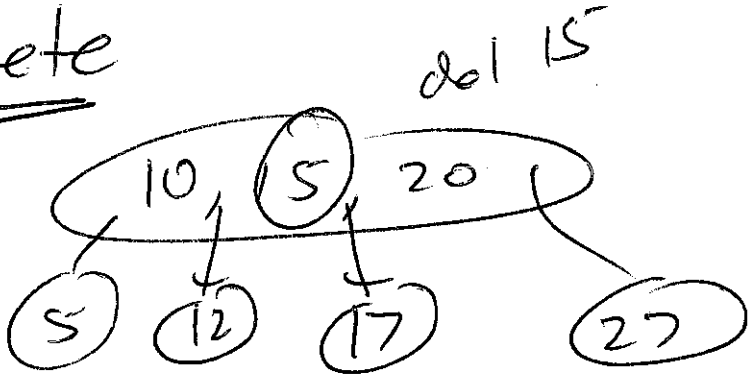
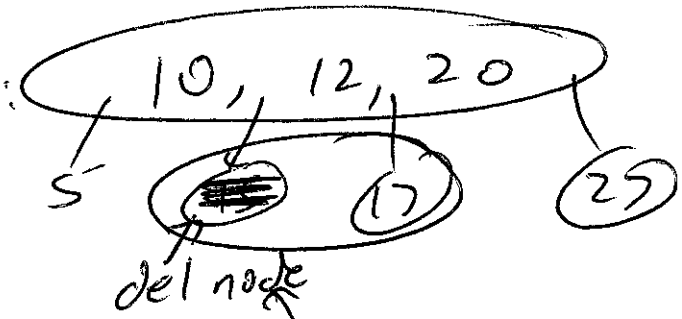
2-4 Trees

Delete



Swap 15 and 12 (largest value < 15)

Temp Pic:

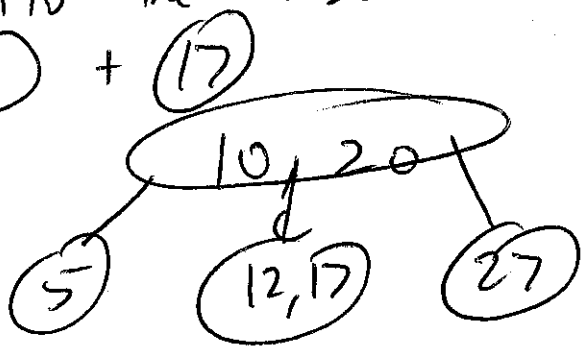


Delete this node (it's on the bottom level)

If adjacent sibling has > 1 value, do a transfer

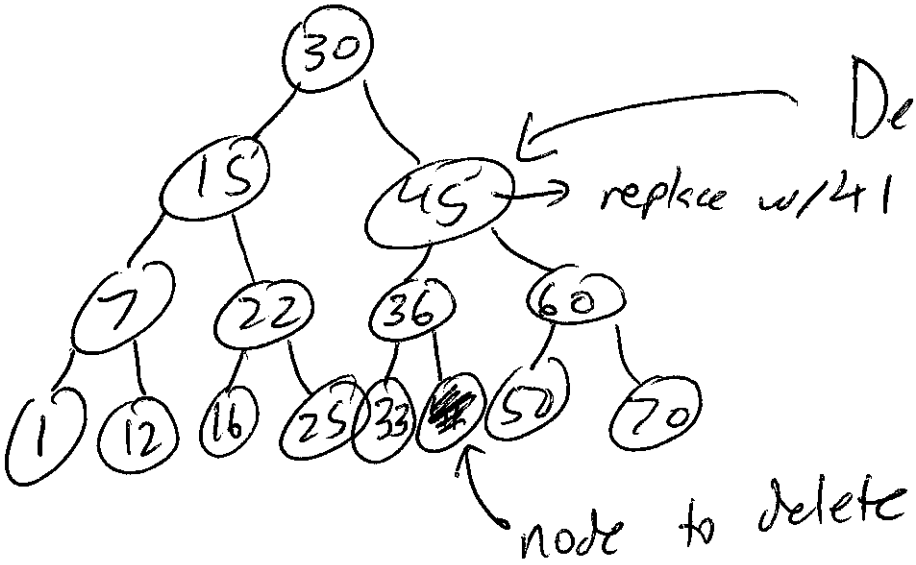
But, if no adjacent siblings has > 1 value, we are safe to drop a value from the parents into the fused node.

fused = ○ + 17



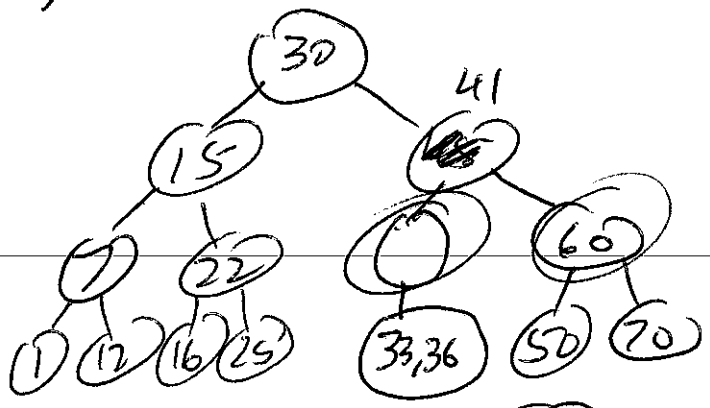
If we drop a value down, the top may become empty!

2/1/18 (2)



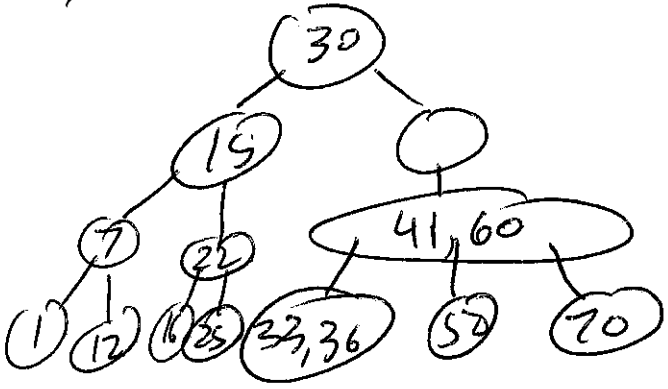
1) fuse 33 + ∅

2) drop 36 down



3) fuse ∅ + 60

4) drop 41 down



5) fuse 15 + ∅

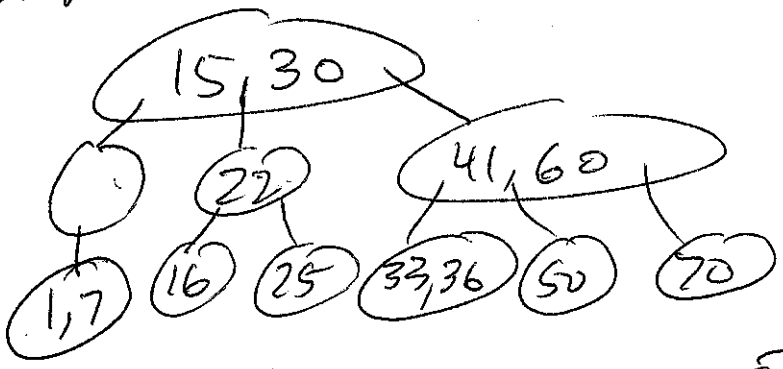
6) drop 30

Get rid of empty root

Delete 12

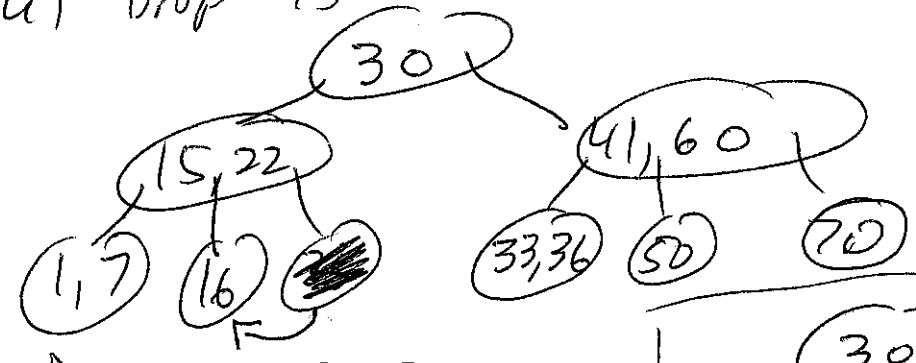
1) fuse (1) + (12)

2) Drop 7 down



3) Sibling has 1 ust, fuse (7) + (22)

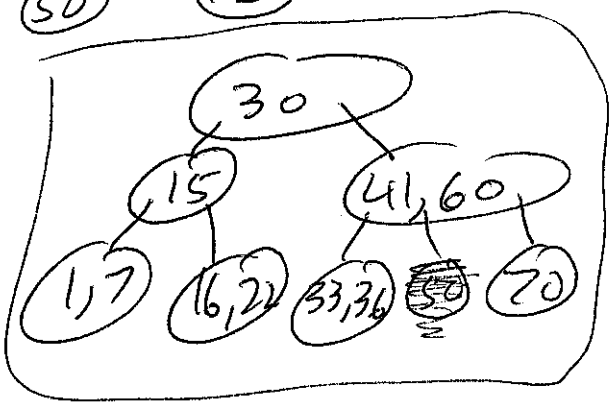
4) Drop 15



Delete 25

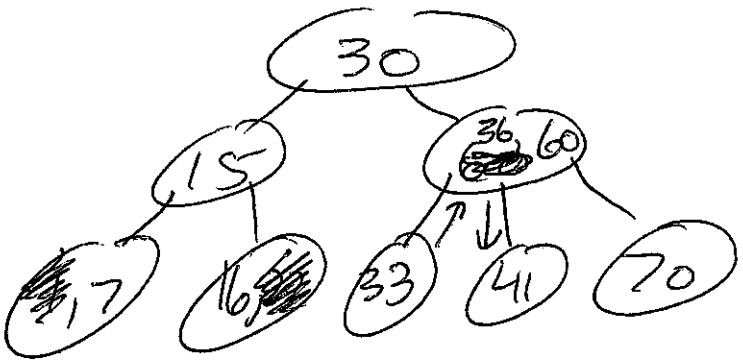
1) fuse (16) + ( )

2) Drop 22



Delete 50 2/1/18 (4)

(1) Can transfer because (33,36) adjacent sibling has > 1 value



36 ↑
41 ↓

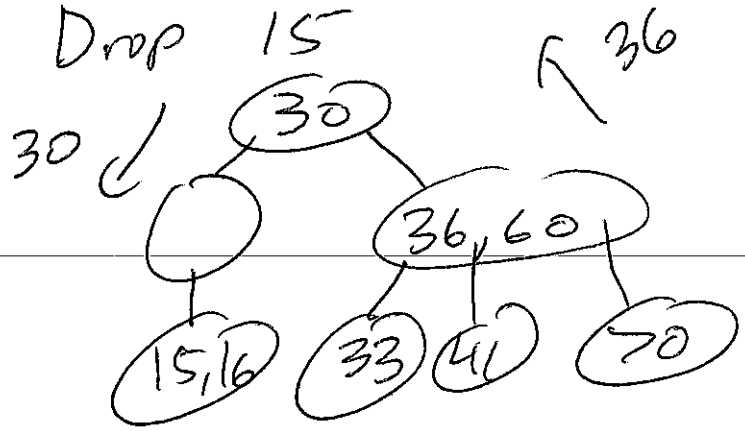
Delete 22

Delete 1

Delete 7

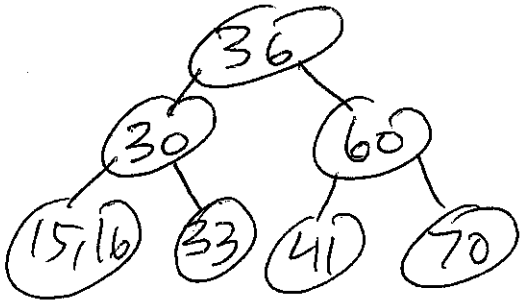
1) fuse ( ) + (16)

2) Drop 15



3) Siblings of ( ) has > 1 value (36,60)
Adj.

transfer → send 36 up
send 30 down



Red - Black Trees

Structure wise exactly like a binary search tree with some added restrictions:

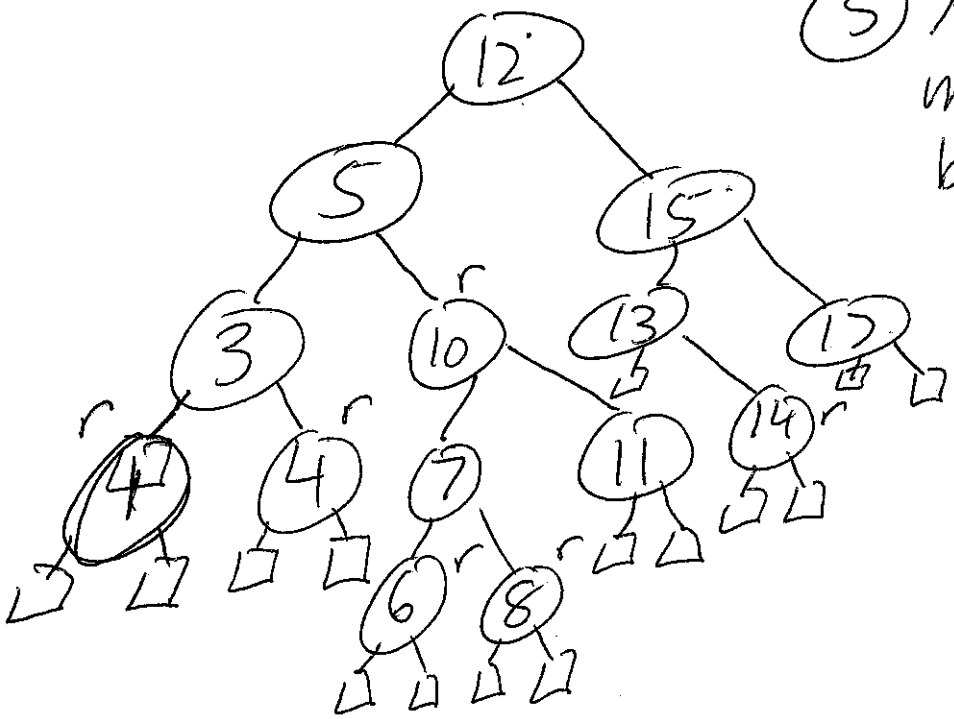
(1) each node has a color (red/black)

(2) root is black

(3) every external node (null children) is black

(4) The "black depth" of each external node is the same.

(5) All red nodes must have black children.



Insert

1) Easy case - insert as child of a black node

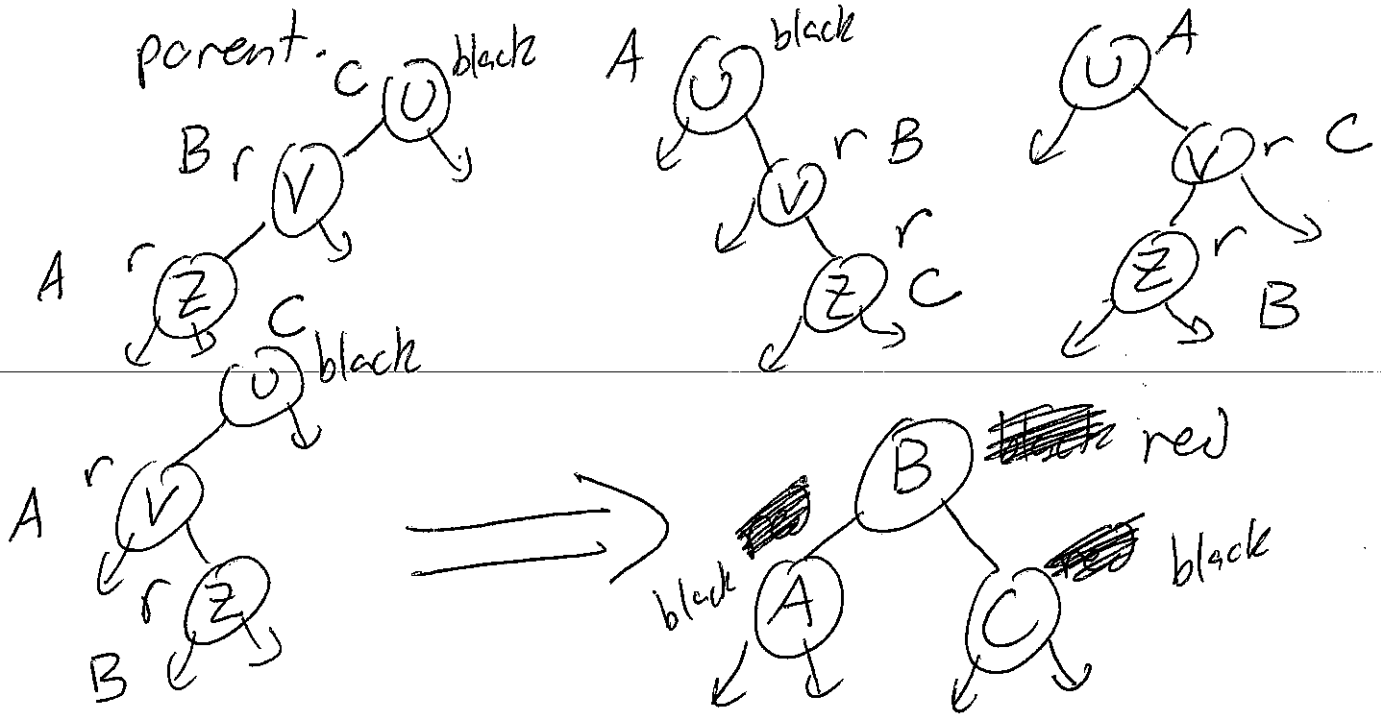
Just make red + stop.

Insert 1

2) Harder case - parent is Red.

Still make node red at first.

Let z be inseted node, v be parent.



Do a recoloring + restructuring that pushes the red up a level. If it fixes the problem great! If not, repeat!