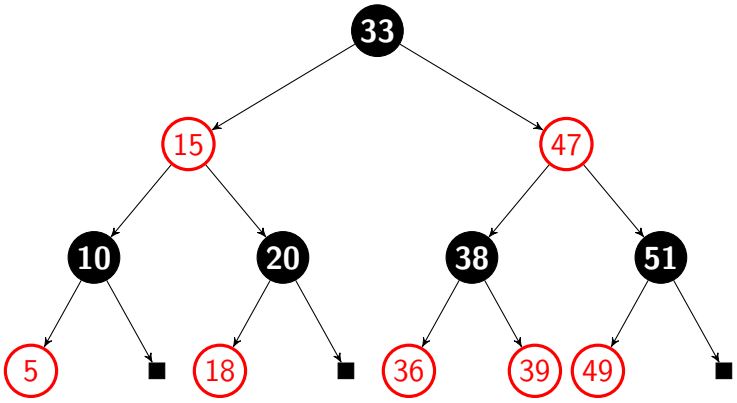
Contents

1 Red-Black Tree



1. Its a Height Balanced Binary Search Tree
2. Every Node is either Red or Black
3. Root of a Tree is Black
4. null is also Black
5. Number of Blacks on Path from root to leaf are same
6. No Two Consecutive Red, Parent and Children
7. Newly Inserted node is Red
8. Height is $\Rightarrow \quad \log(n) \leq h \leq 2\log(n)$

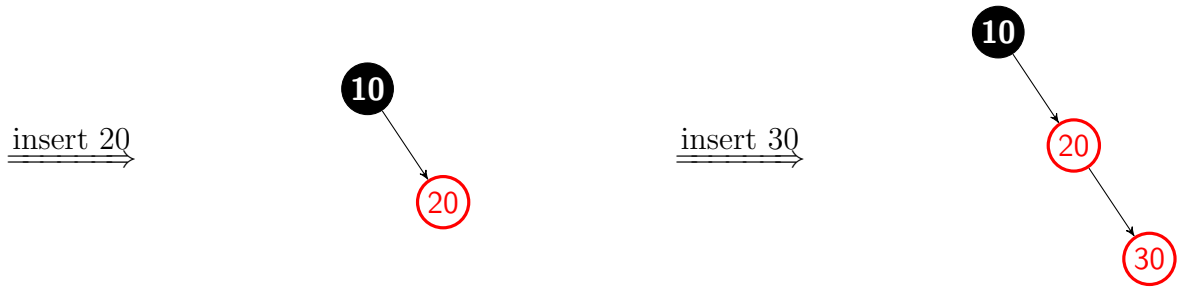
2 Red-Black Tree Creation

keys : 10, 20, 30, 50, 40, 60, 70, 80, 4, 8

Insertion in Red-Black Tree is just like Binary Search Tree

When Inserting New Node the New Node is Red Node

insert 10 ⟹ (10) ⟹ change root to black ⟹ **10**



red-red conflict

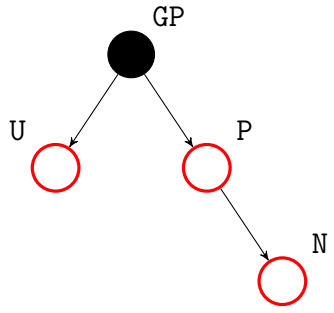
Whenever There is red-red conflict Then you have to do some adjustment for Making It As a Balanced Red-Black Tree .

There are Two Approach for Adjustments

$$\Rightarrow \begin{cases} 1.Re-Coloring \\ 2.Rotation \end{cases}$$

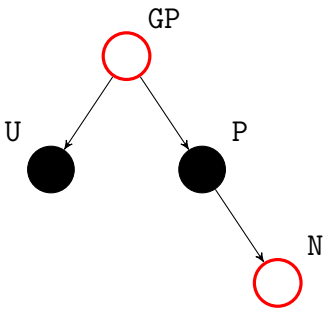
When We Have Red-Red conflict It Can Be Two Situation with the Uncle Of That Newly inserted Node

2.1 Uncle is Red



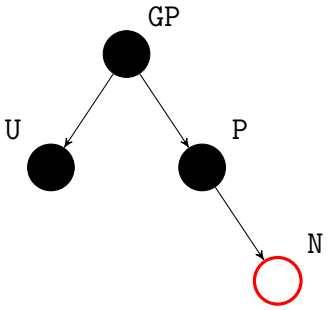
When Uncle is Red We Re-Color

Re-Color ⟹

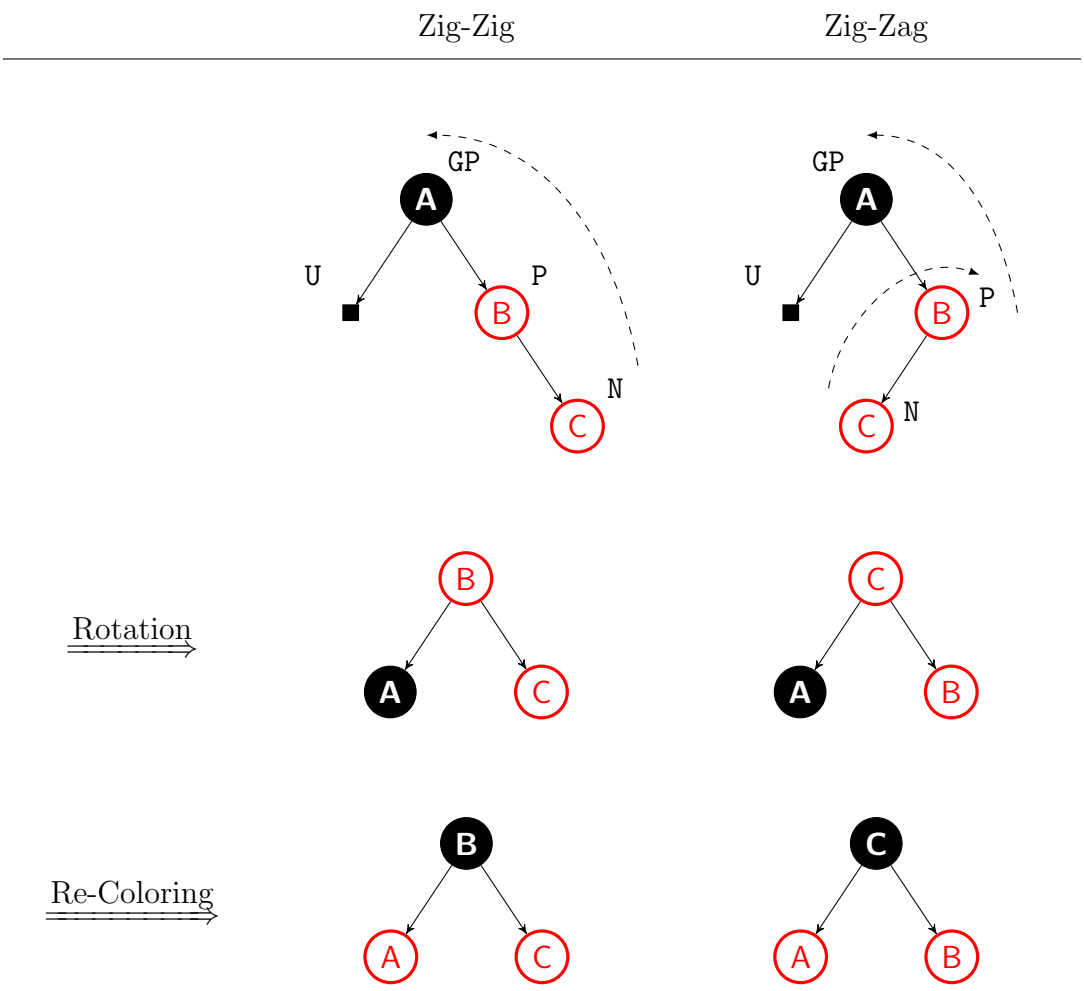



```
if ( GP == root ) {
   Re-Color GP to Black
}
```

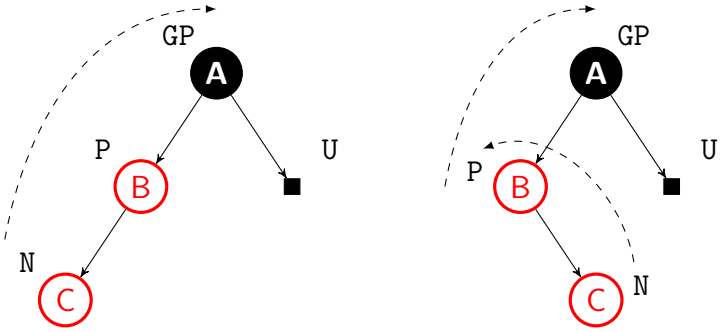
if (GP == root) ⟹



2.2 Uncle is Black



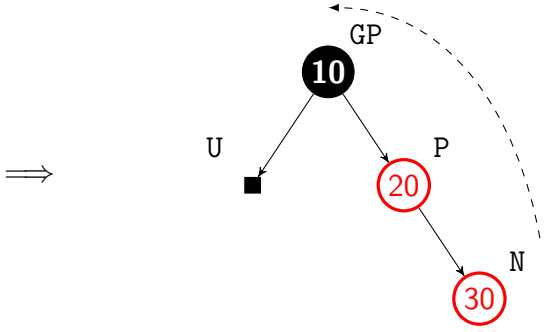
Zig-Zig Zig-Zag



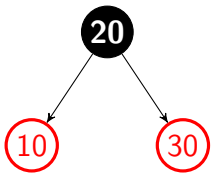
Rotation

Re-Coloring

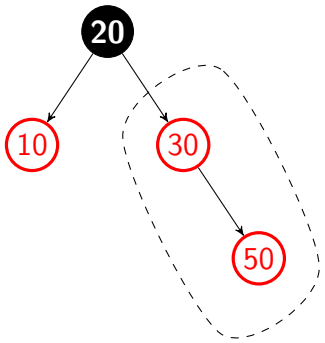
now we continue our Red-Black Tree Creation :



RR-Rotation

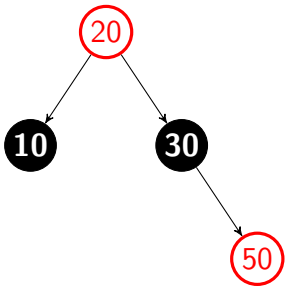


insert 50 ⟹



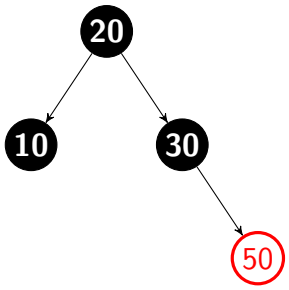
- Red-Red Conflict
- Uncle is red → Re-Coloring

Re-Coloring ⟹

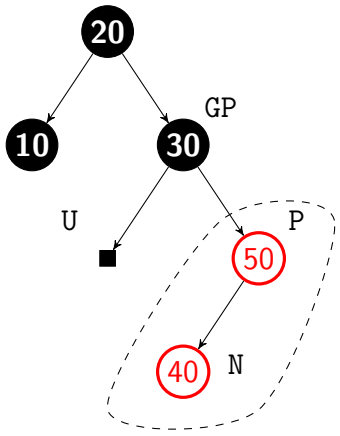



```
if ( GP == root ) {
    Re-Color GP to Black
}
```

Re-Color root to Black ⟹

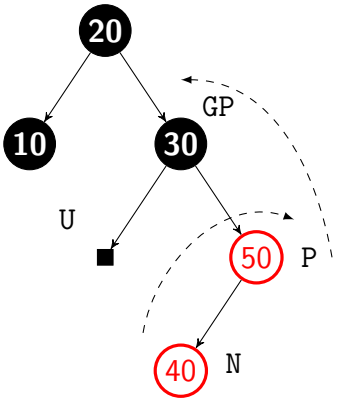


insert 40 ⟹

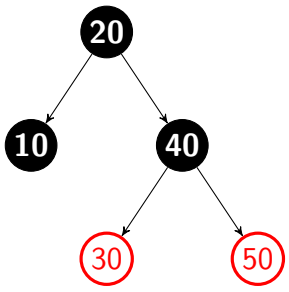


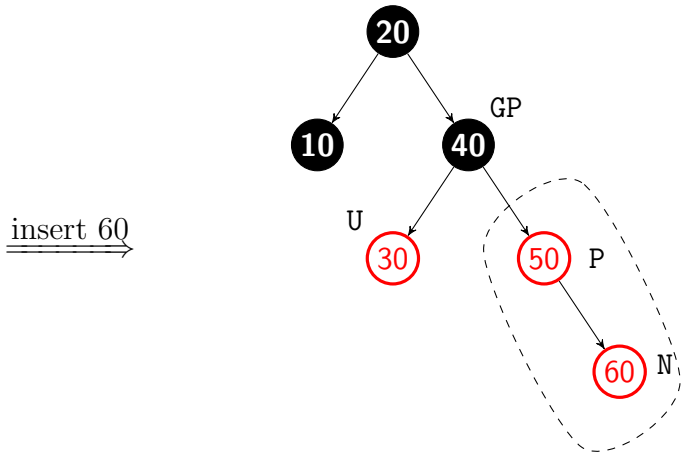
- Red-Red Conflict
- Uncle is Black → Perform Rotation

⟹

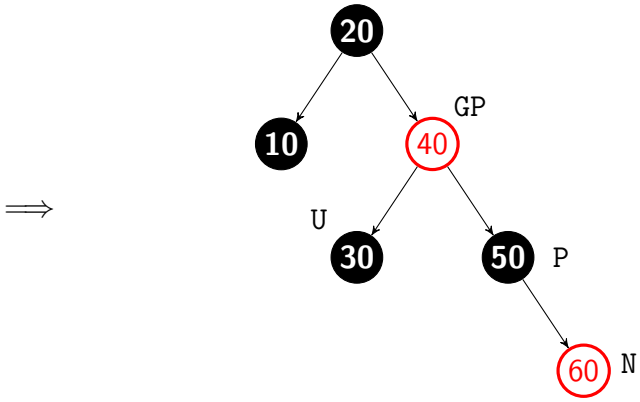


RL-Rotation ⟹



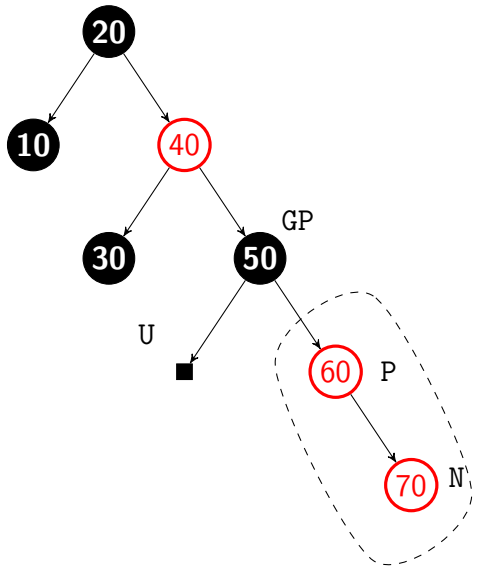


- Red-Red Conflict
- Uncle is red → Re-Coloring



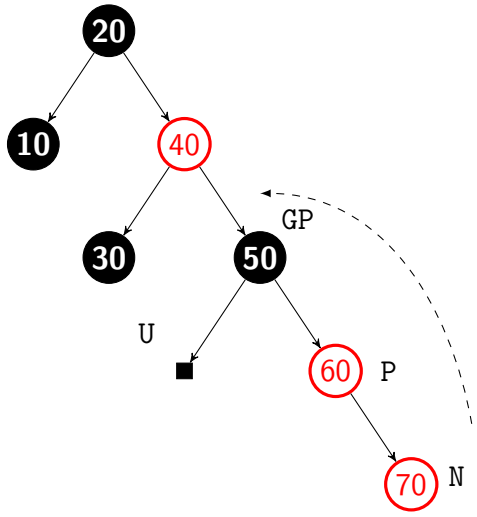
- Now Again for GP, check It's Ancestor to see there is Red-Red Conflict , You should check this untill There is no conflict
- in this case the Ancestor is black and there is no Red-Red conflict, so we don't need to do anything more

insert 70 ⟹

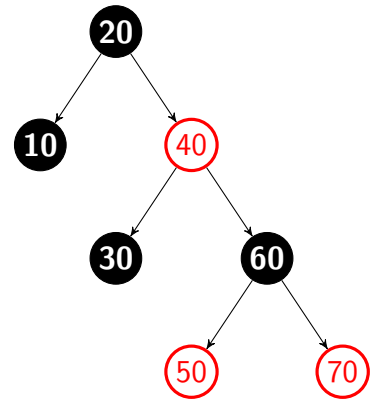


- Red-Red Conflict
- Uncle is black → Rotation

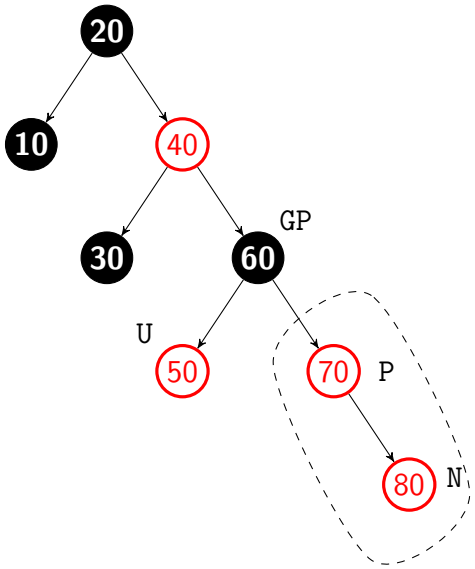
⟹



RR-Rotation

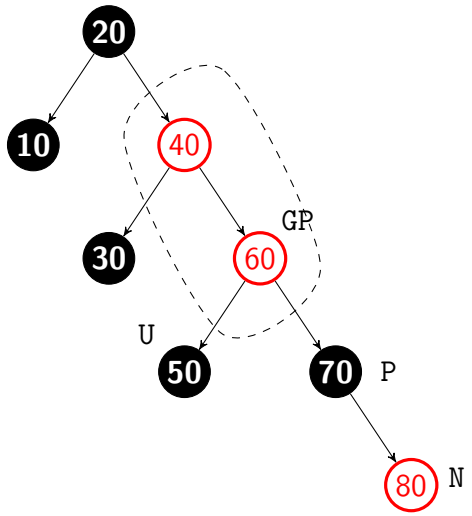


insert 80



- Red-Red Conflict
- Uncle is red → Re-Coloring

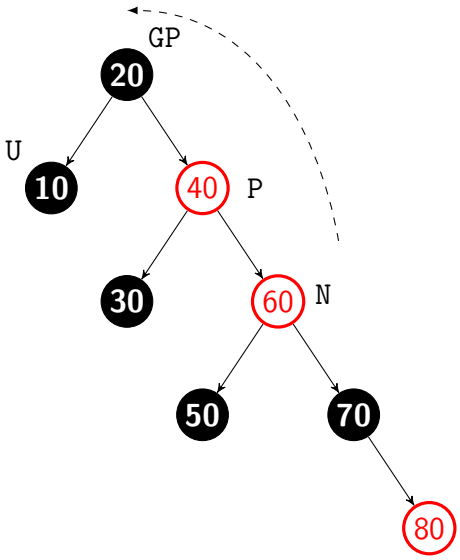
Re-Coloring



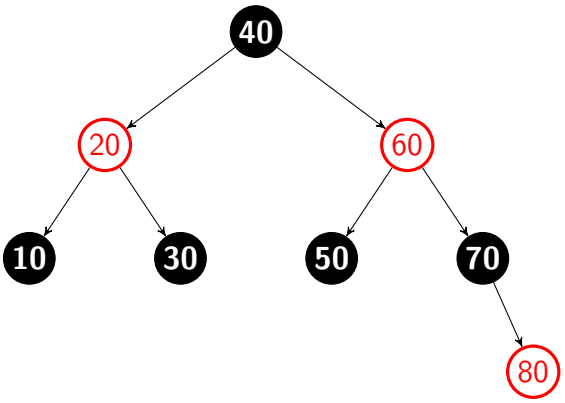
- Now Again for GP, check It's Ancestor to see there is Red-Red Conflict , You should check this untill There is no conflict
- in this case the Ancestor is red and there is Red-Red conflict

- Red-Red Conflict
- Uncle is Black → Rotation

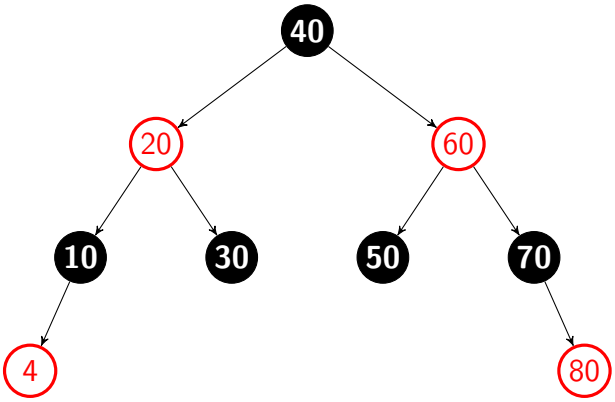
Re-Coloring



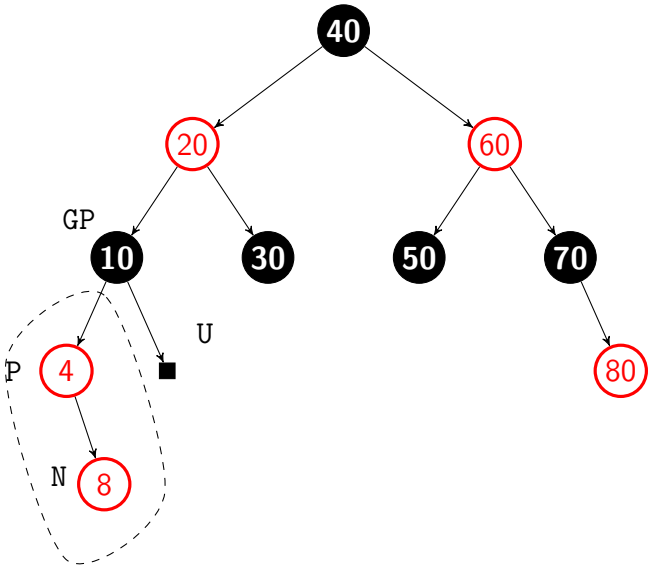
RR-Rotation



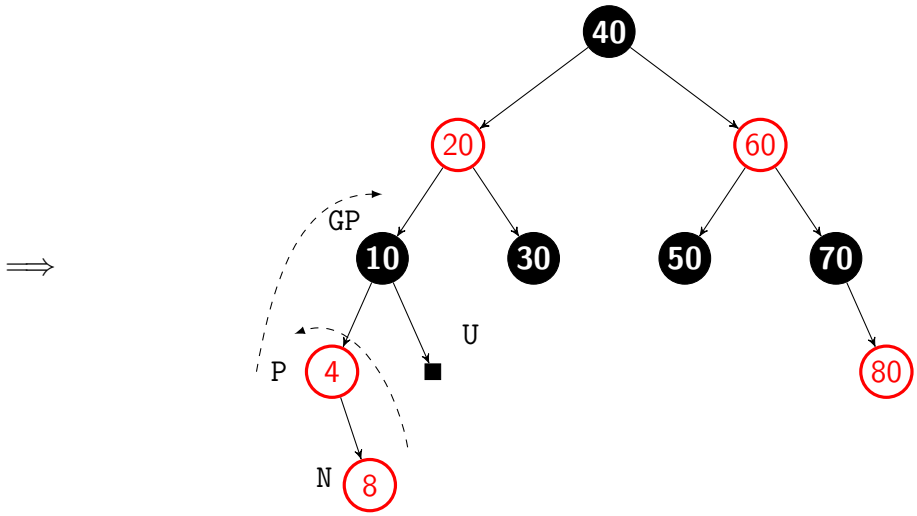
insert 4

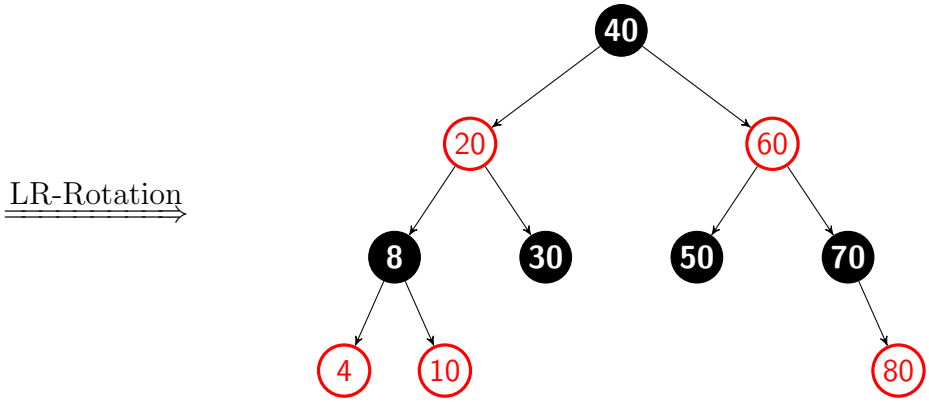


insert 8



- Red-Red Conflict
- Uncle is Black → Rotation





3 Red-Black Tree Deletion Cases

- Deletion from Red-Black Tree is like Deletion from Binary Search Tree And Involving Rotation Or Re-Coloring Of Nodes

3.1 How Deletion Happens in B.S.T?

Actually in B.S.T We Don't Delete The Whole Node, And We Delete Just the Value , and the Node is Kept as It is And Some Other Value Will Take It's Place
who will kept it's place ? either inorder precedesor or inorder succesecor

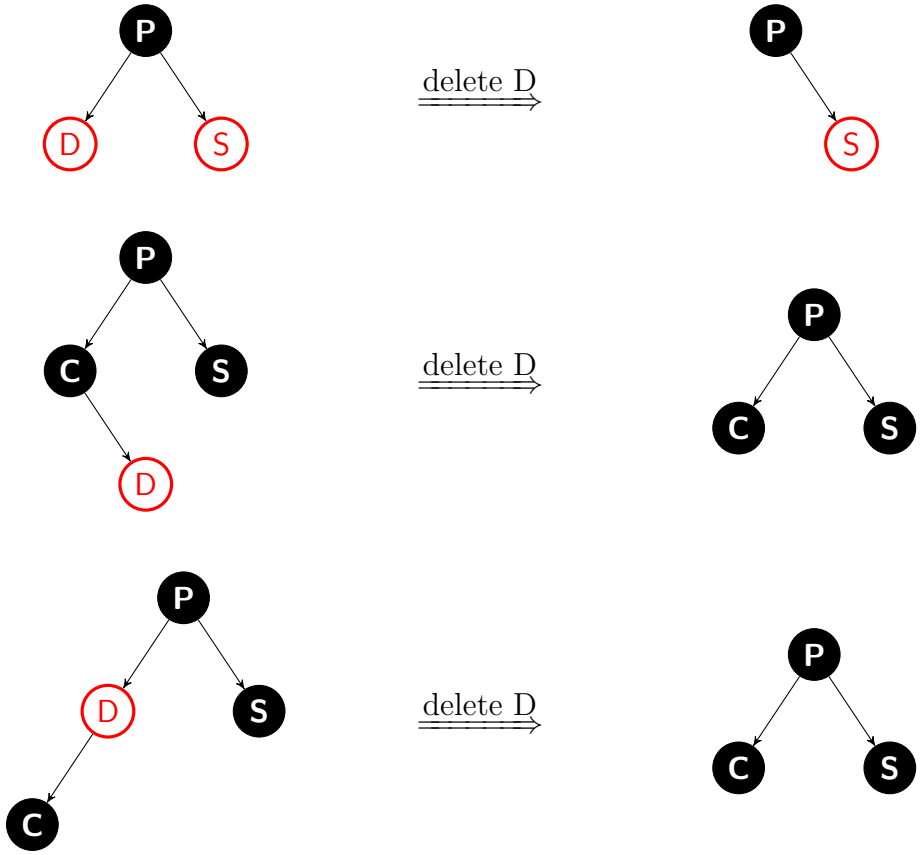
- Whenever you delete from B.S.T It May Be a leaf Node Or It will have exactly one child

- When you delete a Red Node from a Red-Black Tree Simply delete it and if it has a Child (Obvioudly Black) Replace The Child with It

- Why Deleting Red Node From Red-Black Tree is No Problem?

  Answer : Because in Red-Black Tree The Number of Black Nodes Along Any Path should be same So If You Delete Red Color Node It's Not effect Red-Black Tree At All

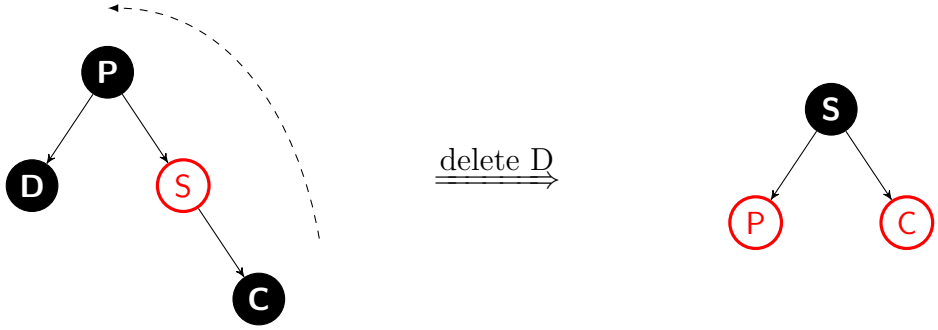
3.2 Case 1



- So Where is the hard Issue in Deletion Red-Black Tree ? Answer : If the Node Is Black

3.3 Case 2

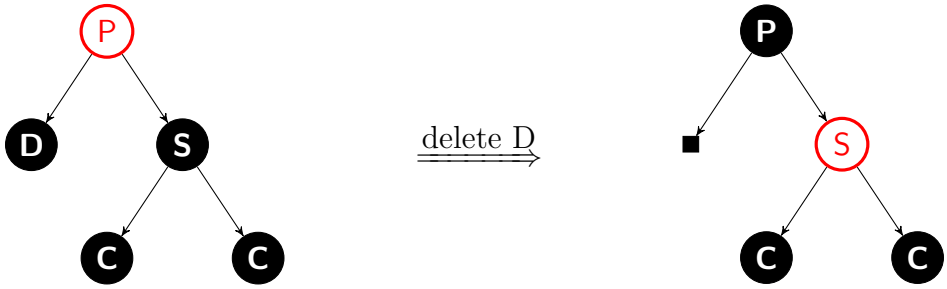
When the Node That you Deleting is Black Then check The Sibling, if Sibling is Red Then , Simply Delete The Node You want And Perform Rotation



3.4 Case 3

When the Node we Want To Delete is Black And The Sibling Is Black Too, Then We Have Multiple choices :

1. If Both The Sibling Children Are Black Then Simply Change The Color (Re-Coloring)
    - based on how many children Sibling has There are defferent cases



1. If Sibling Children Are Red then, Perform Rotation
    - based on how many children Sibling has There are defferent cases

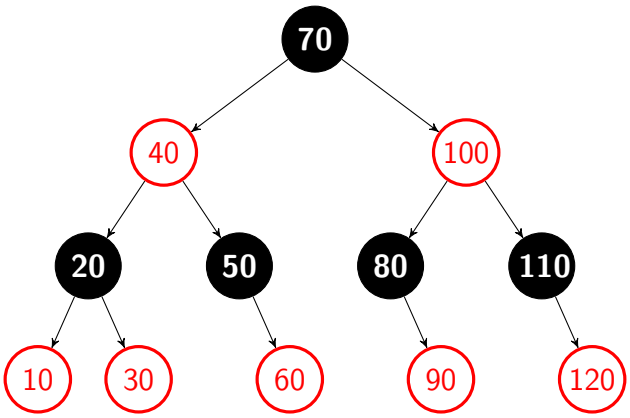
3.5 Summary - Deleting a Red-Black Tree Node

Sibling is Red $\Longrightarrow$ Rotate

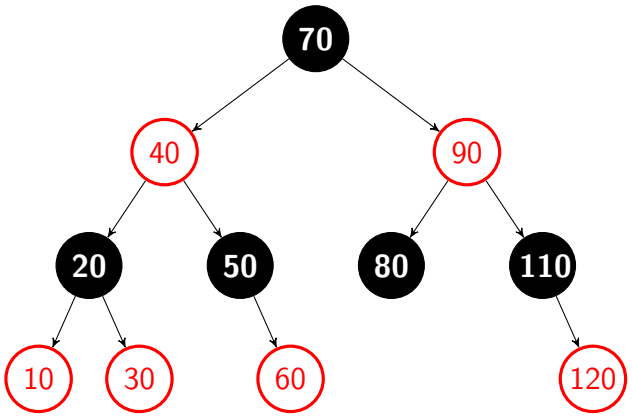
Sibling is Black, Children are Red $\Longrightarrow$ Rotate

Sibling is Black, Children are Black $\Longrightarrow$ Re-Color

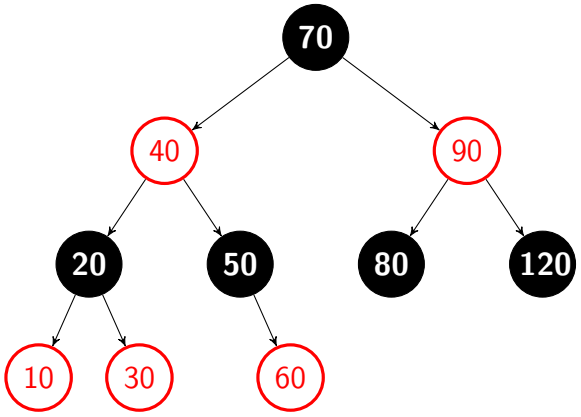
4 Red-Black Tree Deletion Examples



delete 100 ⟹

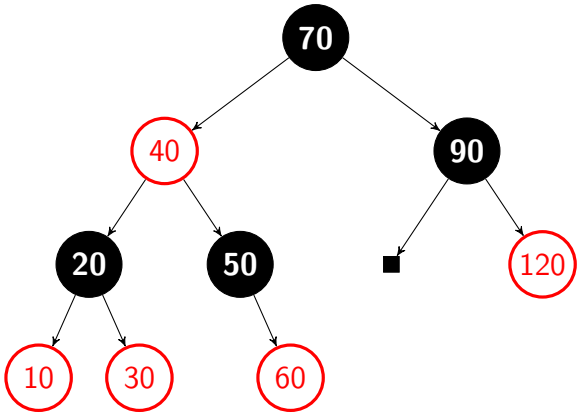


delete 110 ⟹

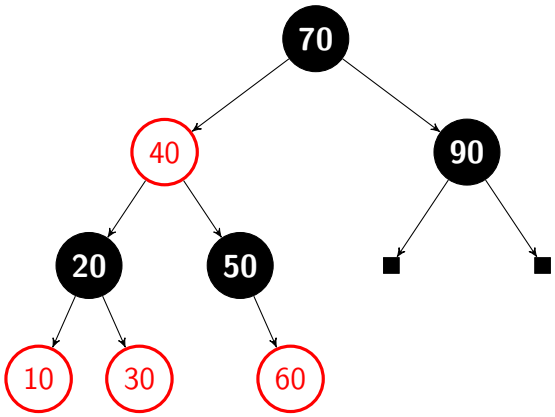


when deleting 80 the Sibling is Black And Their Children is Black So Re-Coloring

delete 80 ⟹

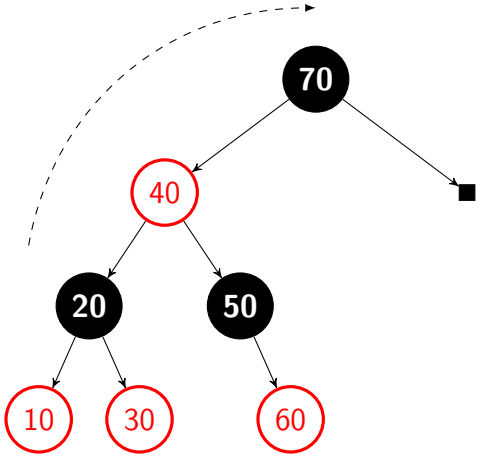


delete 120 ⟹

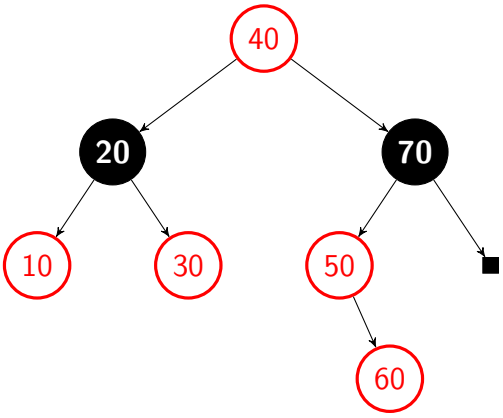


When Deleting 90 , Sibling is red and Their Children Are Black So We Rotate

delete 90 ⟹

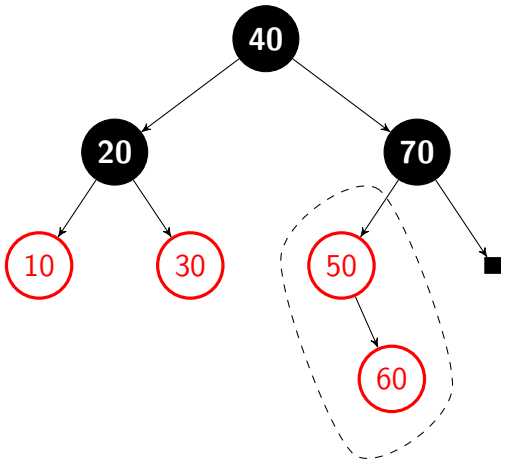


LL-Rotation ⟹




```
if ( GP == root ) {
   Re-Color GP to Black
}
```

$\Longrightarrow$



- Red-Red Conflict
- Uncle is Black so We Rotate

LL-Rotation $\Longrightarrow$