Red-Black Trees

Insertion and Deletion

Red-Black Trees: Properties

1) Every node has a color either red or black.

2) Root of tree is always black.

3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).

4) Every path from root to a NULL node has same number of black nodes.

Black Height of a Red-Black Tree :

Black height is number of black nodes on a path from a node to a leaf.

Insertion

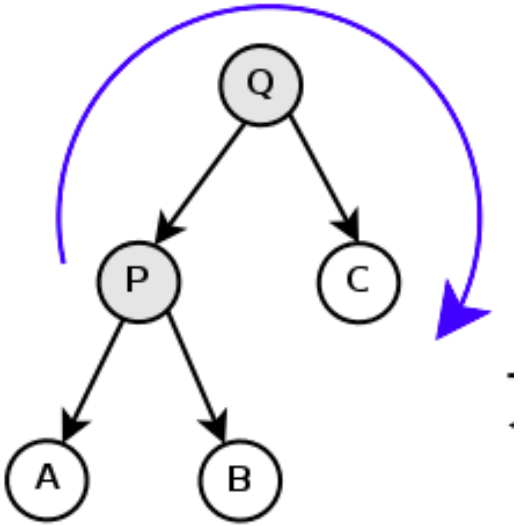
we use two tools to do balancing.

1) Recoloring

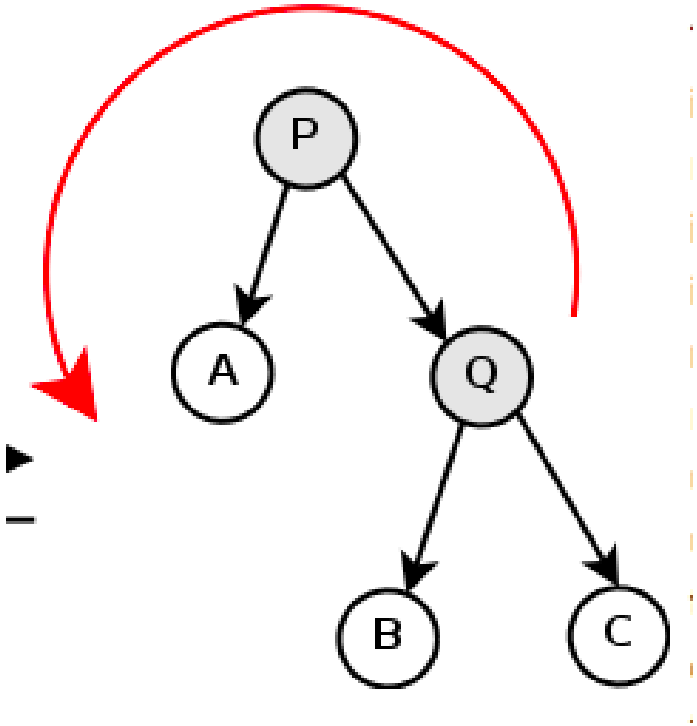
2) Rotation

Color of a NULL node is considered as BLACK.

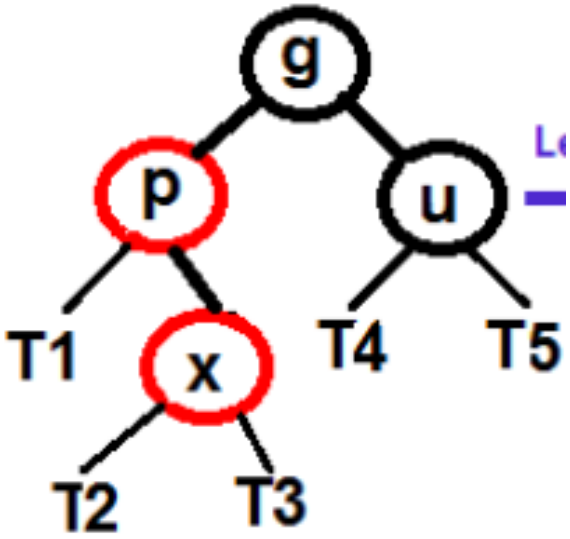
Rotations-Right Rotate P



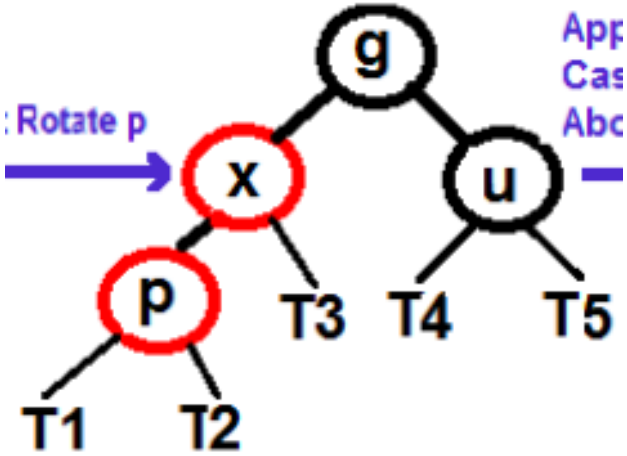
Answer



Left Rotate P



Answer



Let x be the newly inserted node.

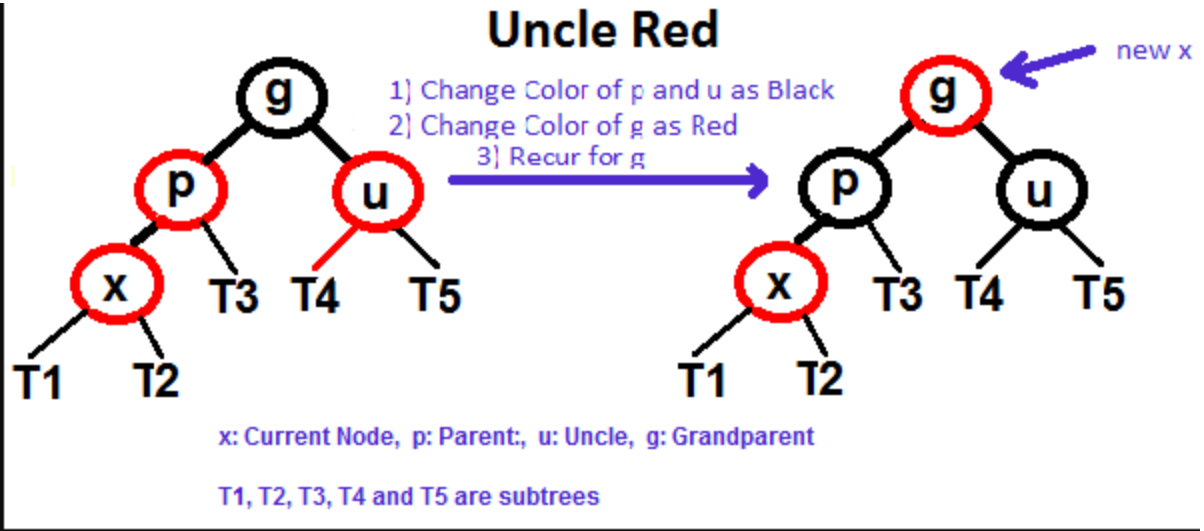
1) Perform standard BST insertion and make the color of newly inserted nodes as RED.

2) If x is root, change color of x as BLACK

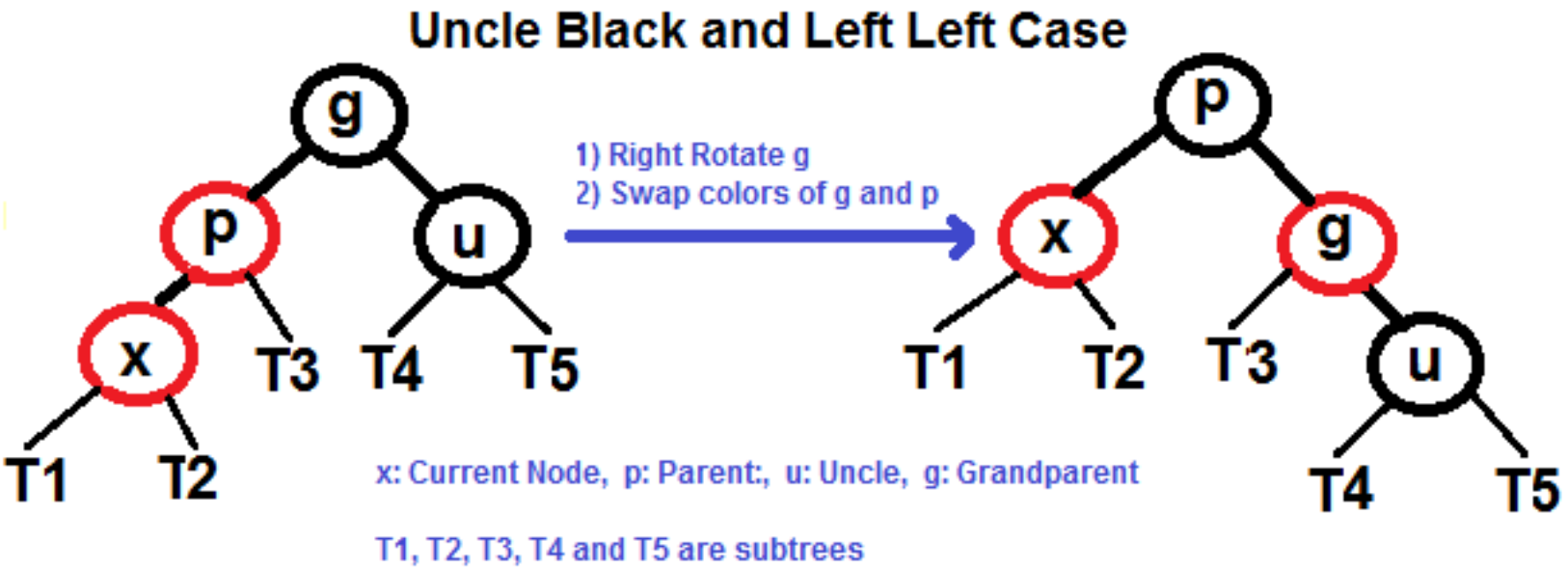
3) Do following if color of x’s parent is not BLACK or x is not root.

a) If x’s uncle is RED

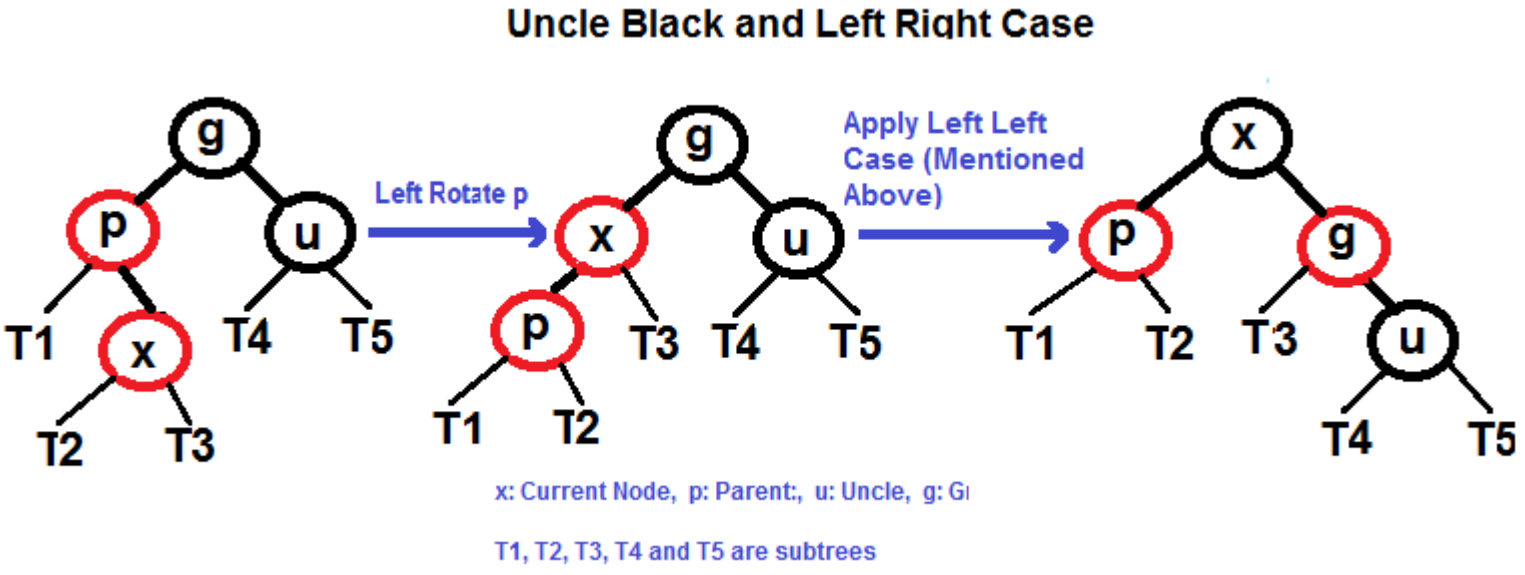
(i) Change color of parent and uncle as BLACK.
(ii) color of grandparent as RED.
(iii) Change x = x’s grandparent, repeat steps 2 and 3 for new x.



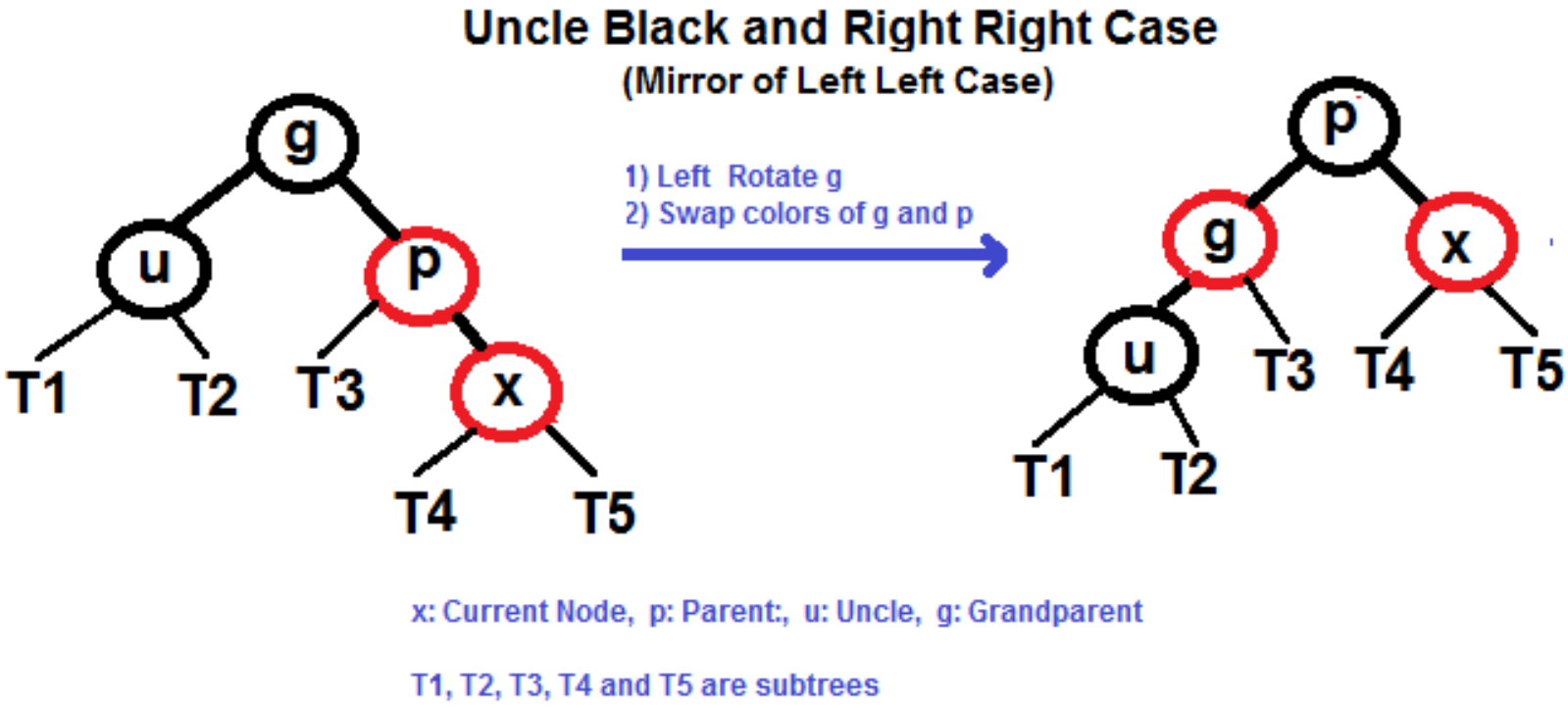
Left Left Case (p is left child of g and x is left child of p)



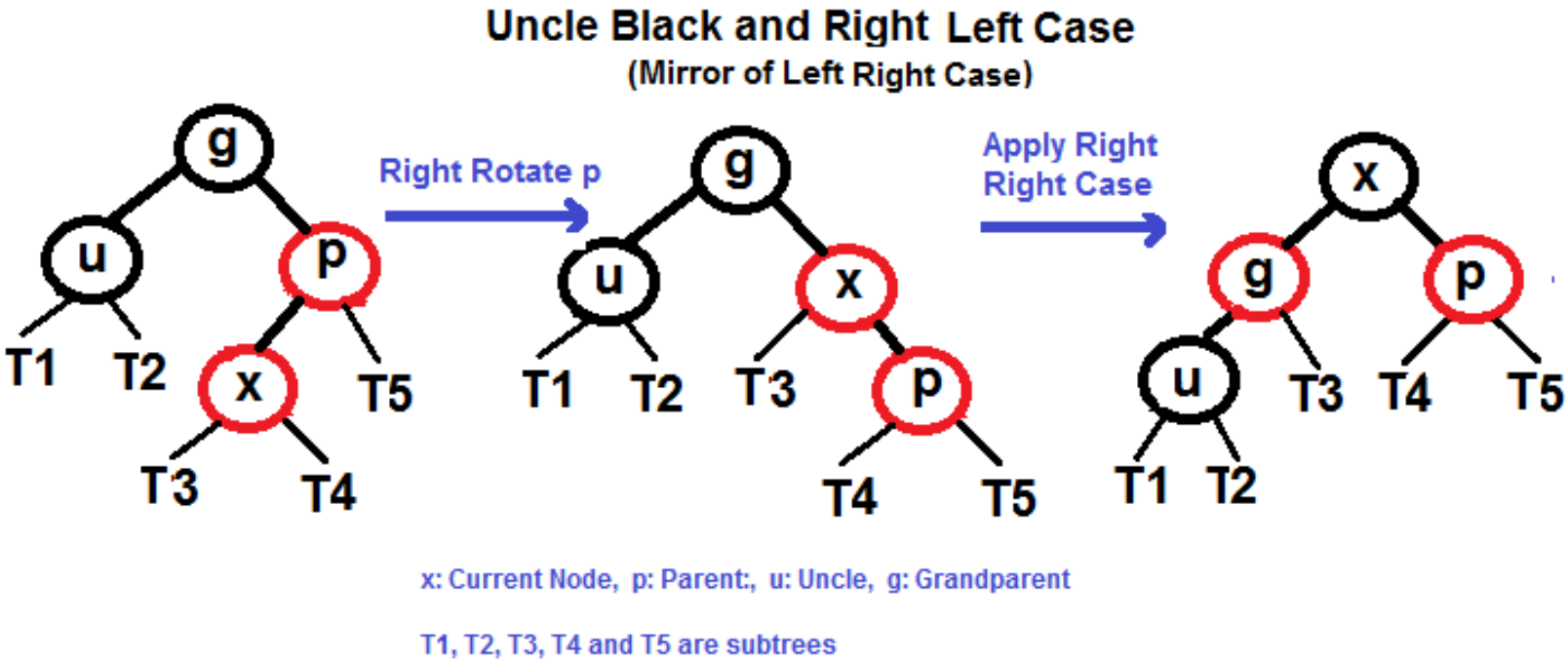
Left Right Case (p is left child of g and x is right child of p)



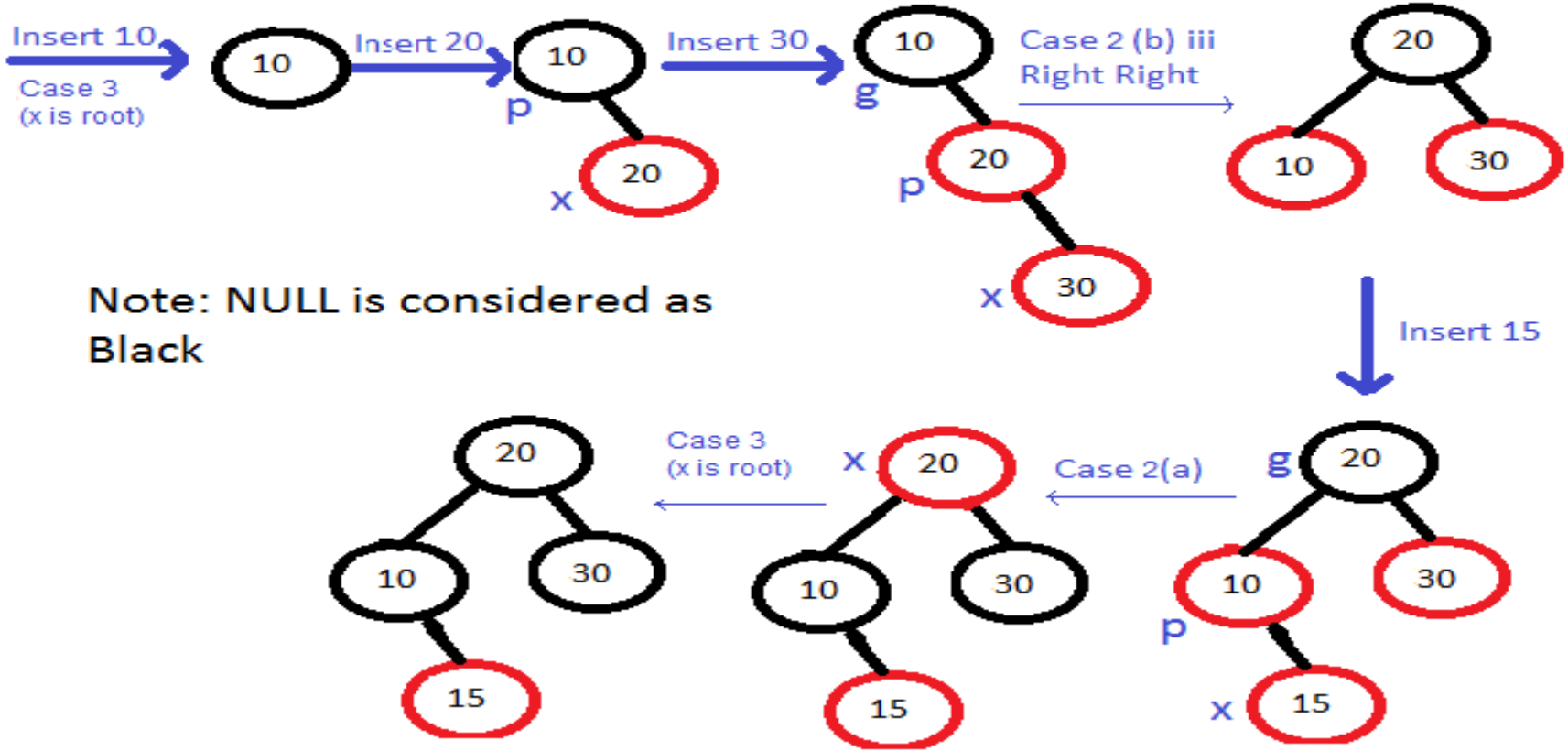
Right Right Case (p is right child of g and x is right child of p)



Right Left Case (p is right child of g and x is left child of p)



Insert 10, 20, 30 and 15 in an empty tree



Note: NULL is considered as Black

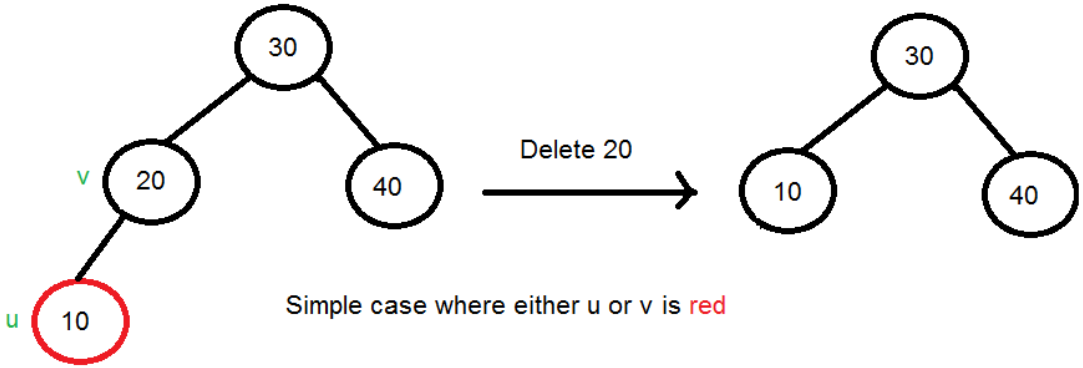
Deletion

In insert operation, we check color of uncle to decide the appropriate case. In delete operation, ***we check color of sibling*** to decide the appropriate case.

The main property that violates after insertion is two consecutive reds. In delete, the main violated property is, change of black height in subtrees as deletion of a black node may cause reduced black height in one root to leaf path.

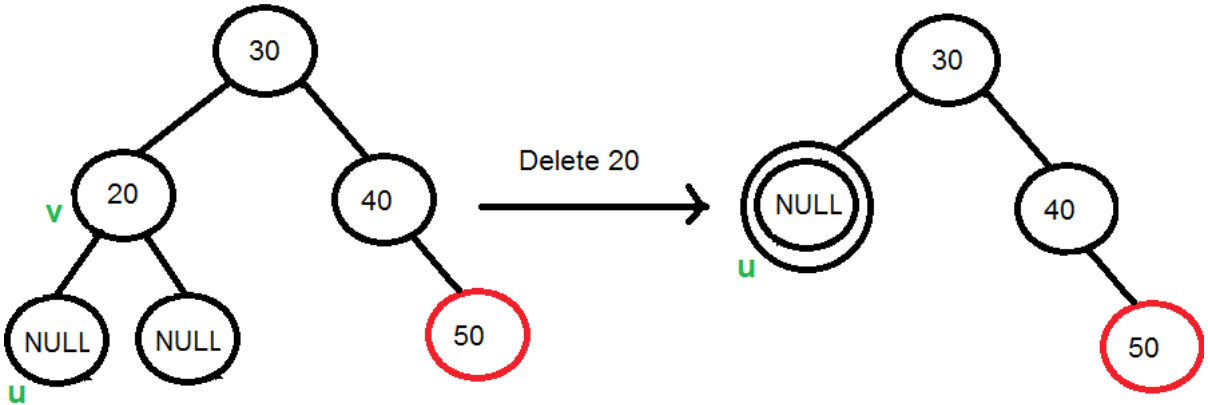
1) Perform standard BST delete. When we perform standard delete operation in BST, we always end up deleting a node which is either leaf or has only one child. (For node with 2 children, we replace with either left most node of right subtree or vice versa) **Let v be the node to be deleted and u be the child that replaces v** (Note that u is NULL when v is a leaf and color of NULL is considered as Black).

2) Simple Case: If either u or v is red, we mark the replaced child as black (No change in black height). Note that both u and v cannot be red as v is parent of u and two consecutive reds are not allowed in red.



3) If Both u and v are Black.

3.1) Color u as double black. Now our task reduces to convert this double black to single black. Note that If v is leaf, then u is NULL and color of NULL is considered as black. So the deletion of a black leaf also causes a double black.



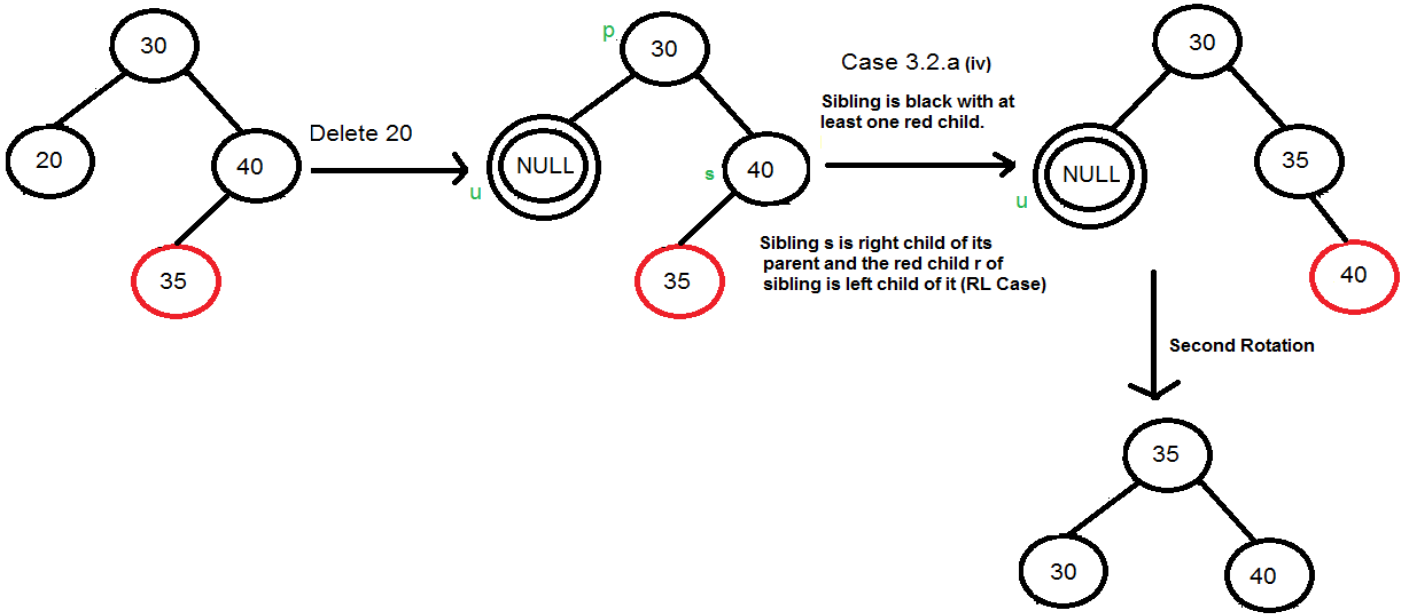
When 20 is deleted, it is replaced by a NULL, so the NULL becomes double black.
Note that deletion is not done yet, this double black must become single black

3.2) Do following while the current node u is double black and it is not root. Let sibling of node be **s**.

(a): If sibling s is black and at least one of sibling's children is red, perform rotation(s). Let the red child of s be **r**. This case can be divided in four subcases depending upon positions of s and r.

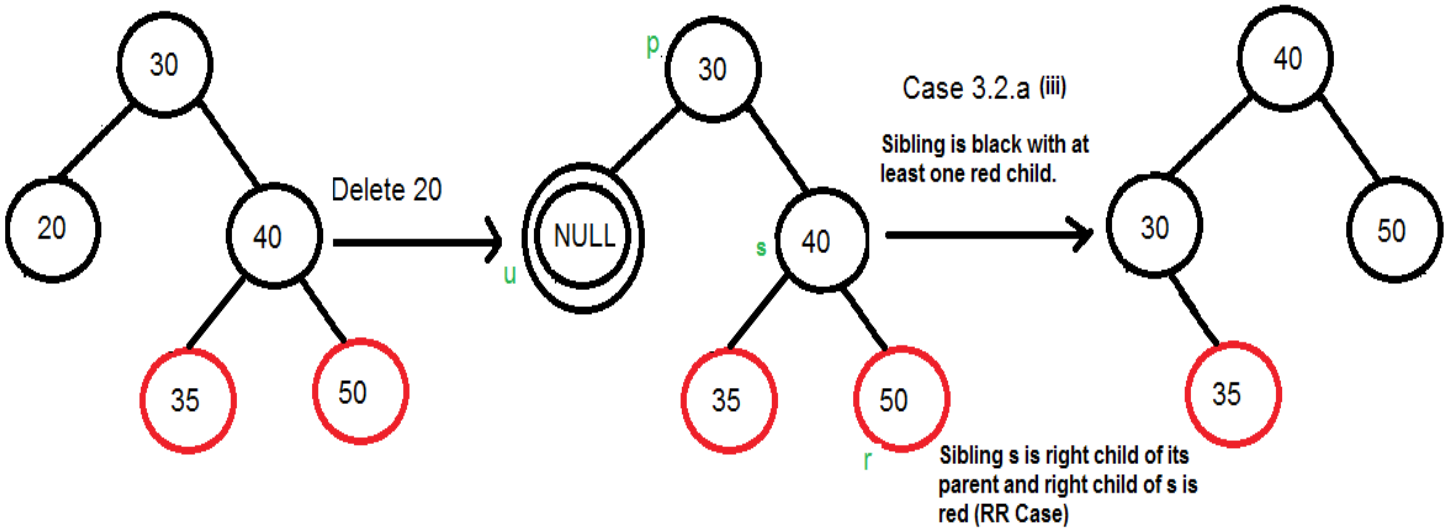
(i) Left Left Case (s is left child of its parent and r is left child of s or both children of s are red). Mirror of Below Diagram

(iii) Right Right Case (s is right child of its parent and r is right child of s or both children of s are red)



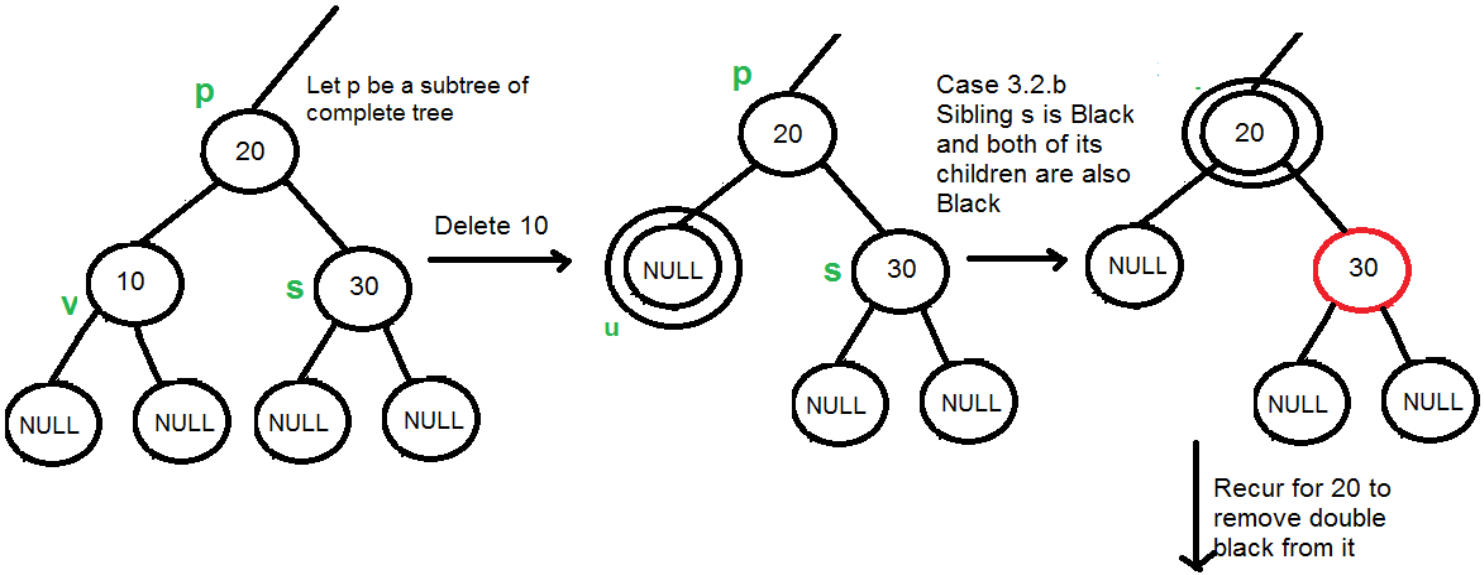
(ii) Left Right Case (s is left child of its parent and r is right child). Mirror of below diagram

(iv) Right Right Case (s is right child of its parent and r is right child of s or both children of s are red)



b) If sibling is black and its both children are black, perform recoloring, and recur for the parent if parent is black.

In this case, if parent was red, then we didn't need to recur for prent, we can simply make it black (red + double black = single black)

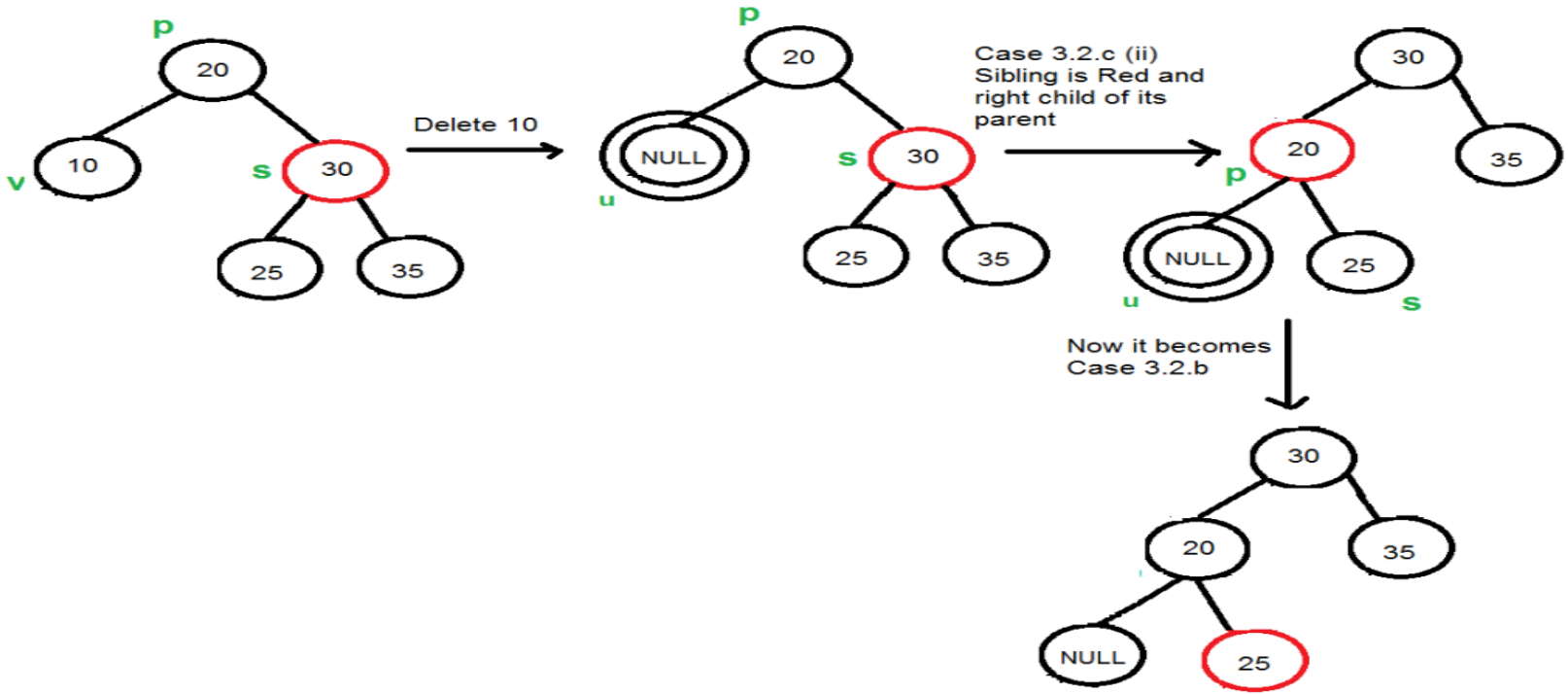


(c): If sibling is red, perform a rotation to move old sibling up, recolor the old sibling and parent. The new sibling is always black (See the below diagram). This mainly converts the tree to black sibling case (by rotation) and leads to case (a) or (b). This case can be divided in two subcases.

(i) Left Case (s is left child of its parent). This is mirror of right right case shown in below diagram. We right rotate the parent p.

(iii) Right Case (s is right child of its parent). We left rotate the parent p.

3.3) If u is root, make it single black and return (Black height of complete tree reduces by 1).


p
20
v
10
s
30
25
35
Delete 10
p
20
NULL
u
s
30
25
35
Case 3.2.c (ii)
Sibling is Red and
right child of its
parent
30
20
35
p
NULL
25
u
s
Now it becomes
Case 3.2.b
30
20
35
NULL
25