* Red-Black Tree

A red black is a self balancing binary search tree where each node has an extra color bit Black / Red

Properties

→ Every node is either Red or black
→ The root is always Black.
→ All leaves are Black.
→ Red node cannot have a red parent / no two red nodes can be adjacent.
→ Every path from a node to its descendant leaves must have same no of black nodes

Step I :- Insert new node with color red.
Step II
Case 1 :- Node is Root recolor it to black.
Case 2 :- Red red violation
• If new node parent is red
Case 2.1 Uncle is red.
→ Recolor parent & uncle to black & grandparent to red. Repeat fix up process from grandparent

Case 2.2 Uncle is black / null.
→ Perform rotation to balance the tree
→ Recolor nodes accordingly

Eg 10, 20, 30, 15, 25, 5, 12, 35, 40, 32, 50

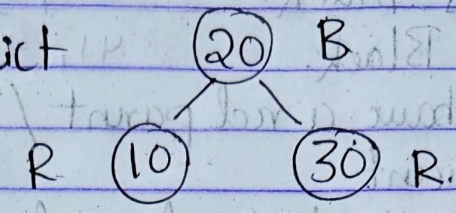
Three basic operations :- Recolor
Rotation.
Rotation followed by Recolor

1) 10 R ⇒ 10 B 2) 10 B — 20 R

3) 10 B — (B) 20 R — (R) 30 R.

Red-Red conflict
Uncle of 30 no uncle.
parent so rotation

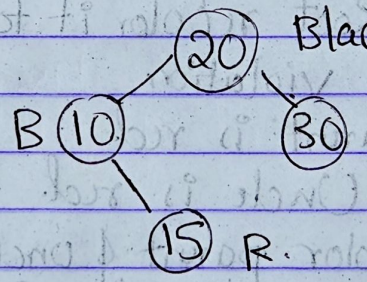
⇒ RR conflict



4) 20 B — R 10 — 30 R — R 15

Uncle of 15 is 30 & it is Red color
→ recolor parent, Uncle → black
grandparent - red

⇒



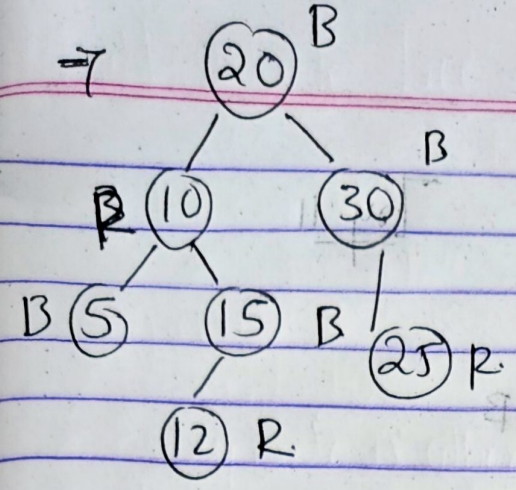
5) 20 B — B 10 — 30 B — R 15 — 25 R.

6) 20 B — B 10 — 30 B — R 5 — 15 R — 25 R

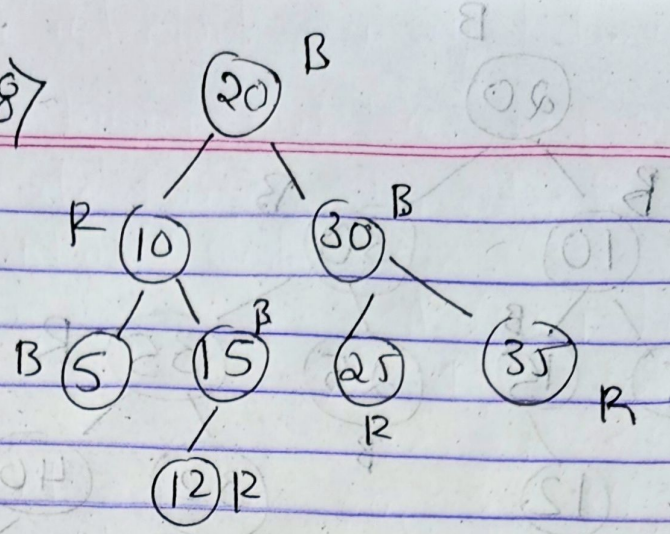
7) 20 B — B 10 — 30 B — R 5 — 15 R — 25 R — 12 R.

Red red conflict
Uncle is red.

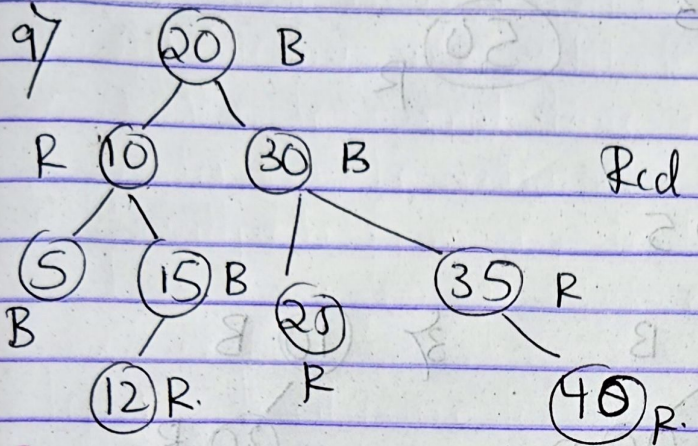
7)



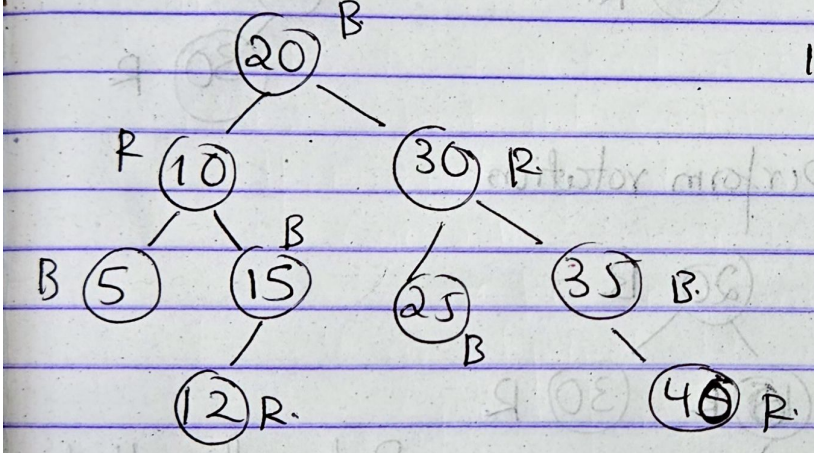
8)



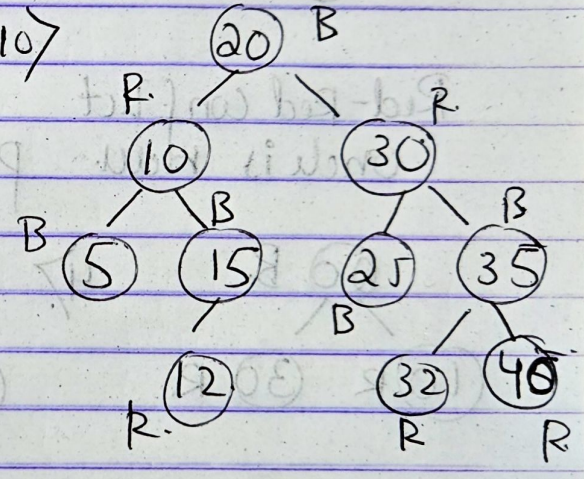
9)



Red Red conflict 25 is Red recolor parent, uncle, gra

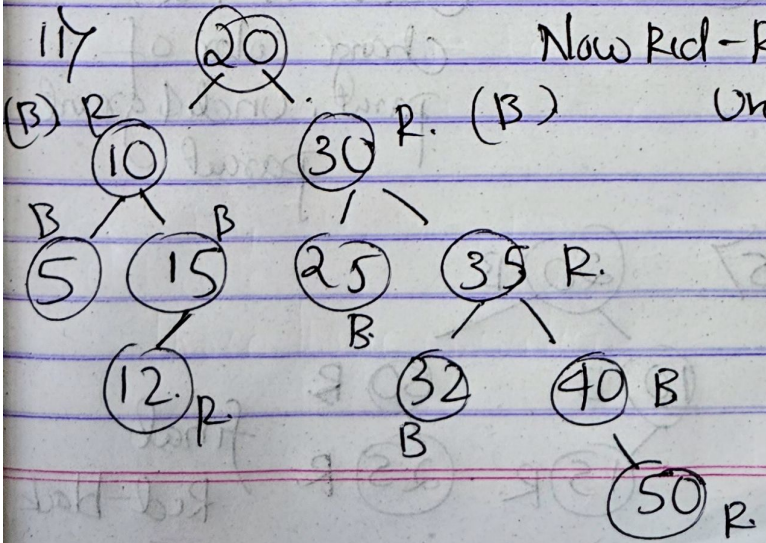


10)



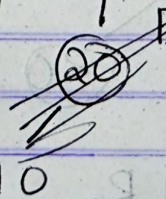
B (13)

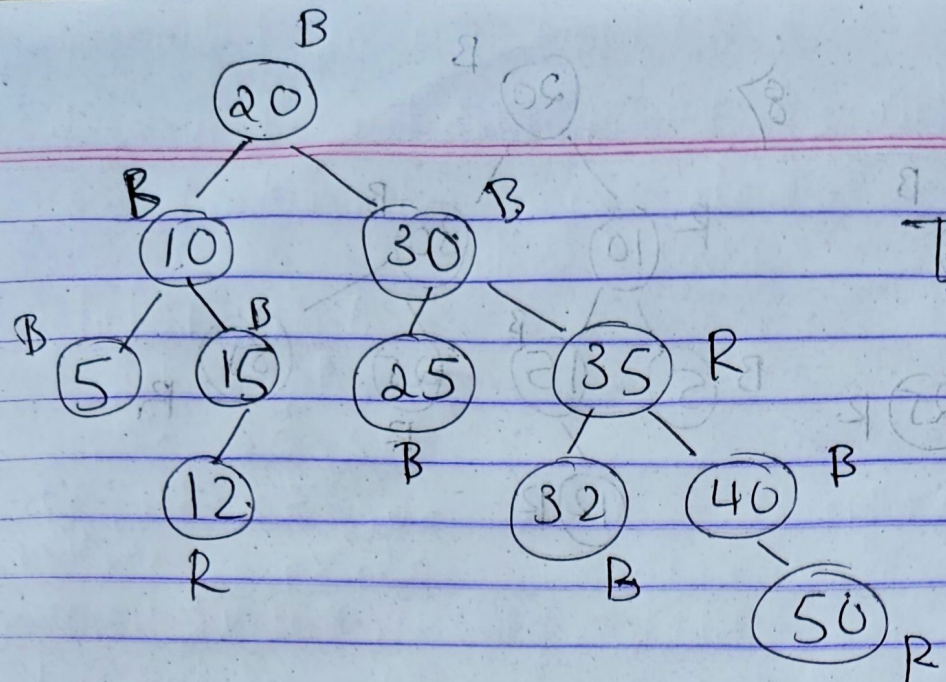
11)



Now Red-Red conflict. Change parent, uncle, grandparent

7





Try it

Eg 2

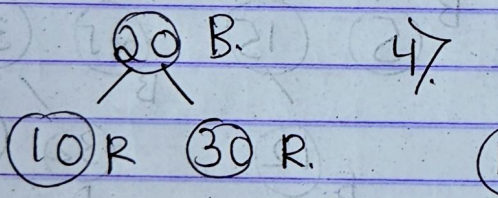
10, 20, 30, 15, 25

1) 10 R. ⇒ 10 B

2) 10 B — 20 R.

3) 10 B — 20 R — 30 R.

Red-Red conflict
uncle is Null. perform rotation



4) 20 B. — 10 R — 30 R. — 15 R

Red-red conflict
Uncle is red
change color of parent, uncle & grand parent

⇒ 20 B — 10 B — 30 B. — 15 R.

5) 20 B — 10 B — 30 B. — 15 R. — 25 R.

final Red-black Tree.