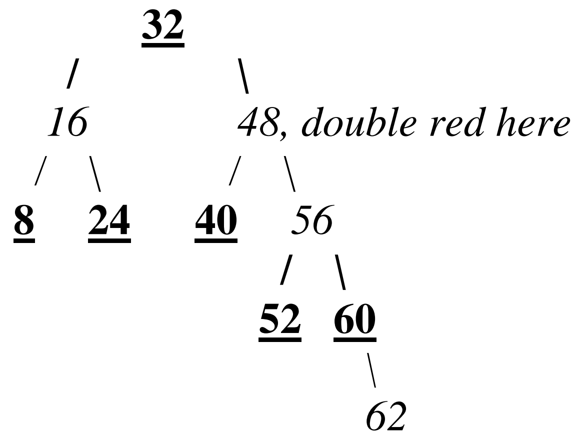


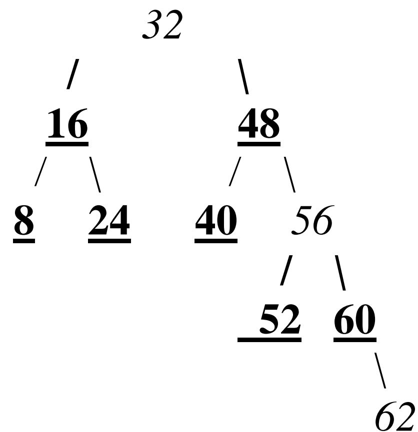
Note: The corresponding 2-4 Tree is as follows:



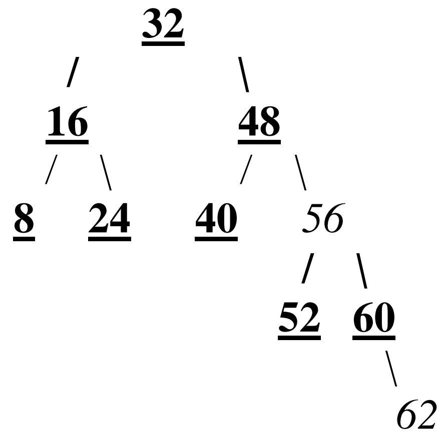


Initially, we will color the 52 and 60 black, while recoloring the 56 red. This will then cause another double red problem which can be fixed by another recoloring

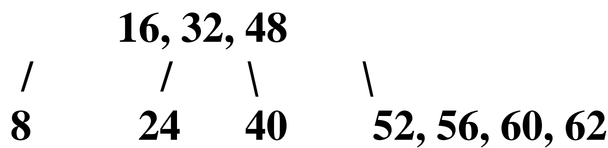


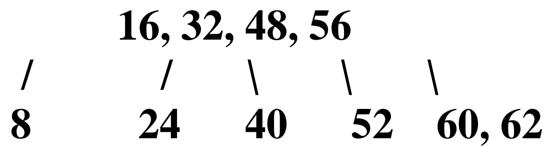


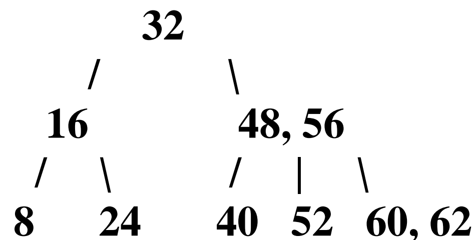
The problem with this tree is that the root is red. Of course, we could simply have never changed its color to red:



Consider the corresponding 2-4 Tree insertion:



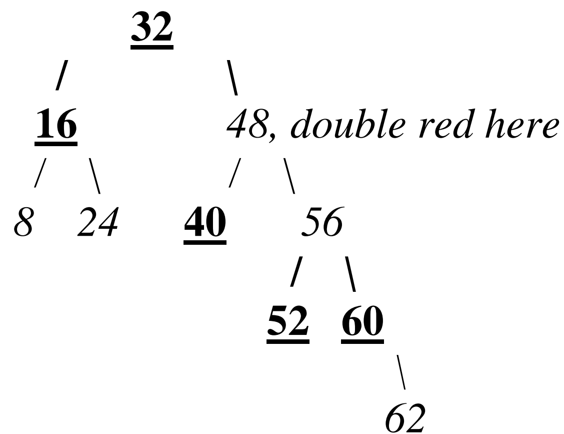




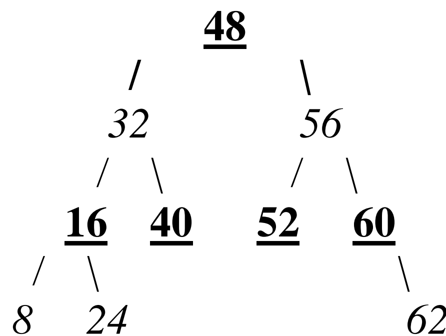
Which is the exact same tree you get when you convert the result Red-Black Tree into a 2-4 Tree. (The recolorings are equivalent to pushing a node up to a parent node in a 2-4 Tree.)

Case 1 Example

Now consider the same example above, except where 16 is black, and both 8 and 24 are red. Everything will be the same until you get to the second step when both the 48 and 56 are red. Here, 56's uncle, is a black node 16, which means we are in the first case and not the second.



In this situation, we will make 48 the root (and make it black), with a left child or 32 and a right child of 56, both of which will be red:



Notice how just changing the colors in a Red-Black Tree and NOT the structure will change the structure in the corresponding 2-4 Tree.