Red Black Trees

Defⁿ It is a BST with colours Red & Black. It is balanced
height = $O(\lg n)$ worst case $O(\lg n)$

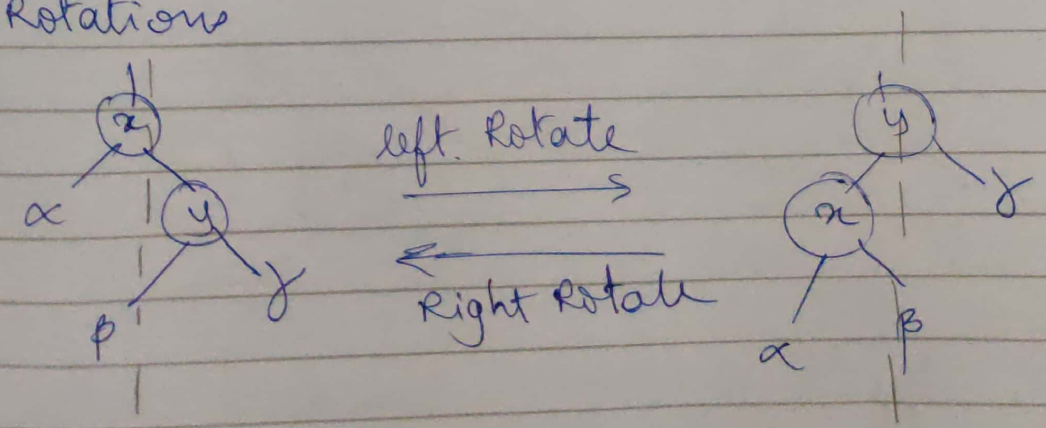
Properties

① It is a ~~soft~~ self balancing BST
② every node is red or black
③ every leaf is black
* ④ if a node is red, children are black { no 2 consecutive reds
⑤ root is black
* ⑥ For each node all paths from the node to descendent leaves contains same no. of black nodes

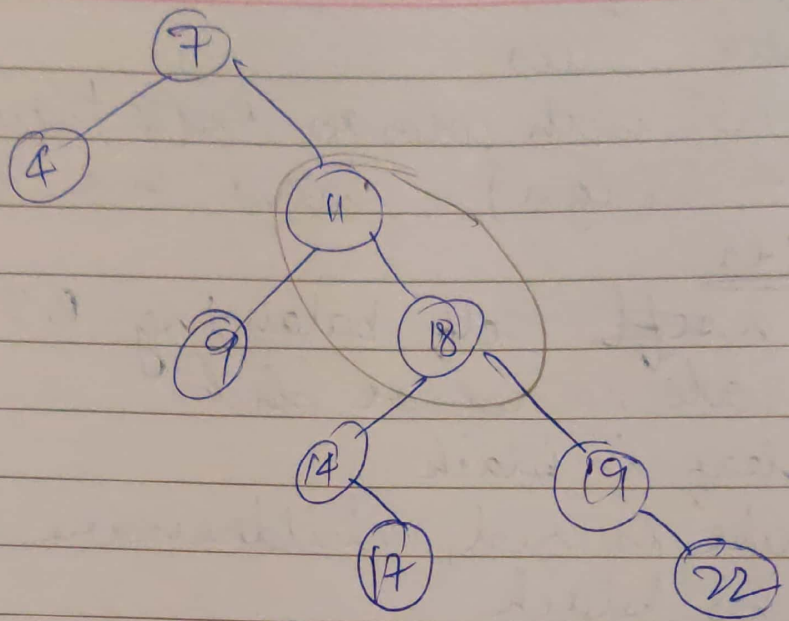
for any node at height h, black-height $\geq h/2$
⇒ for n internal nodes, height $\leq 2\lg(n+1)$

Deletion →
① if red node its okay, black height not affected
② if black node { black height affected,
* { what if new node put in place is red & we violated rule ④, ⑤, ⑥

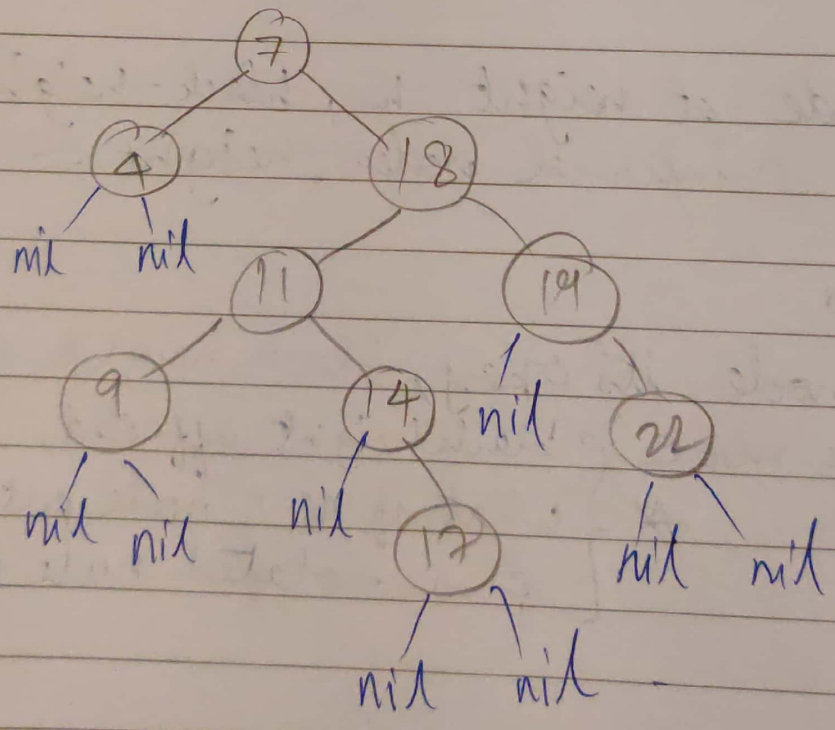
Rotations

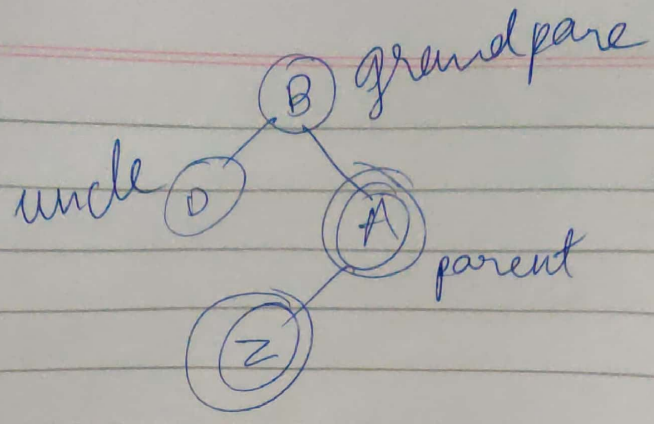


takes constant time $O(1)$ since there is only pointer manipulation



left Rotate(11, 18)



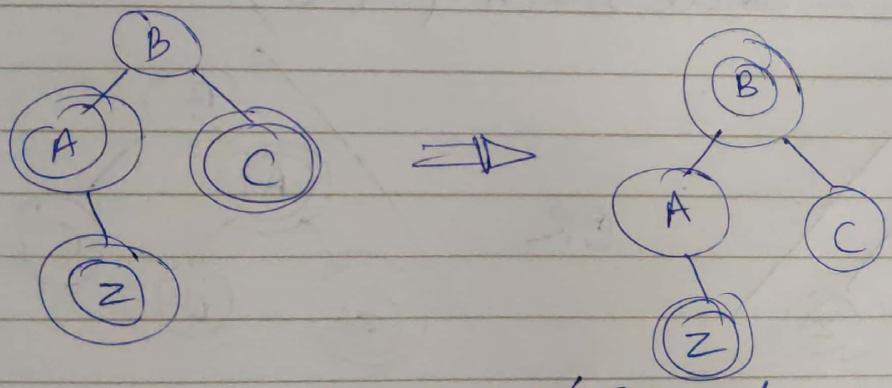


= Black
= Red.

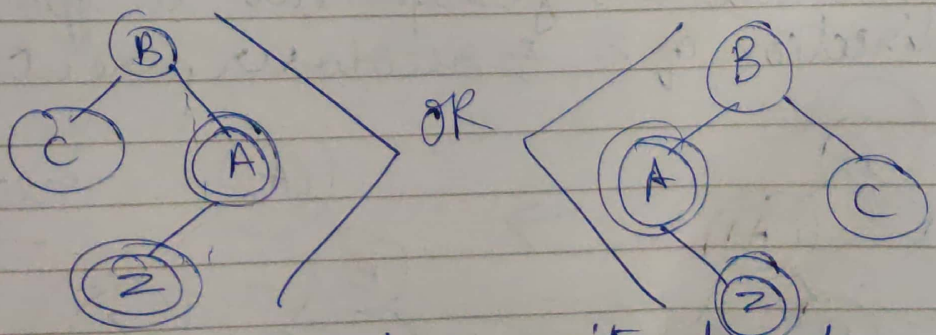
① Insert z, color red
② Rotate & recolour

Case → ○ z is root
recolour to Black

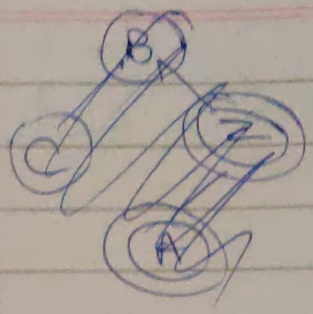
Case 1 z has red uncle
re colour parent, grandparent & uncle.

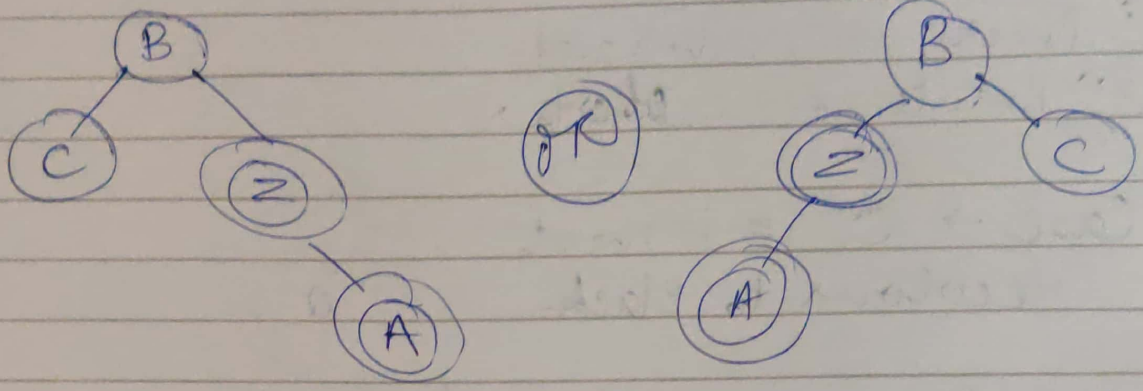


Case 2 uncle Black (Triangle formed)

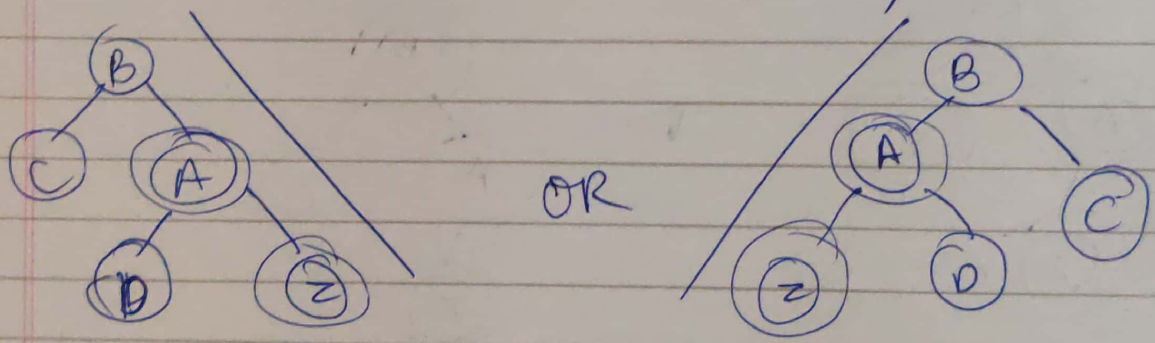


rotate z parent in opposite direction of z
if z on left do right rotation
z on right do left rotation.

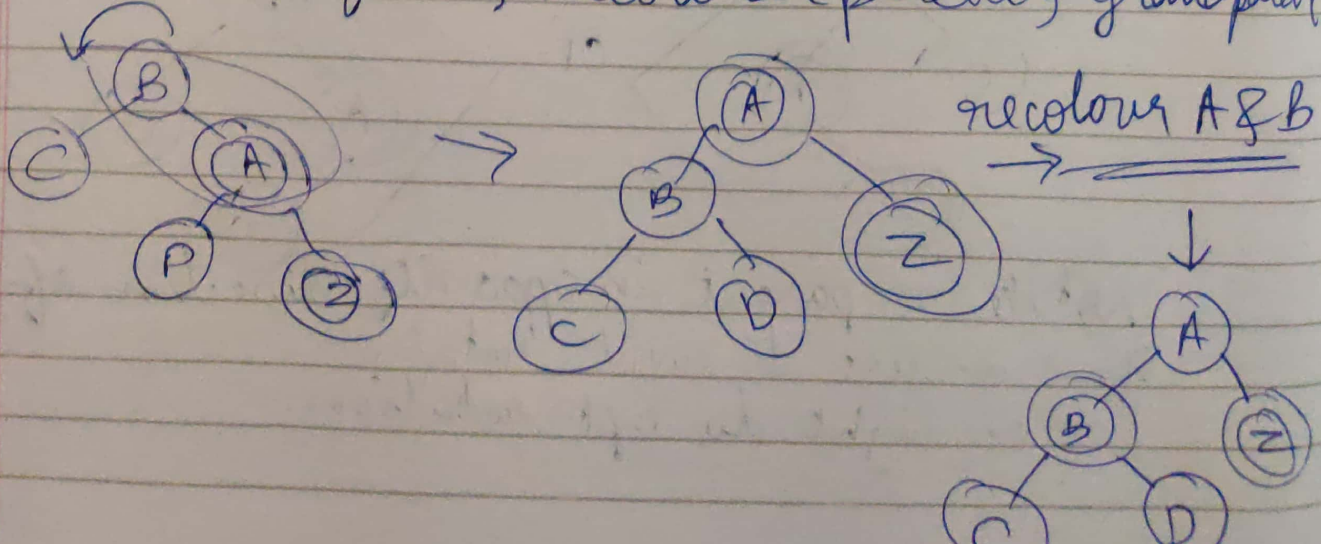




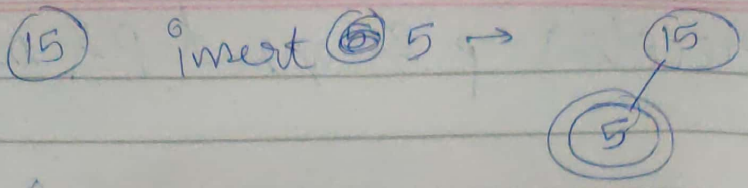
Case 3 z uncle Black (Line)

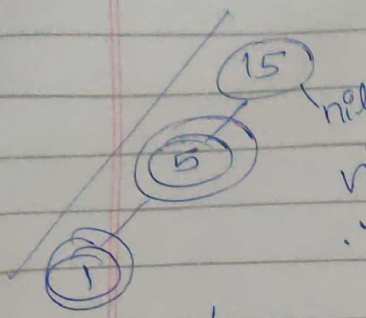


we rotate z's grandparent in opposite direction of z & recolour parent & grandparent

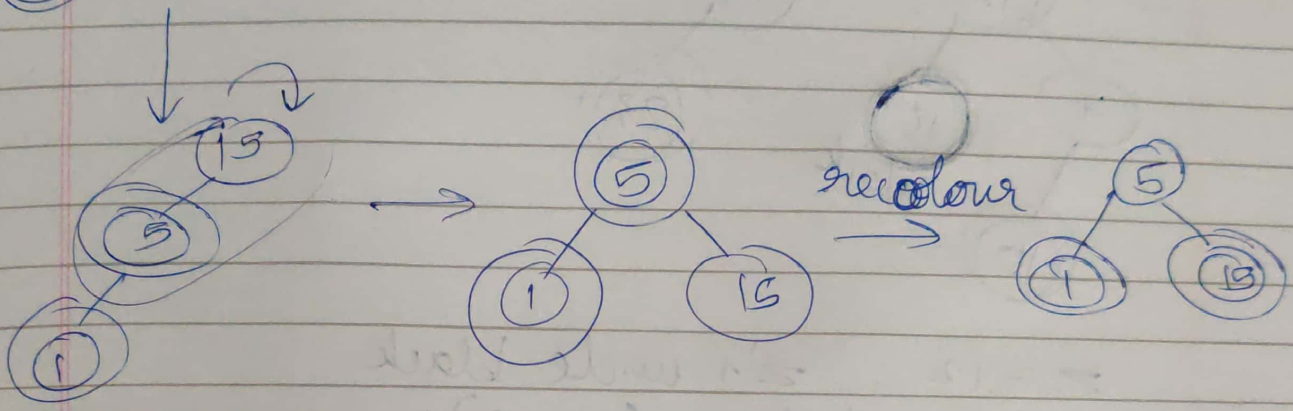


example 1

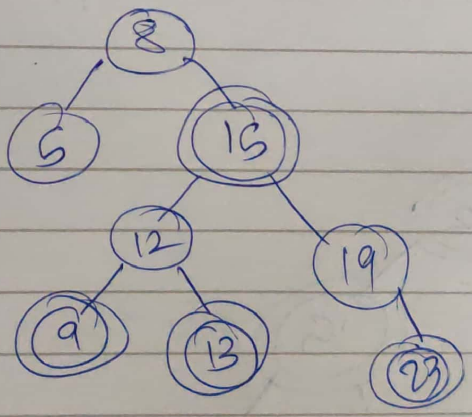




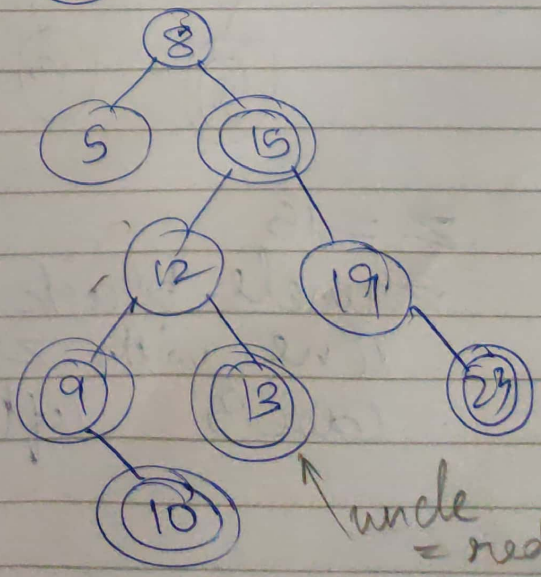
violation, uncle is nil (black) & LR formed
∴ Case 3



example 2

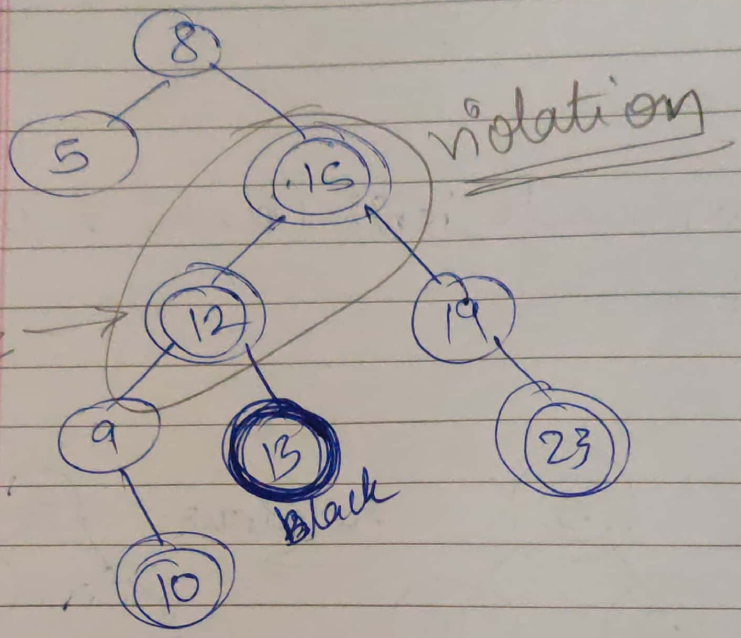


(insert 10 acc to BST Rules) we get

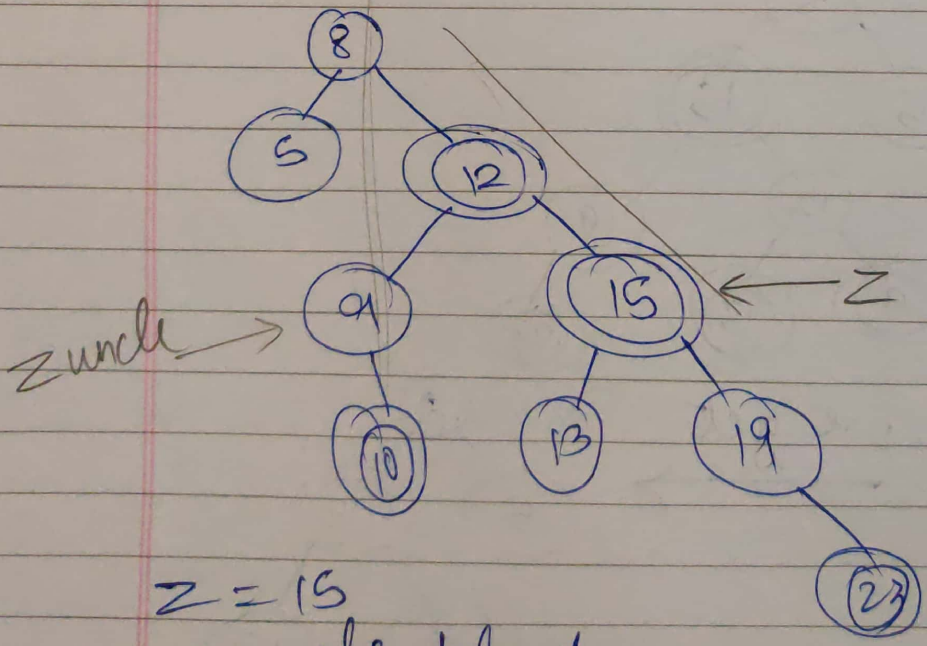


⇒ Case (1)

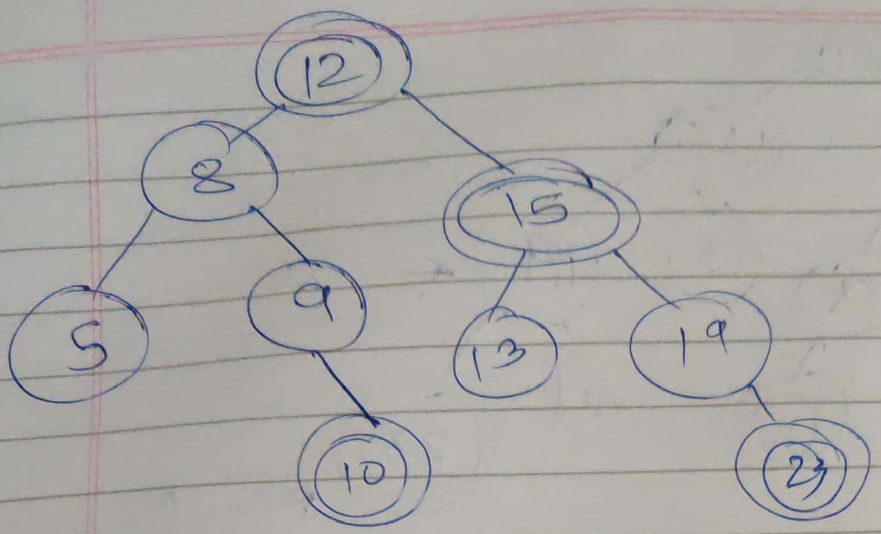
Case (1) recolor p & grand p & uncle



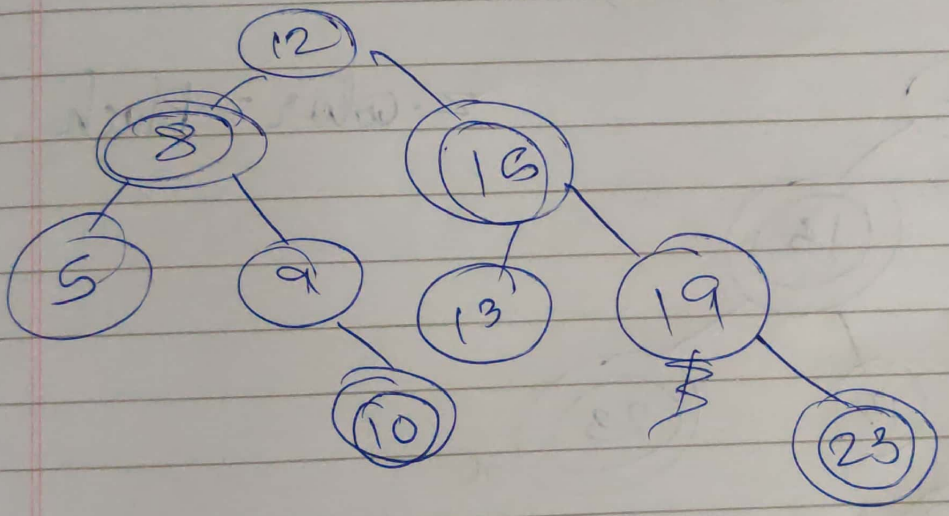
z = 12 z's uncle black
right rotation (z.p)



z = 15
z uncle black
line with z on right
case 3 left rotate (z.p & z.grand p)

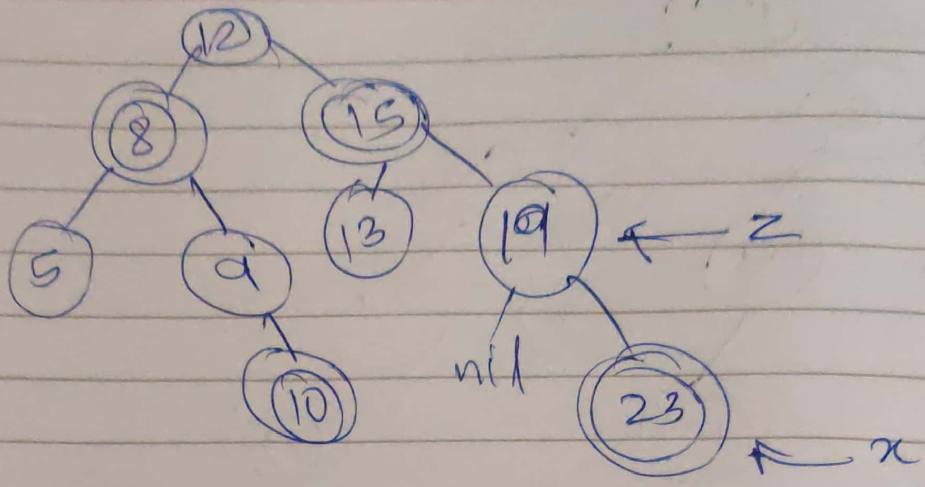


recolour z's p & grandp.



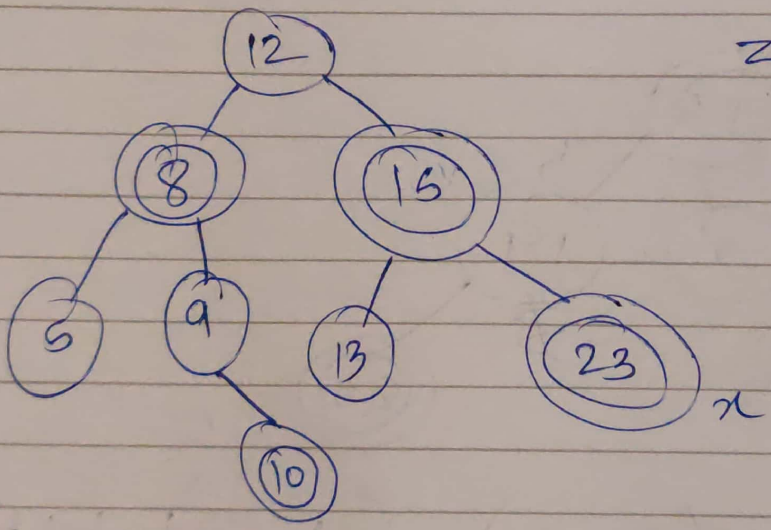
Insertion → O(log n) maximum height
recolour O(1)
Fix violations O(log n)

Total = O(log n)



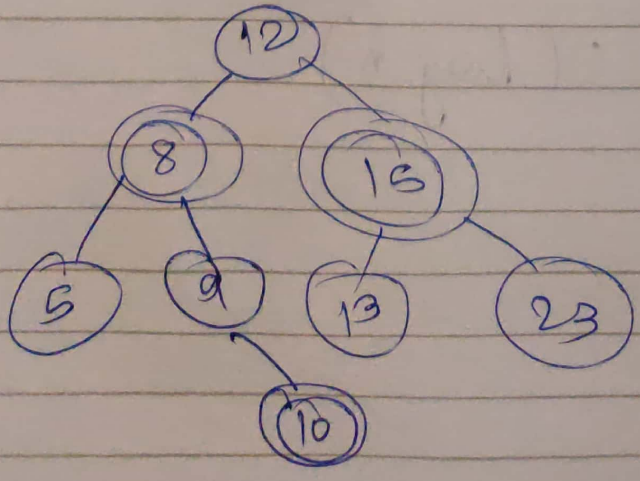
delete(19)

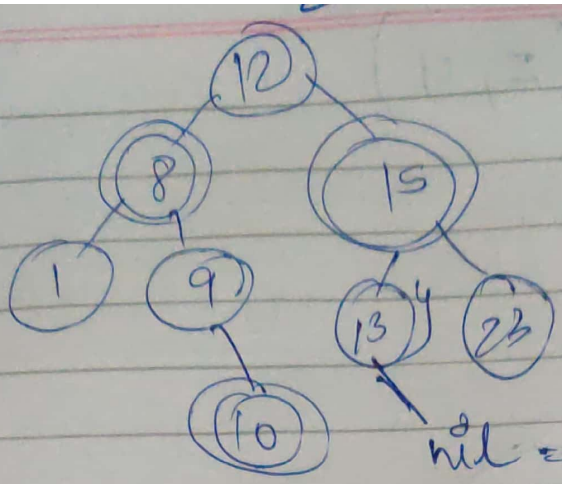
① Transplant(z, x)



z.color = Black.

delete-fixup(x)
→ recolors to black





delete(12)

z = 12 has 2 children (case ③ of deletion)

① find successor

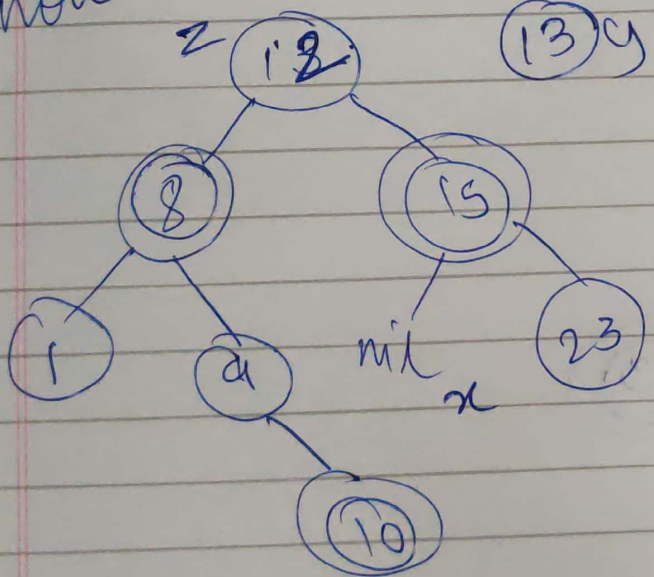
↳ find min in right subtree of 12

= 13

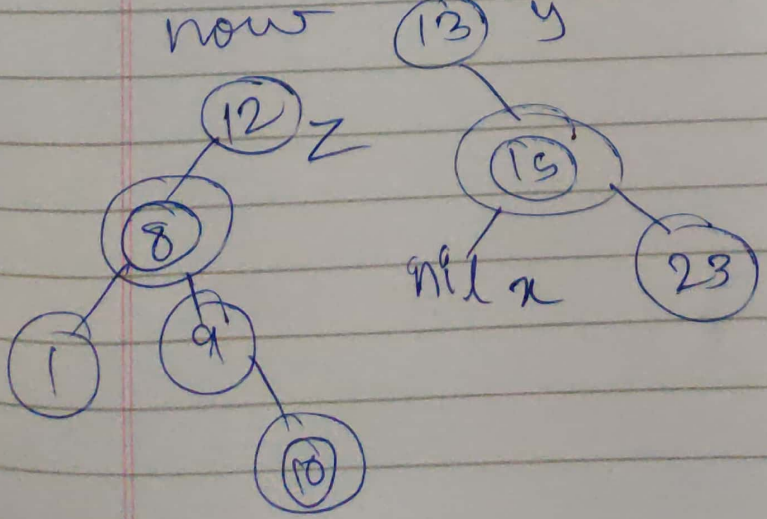
let y = 13

∴ Transplant (y, x)

now



now

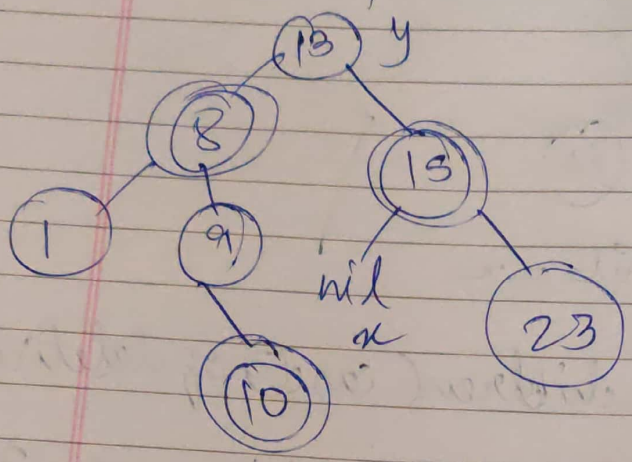


original colour was black. This means black height affected

⇒ call fixup(x)

~~transplant~~

transplant (z, y)



(13) — (15) — nil

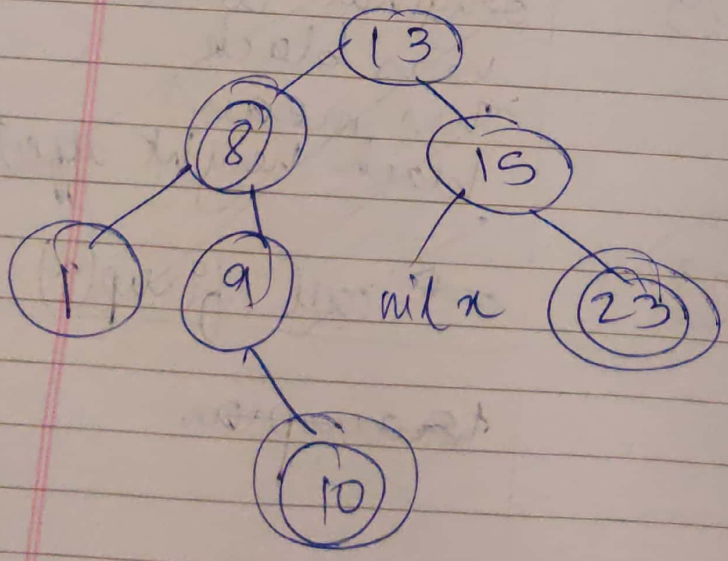
black height = 0

(15) — (23) — nil

black height = 1

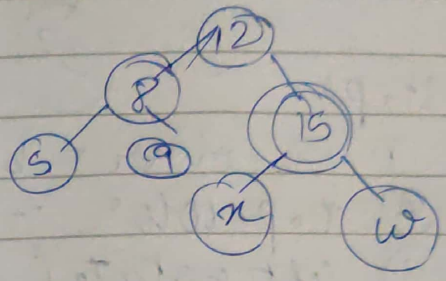
* since z's original colour was black deletion of z affected black-height

∴ call deletion-fixup(x) to recolour



delete_fixup(x)

1. x's sibling w is red
2. w black, w children black
3. w



delete_fixup(x)
w is x's sibling
1. ~~w is red~~

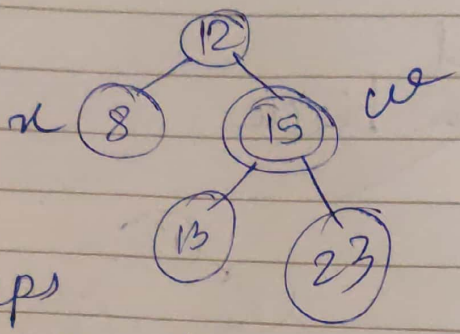
Types of fixes

1. w is red
2. w is black w children black
3. w black, w.left red w.right black
4. w black and w.right is red

* at the end of fix make x = black

All examples are subtrees

Type ① w is red



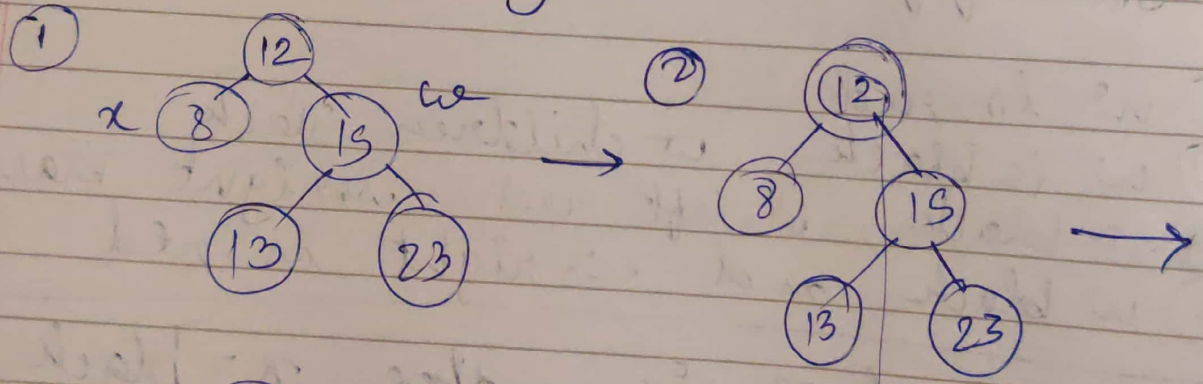
steps

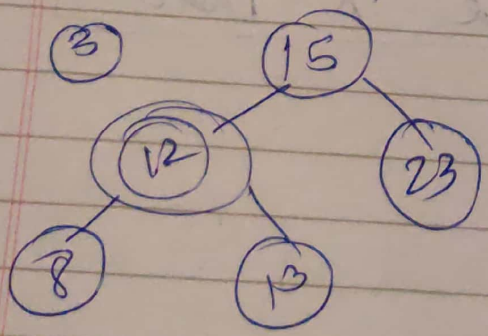
① w.colour = black

② x.p.color = red

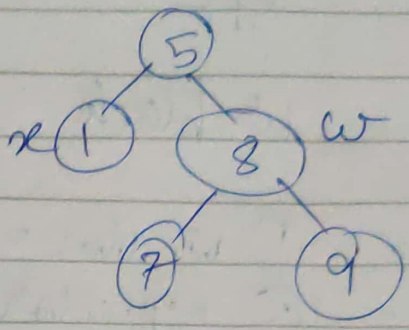
③ left rotate(x.p)

④ w = x.p.right

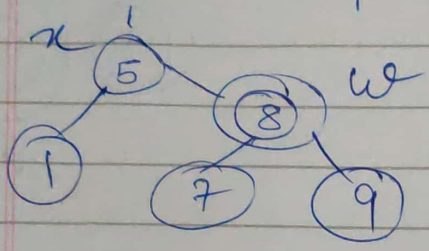




Type (2)
w is black
w children are black

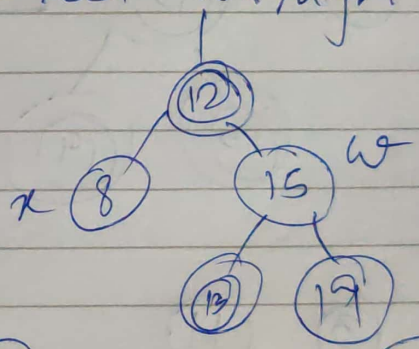


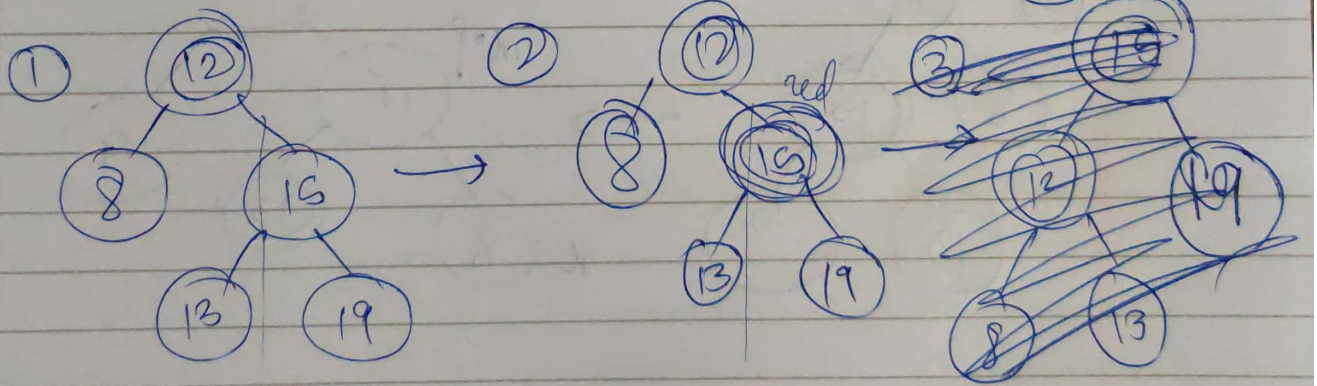
(1) w to red
(2) x = x.p.

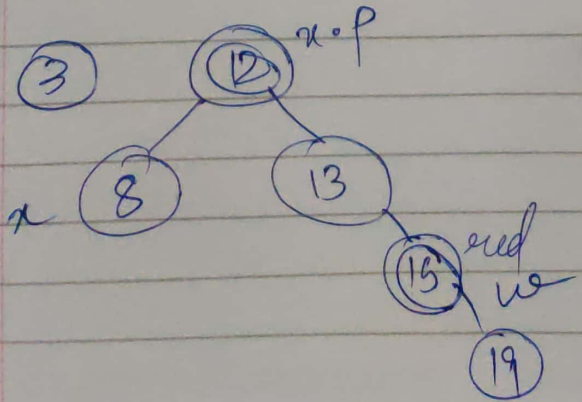


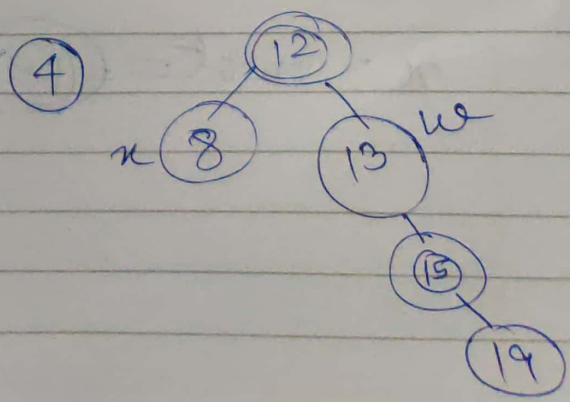
Type (3)
w is black w.left = red w.right = black

(1) w.left = black
(2) w.color = Red
right rotate w
w = x.p.right



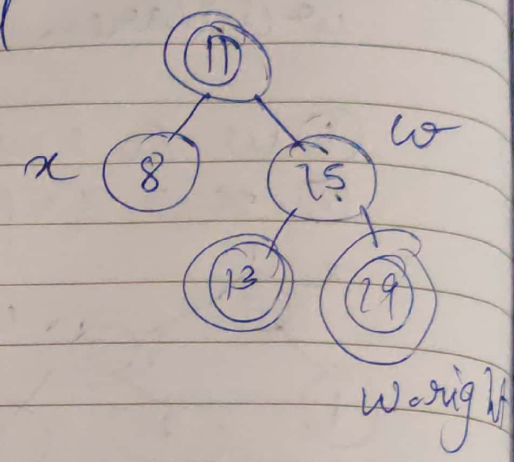




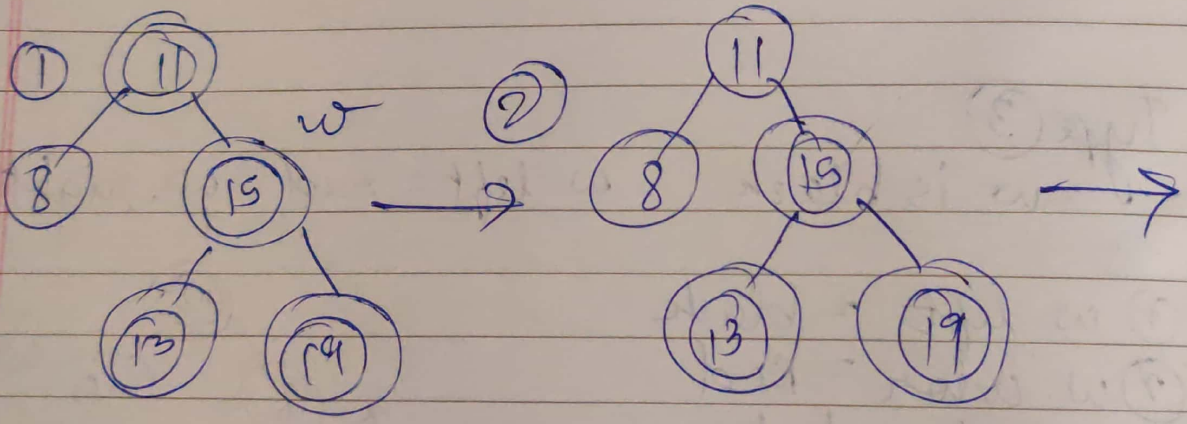


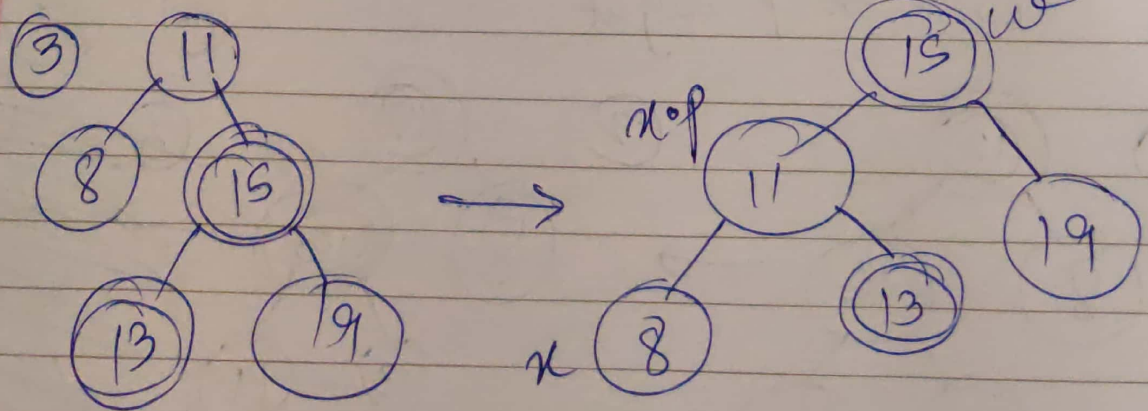
Type 4

w black w.right - red



① w.color = x.p.color
② x.p.color = Black
③ w.right.color = Black
left-rotate (x.p)
x = root





x = root = 15