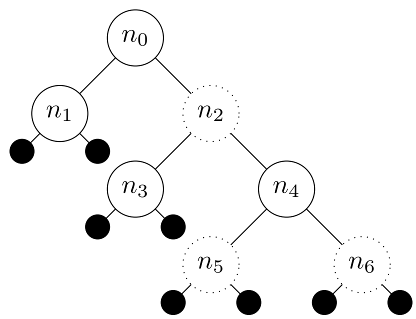
1 The Rules

1. All nodes must be either red or black
2. The root and leaves must be black
3. Red nodes cannot be adjacent
4. All simple paths from a node (exclusive) to a descendent leaf must have the same number of black nodes

1.1 Black Height

Count all black nodes along a simple path from one node to all its descendent leaves.



$$bh(n_1) = bh(n_3) = bh(n_5) = bh(n_6) = bh(n_4) = 1$$

$$bh(n_2) = bh(n_0) = 2$$

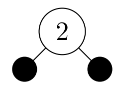
1.2 2-3-4 Relationship

- All red nodes merge with parents to form a 3 or 4 node.
- Corresponding 2-3-4 tree has the same height as the black height of the red-black tree.

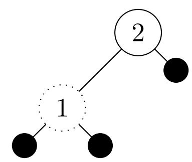
2 Insertion

Keep adding a node strictly less than all other nodes in the tree. This leads to imbalance; why? (I did not specify a color for that node, violating rule 1.)

2.1 Case 1

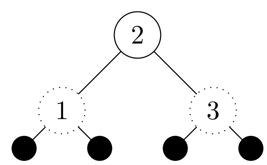


Insert the value 1 into this tree:

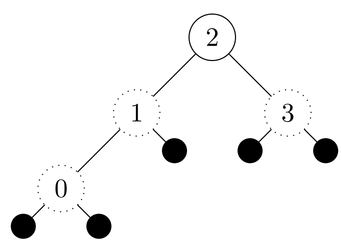


We are done. This satisfies all properties of a red-black tree!

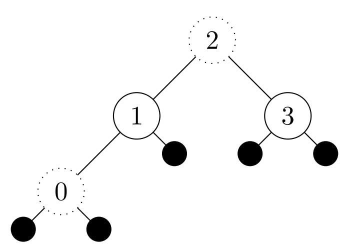
2.2 Case 2



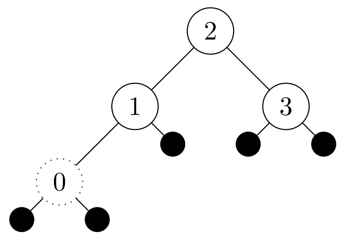
Insert a 0 into this tree:



This violates rule 3, the 0 and 1 nodes are both red, and adjacent. In this case, the new node's "uncle" is also red, so we can recolor the graph:

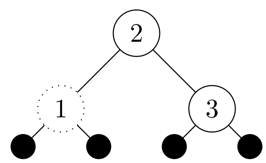


We've now violated a new rule; rule 2. This is not a big deal though, we can recolor the root without violating rule 4:



In general, you have to work back up the entire tree, starting at the "grandparent", fixing rule violations as you go. Here the grandparent was the root, so we only had to fix one thing.

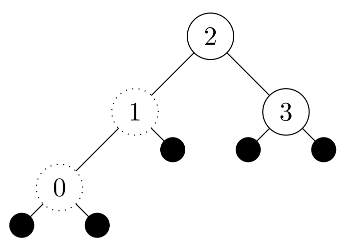
2.3 Case 3



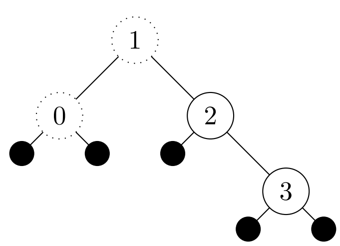
First off, is this a proper red-black tree?

- When inserting, we only care about a small portion of the subtree.
- This, and most future examples, will not themselves be proper red-black trees.

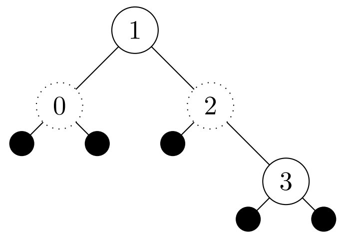
Insert 0 into this tree:



This violates rule 3 again, but the "uncle" node is black, so we cannot recolor. We must first rotate:

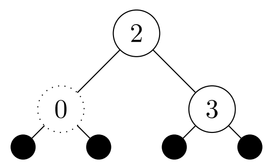


Then, we may "invert" the color of the rotated nodes (1, 2):

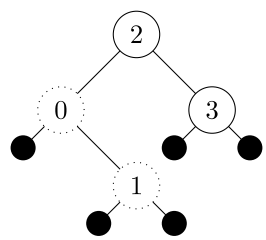


Here, since we have a black node at our "root", we're done.

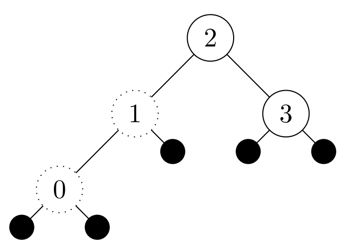
2.4 Case 3, Part 2

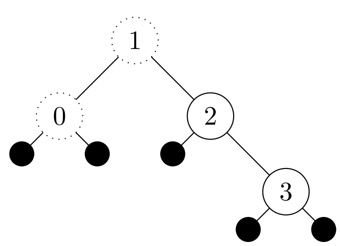


What if we inserted 1 into this tree?

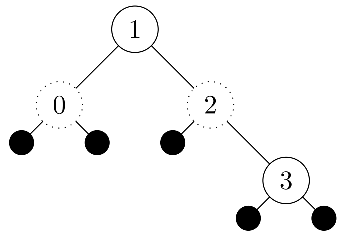


Looks similar to the previous situation, but we need to do a left-right rotation before recoloring:



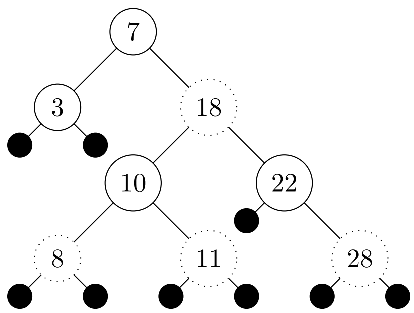


Then, invert colors as usual:



There are mirrors for right-right, and right-left imbalances.

3 Putting it All Together

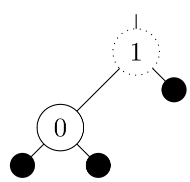


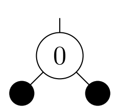
Insert 15 into this tree.

4 Deletion

4.1 Case 1

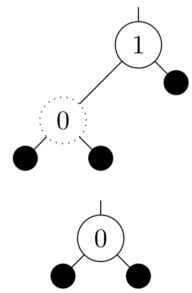
If you are deleting a red node with a black child, simply move that child up into the deleted node's place.





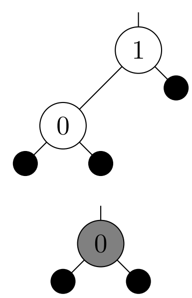
4.2 Case 2

If you are deleting a black node with a red child, simply move that child up, and recolor:



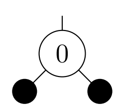
4.3 Case 3

If they are both black, there is a potential problem. We have to introduce a "double-black" node to keep track of where this problem exists.



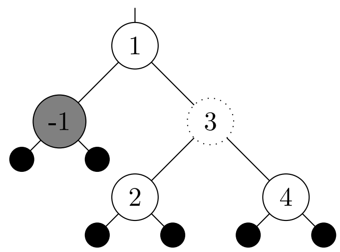
4.3.1 Case 3a (Terminal)

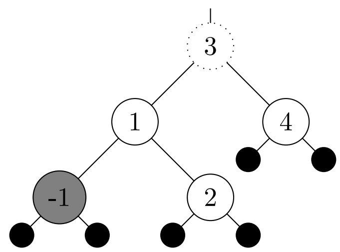
If the node to be deleted was the root, we're fine. The black height just reduces by one.



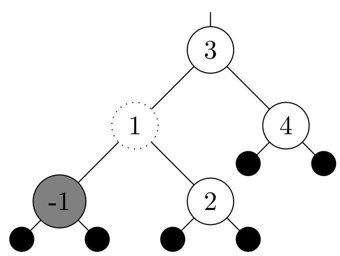
4.3.2 Case 3b

If there is a sibling node that is red, we have to rotate to eliminate the double-black node:





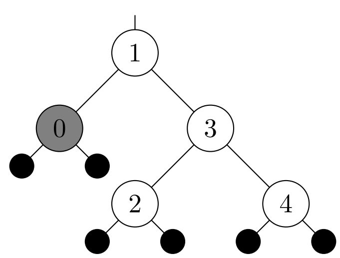
Then, we have to recolor the 1 and 3 nodes.

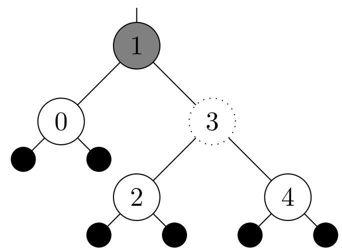


We have to now continue to a new case.

4.3.3 Case 3c

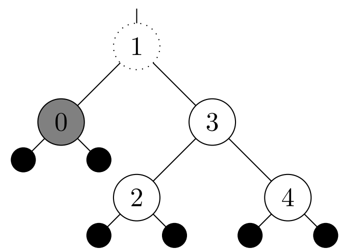
If the sibling node is black, and both of its children are black, recolor the sibling and move up



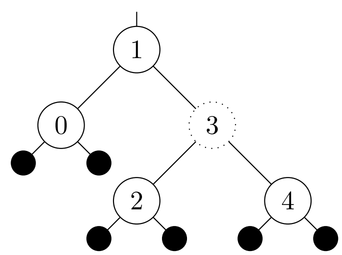


4.3.4 Case 3d (Terminal)

If the sibling is black, and the parent is red:

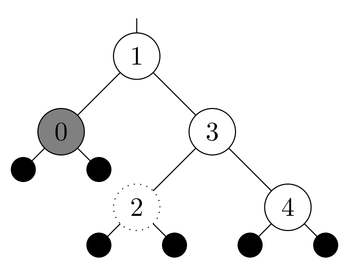


Recolor, and you're done:

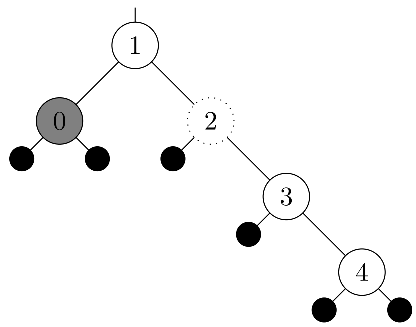


4.3.5 Case 3e

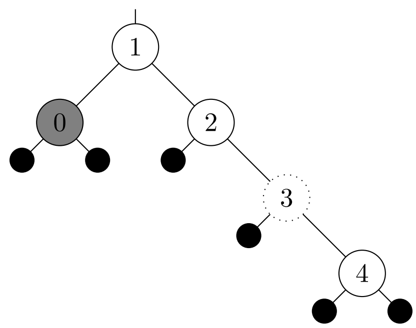
If the parent, sibling, and sibling's right child are black, but the sibling's left child is red:



You must rotate to make the sibling's right child red:



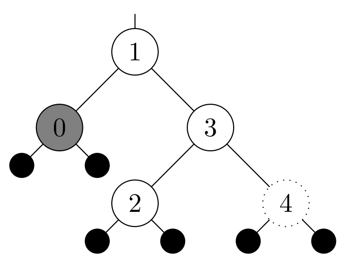
Don't forget to recolor:



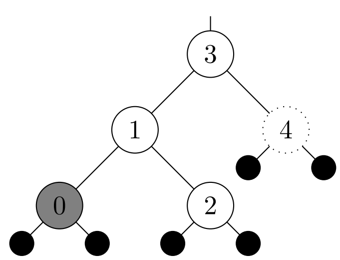
Then, continue on to the next case.

4.3.6 Case 3f (Terminal)

If the parent and sibling are black, but the sibling's right child is red:



You must rotate (to the left) around the parent:



Recolor, and you're done: