R-B Tree Insert Summary 2/3/2022

1. Insert node as red.
2. If parent is black, no further work is necessary.
3. PROBLEM CASE: Parent is Red.

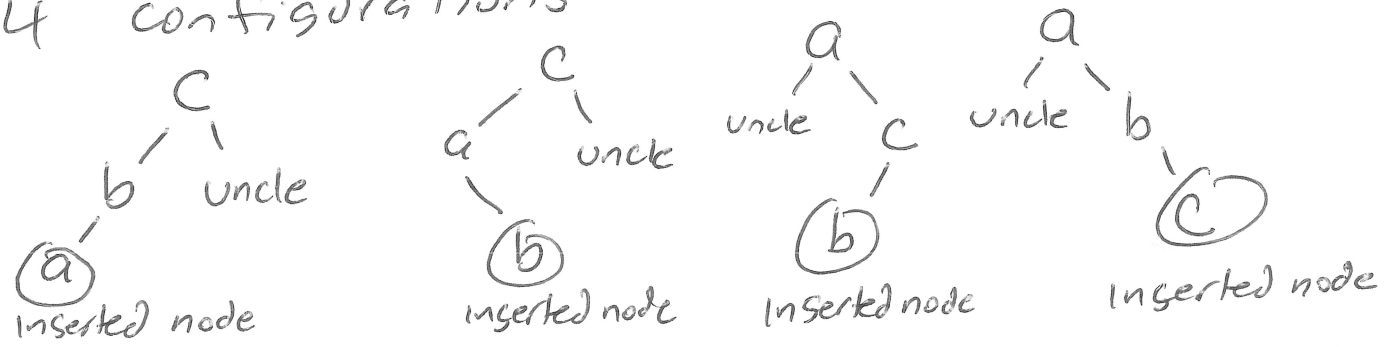
~~Case~~ PROBLEM CASE

If parent is red, look at uncle (could be null, in which case it's black. Work splits into 2 cases:

Case 1: Uncle is Black
Case 2: Uncle is Red.

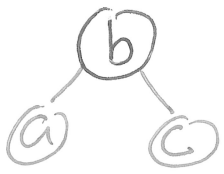
~~Ca~~ Case 1

4 configurations



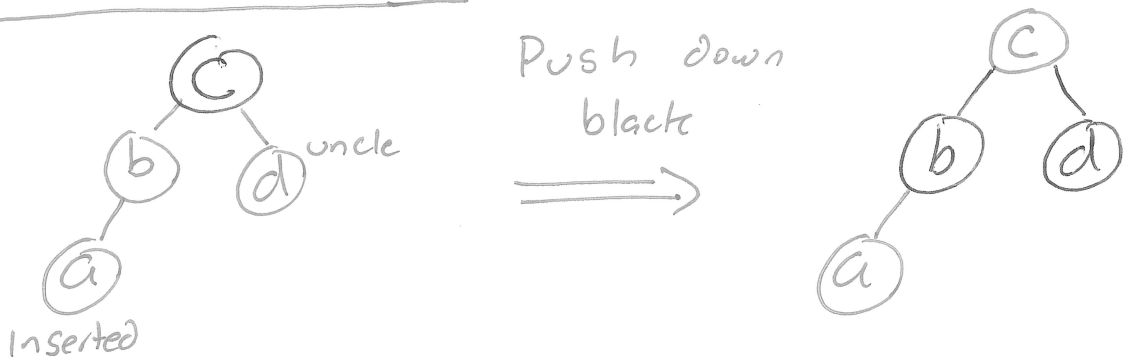
I assign $a < b < c$, and inserted node is on the bottom level, other 2 nodes are parent, grandparent, w/ parent = red, inserted = red, grandparent = black, uncle = black

In all four cases, we fix this way:



Uncle attaches as necessary by bst ordering.

Case 2: Red Uncle



No structural change, but now c's parent might be red.

R-B Delete

Initially do a regular BST delete

3 (1) Leaf node (Black)

1 ~~(2)~~ Leaf node (Red)

(2) Black node with a Red Child

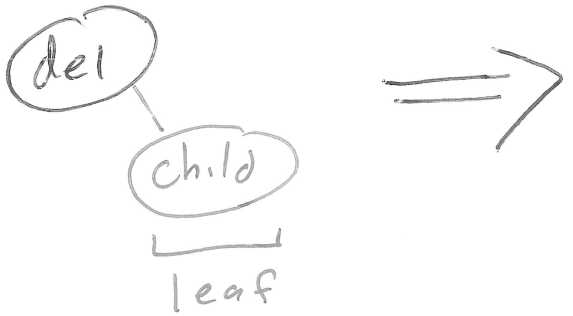
Note: Structurally, we never delete a node with 2 children. Replace w/ largest on left or smallest on right.

Note #2: There will never be a red node with one black child because that would be imbalanced (black height).

Case 1: Red Leaf Node

Regular Delete Works.

Case 2: Black Node w/ Red Child

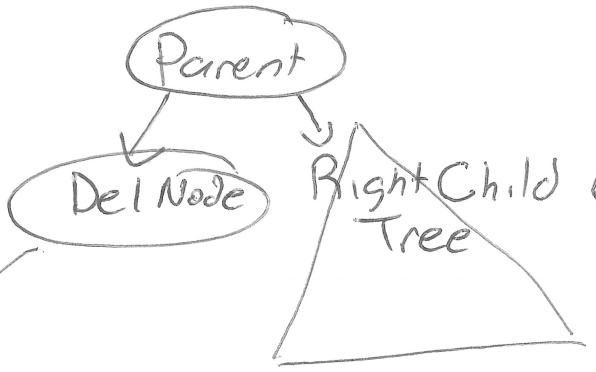


We color black for 2 reasons.

1) Maintain black height

2) ~~E~~ Deleted node's parent may be red.

Case 3: Black Leaf Node

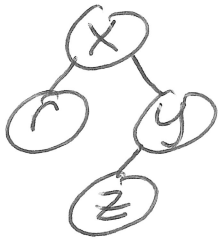


I could delete, leaving "null node" black.
But, black height not preserved, so we will temporarily color it "Double Black".

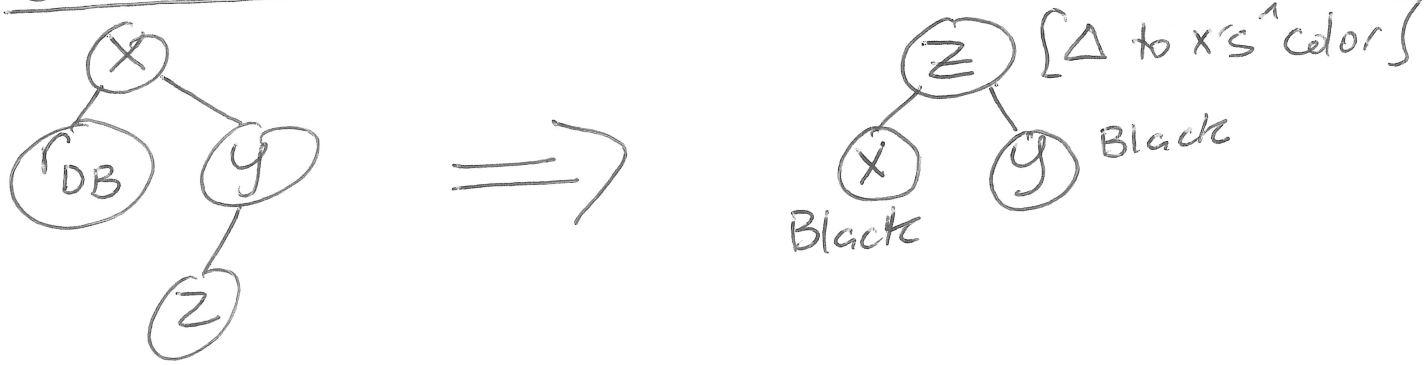
Cases for DOUBLE BLACK NODE

1) Sibling is black and has a red child.
2) Sibling is black and has 2 black children.
3) Siblings is red.

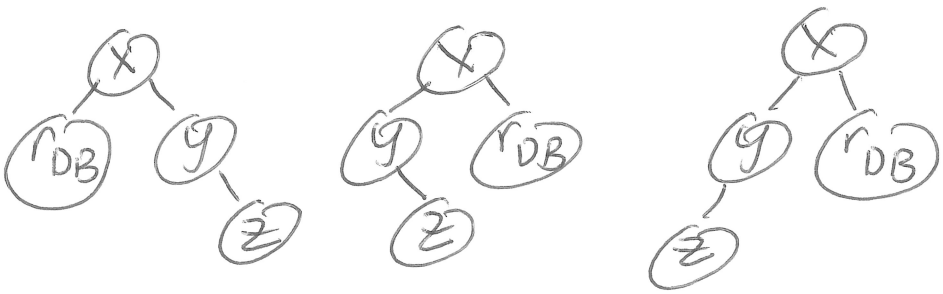
Child of del node = r (init null node)
Sibling of r = y
Child of y = z
Parent of y = x



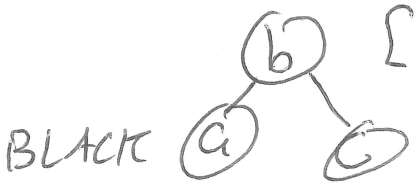
Case 3a: Sibling is black w/ red child



3 OTHER PICS:

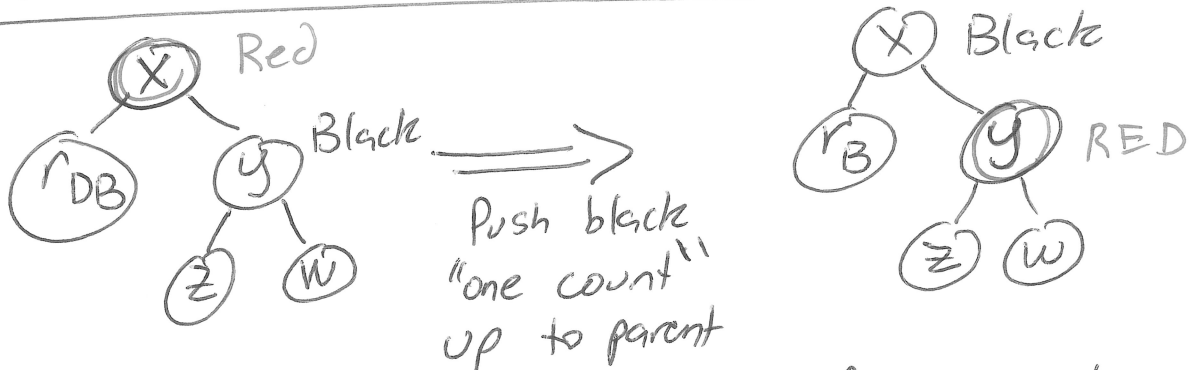


If we relabel $a < b < c$, in all pics



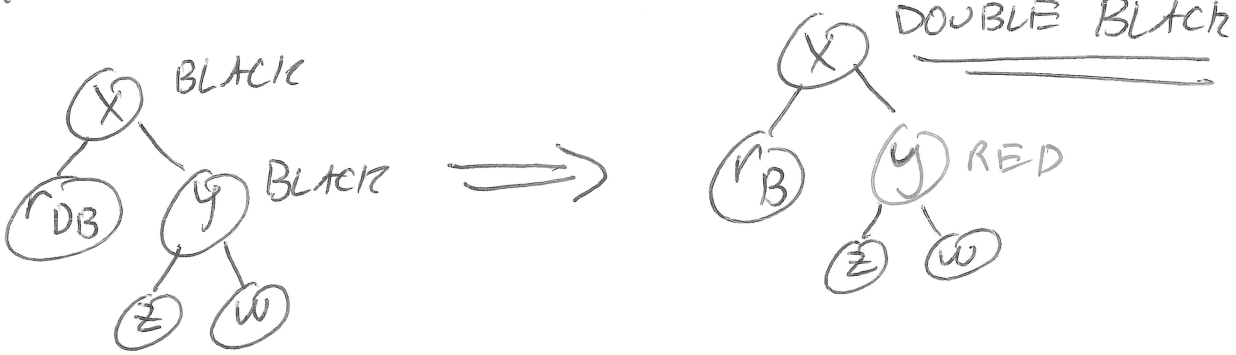
[original color of double black node]

Case 3b: Sibling is black and has 2 black children



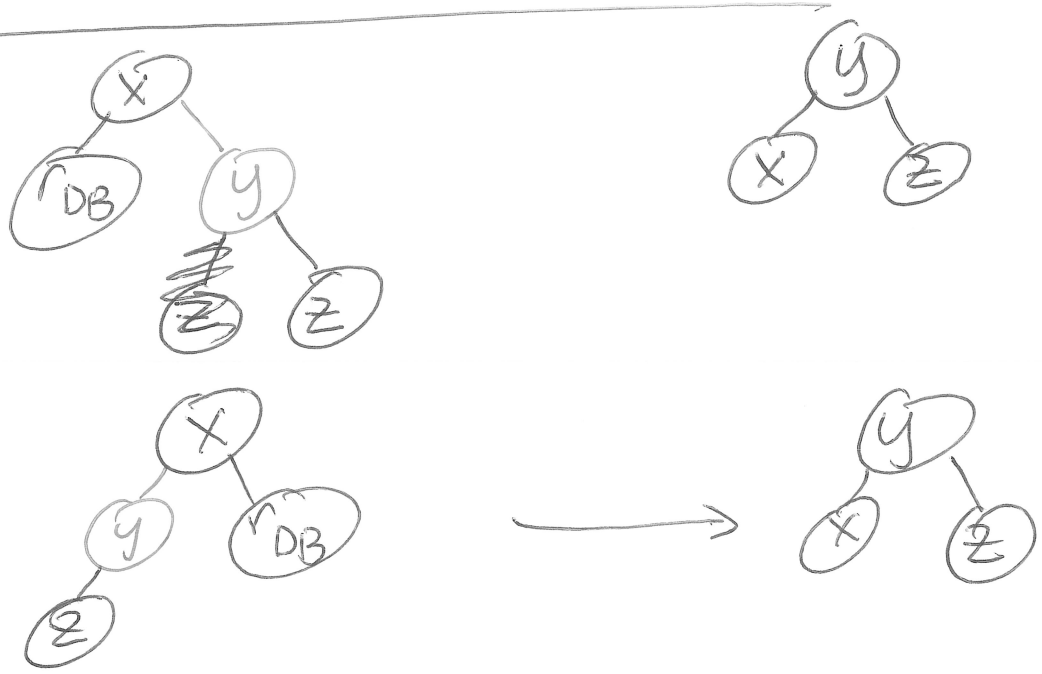
This is like the transfer operation.

If x was black to begin with, we have

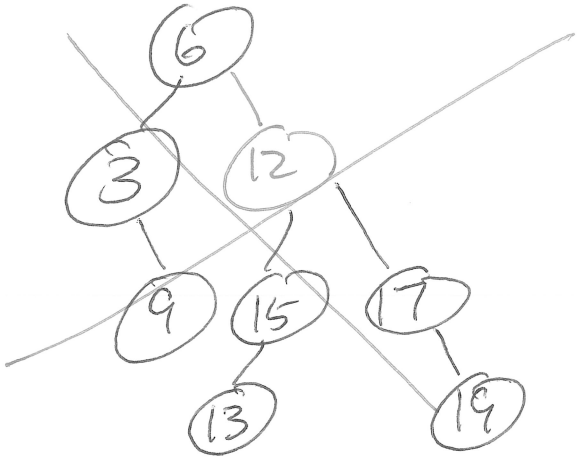


So here, we push the double black up one level.

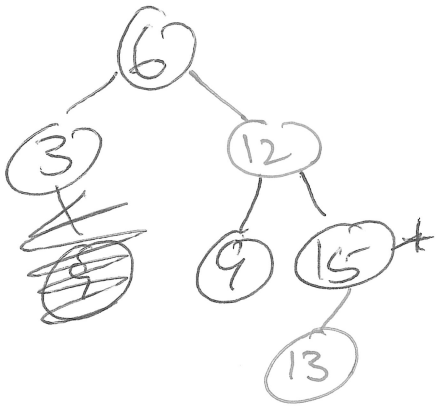
Case 3c: Sibling is Red



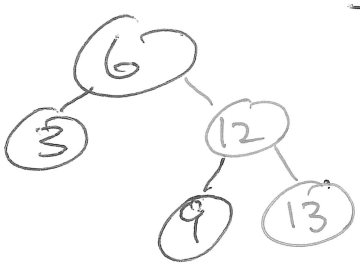
Examples



Delete 13

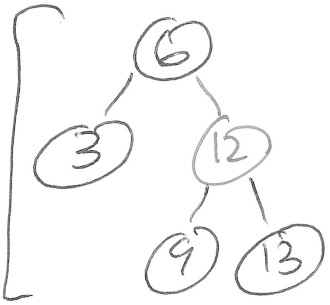


Delete 13 → just erase
or Delete 15

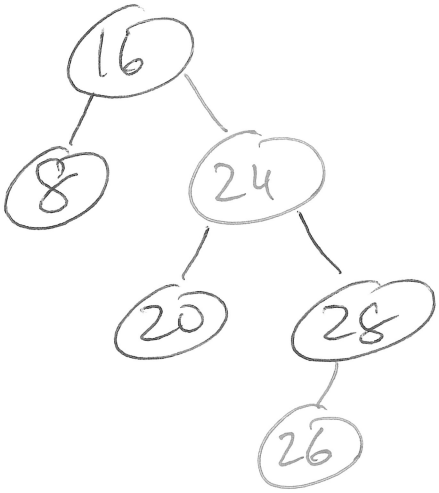


Case 1

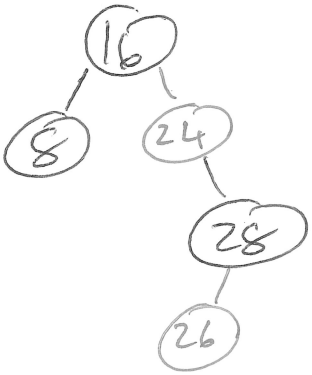
Case 2

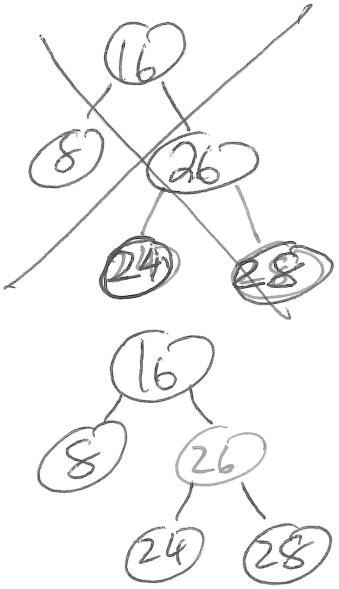


Recolor 13 to black

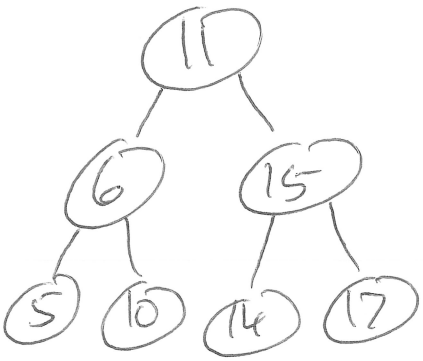


Delete 20

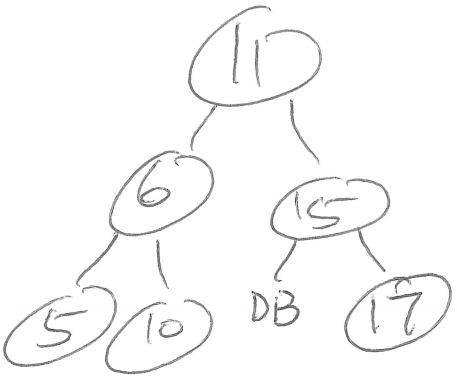




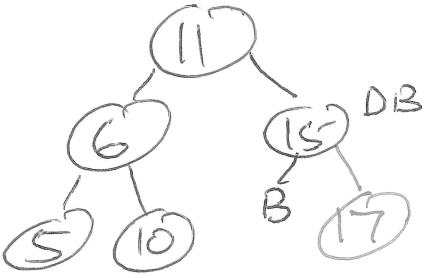
Example



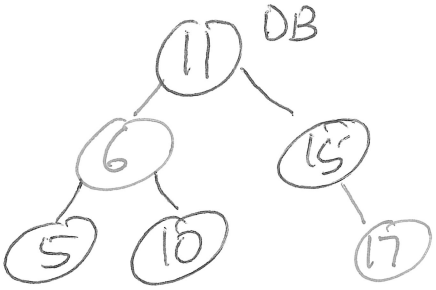
Delete 14



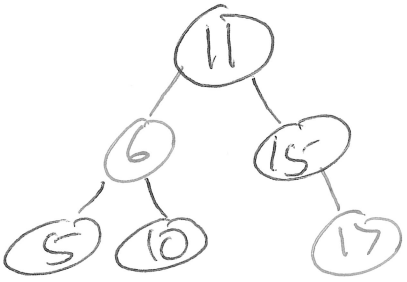
Case 3B

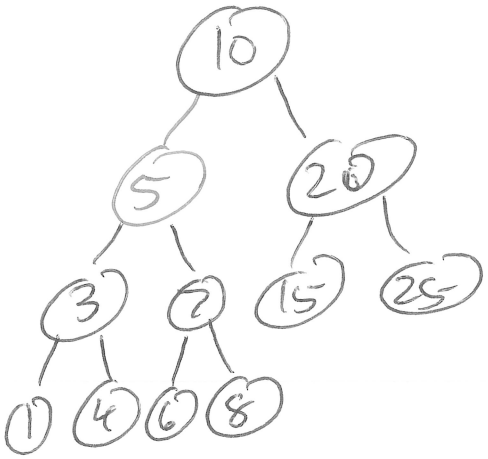


Case 3B

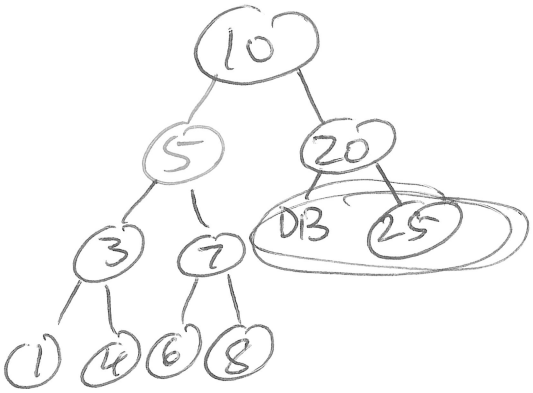


finish

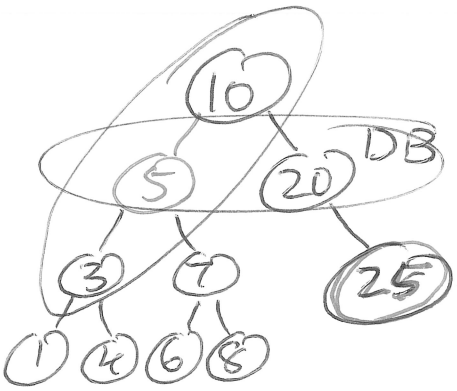




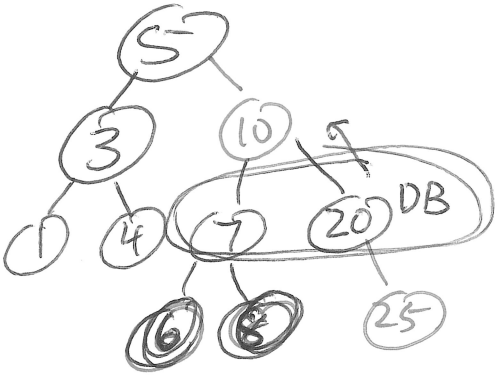
Delete 15



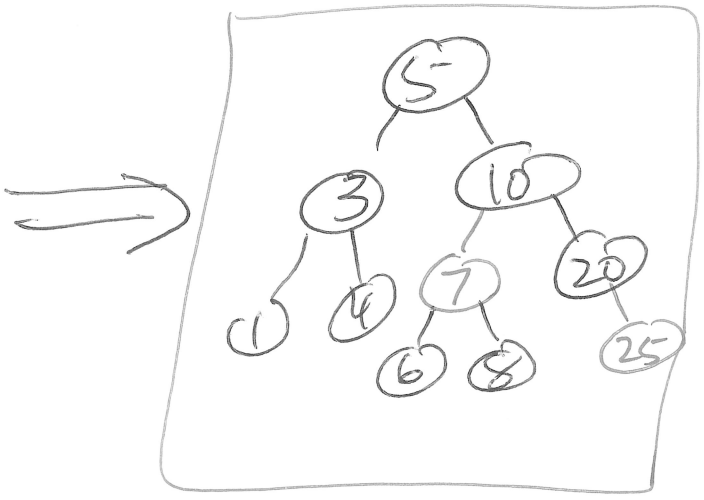
Recolor Case 3B

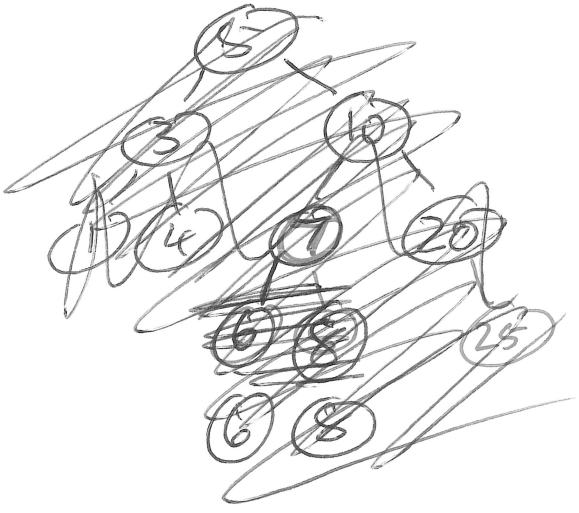


Rotate Case 3C



Case 3B





R-B Tree Delete

Easy case 1: Red Leaf Node ⇒ NO OTHER ACTION

Easy case 2: Black Node w/ONE red child
⇒ (a) Connect parent of deleted node to child
(b) recolor child to black to maintain black height.

Hard case 3: Black ~~#~~ Leaf Node

Initially, we delete the node and color its "stub" Double Black. From here, the DB status may "percolate up" the ancestral path + may include structural changes (and/or coloring changes) Here are the cases of what to do with a Double Black node:

Case (3a) DB node has black sibling with ≥ 1 red child.
- rebalance operation completes the work.

Case (3b) DB node has black sibling with 2 black children
- Subtract 1 from black count of DB and black sibling, add 1 to black count of parent of old DB node. (Will change black sibling to red and parent of old DB might become DB, propagating the error up.

Case (3a) DB node has red sibling

- rebalance operation will propagate into either Case 3a or Case 3b.