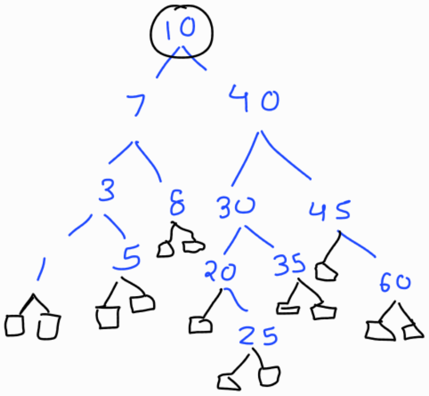
Red Black Tree

- BST and Height balanced Tree.
- At most 2 rotations required to balance or only requiring color-changing.
- Terminology – Red, Black, Terminals ■

Rules:

- Each Node is either red/black
- Root and all external are black
- No root-to-external-nodes path can have two consequtive red nodes.
- All root-to-external-node paths have same no. of black nodes.
- Red node has both children black.
- Black node can have either color children



height of Red-black tree having 'n' nodes is between $\log_2(n+1)$ and $2\log_2(n+1)$

Amortized Complexity → Overall sum total complexity of all operations we perform

↳ Complexity ⊛

Red-black : Algo :

```
Struct {
    Key,
    left, right, parent
    color
      ↳ default → red }
```

Steps :

1] → Check if tree empty.

2] → if empty, insert nn as root with color black & exit.

3] → if not empty, create new node, and insert as (BST insertion) leaf node with color red.

4] → If parent of newnode is black, exit from operation.

5] → If parent of NN is red, check color of parent node's sibling (Uncle of Newnode).

6] → If uncle is black or NULL then make suitable rotations and recolor [p-nn and g-nn]
[Null, also considered black]

7] → If colored red then perform recolor [p→nn and g→nn]

8, 18, 5, 15, 17, 25, 40, 80

Case -1 :

Parent & uncle are red.

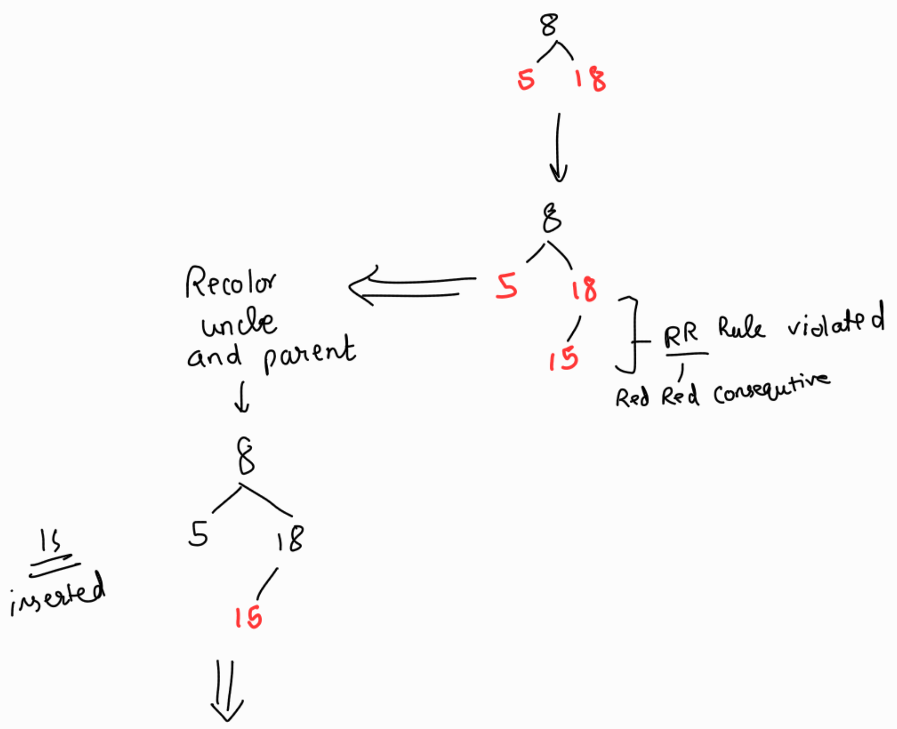
① Recolor parent & uncle black

② Move nn to grandparent.

i) if current nn is root - color it black

ii) else color it red.

③ Repeat ① and ② till you reach root.

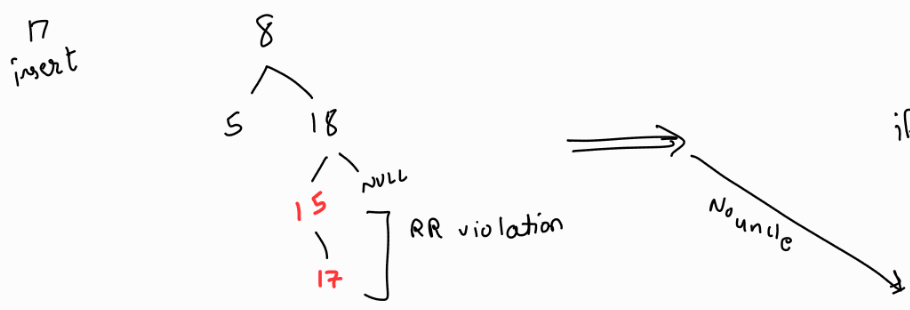


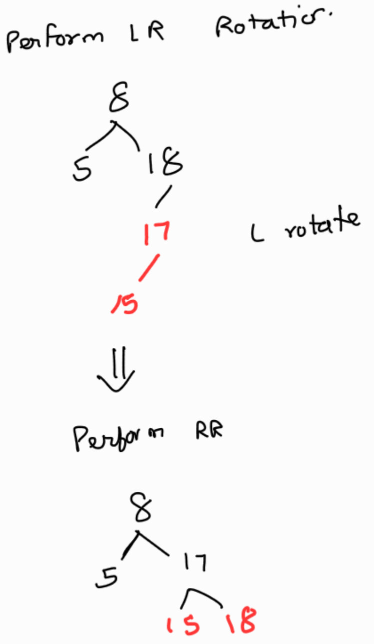
Case 2

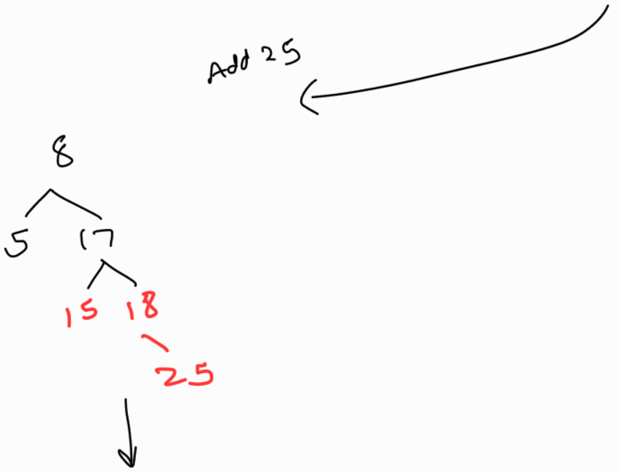
Uncle is black or NULL

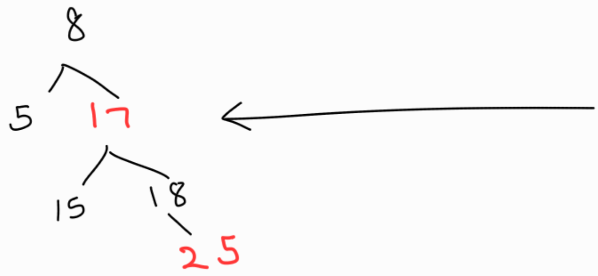
if NN is right child 'case 2'

else is left child 'case 3'



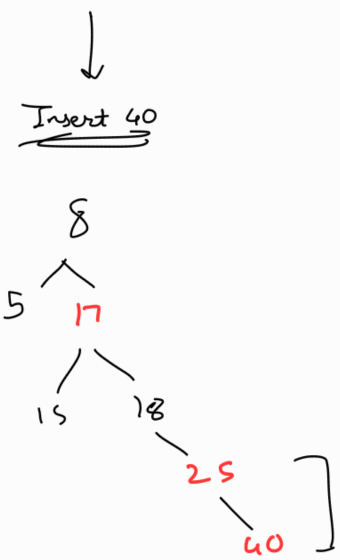






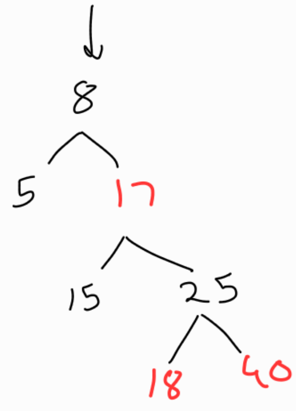
Case 1 : if nn→grandparent(n) is root → color=black else color=red.

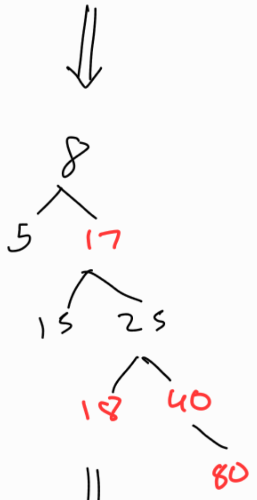
Insert 40

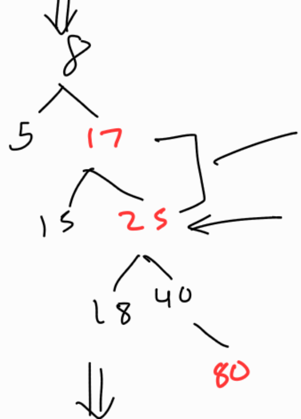


RR Rule vioted.

No uncle ∴ RR rotate.

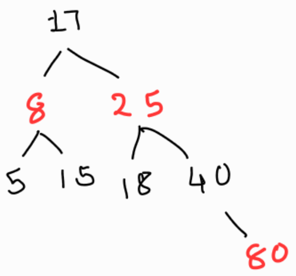






NN's parent not root ∴ color red.

∴ RR occured at 17 and 25



Delete :

- Delete as in BST
- If red deleted, no rebalancing needed.
- If black node deleted, a subtree beomes black deficiant.

NULL counted as BLACK.

① Delete a black leaf :

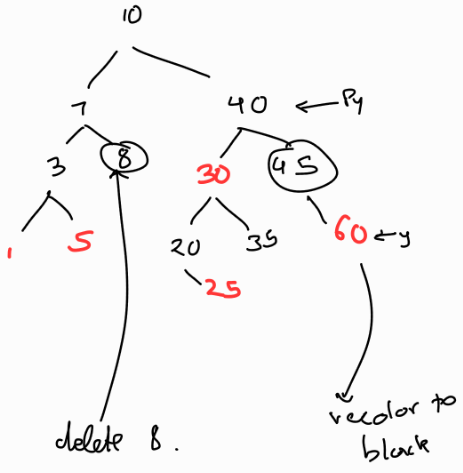
② Node is black with degree 1 i.e. 1 child.

③ Delete a black node with degree 2.

delete 45

↓

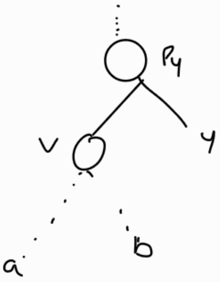
if y is red make it black

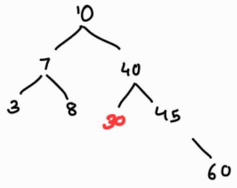


④ Y is black root

⑤ Y is black & not root (there is PY)

- Xcn
  - → Y is right child of PY ⇒ X = R
    else if Y is left child of Py ⇒ X = L
  - → C is the color of sibling of Y.
    Pointer to V is black ⇒ C = b
  - → V has 1 red child then, n = 1





Cases :

① RbO : → y is right of Parent, Sibling of Y is black, Sibling has 0 red child.
(Py is black)

- change color of y's sibling to red.

Now, Py is root of deficient subtree
Y = PY, repeat coloring till root.

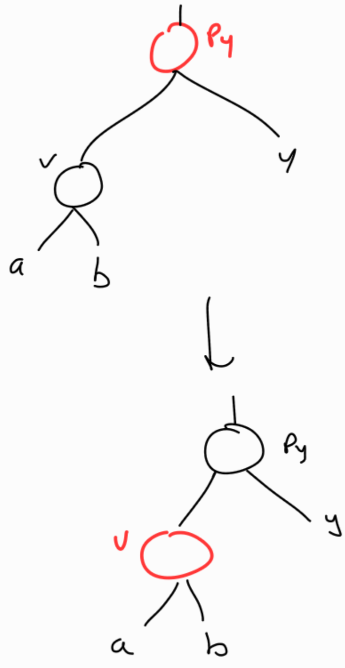
{ Made whole tree 1 black deficiant }

② Case 2 : Py is red
Rb0

Make Py black, Make y's sibling i.e. v red.

No need to go to root or lower as we are adjusting there only.

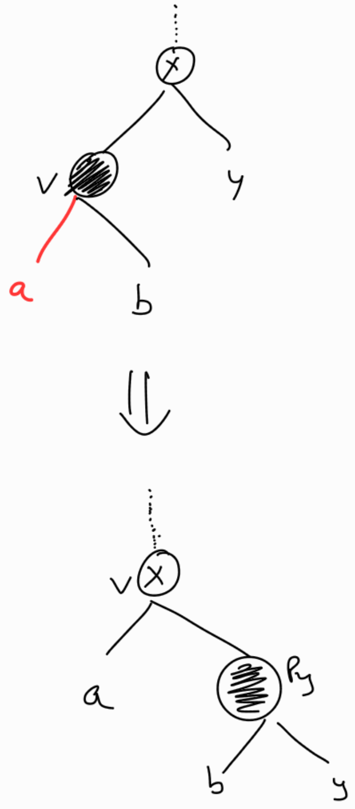
→ Color change, Deficiency, eliminated



③ Rb1 (case 1)

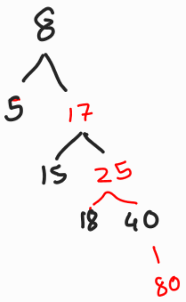
LL Rotation:
Swap colors of Left (v) and Parent (Py)

X - don't care color

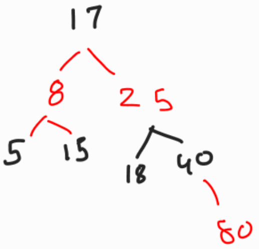


8, 18, 5, 15, 17, 25, 40, 80

root → color black



← nn not root → color red.



XD