Red Black Tree

Arpan Khanna
200101121

Properties of Red Black Tree

1) Root Property

- The root is black

2) External Property

- Every leaf (leaf is a NULL child of a node, is black in Red-Black tree.

3) Internal Propery

- The children of a red node are black. Hence possible parent of red node is a black tree.

4) Path Property

- Every simple path from root to descendant leaf node contains same number of black nodes.

5) Depth Property

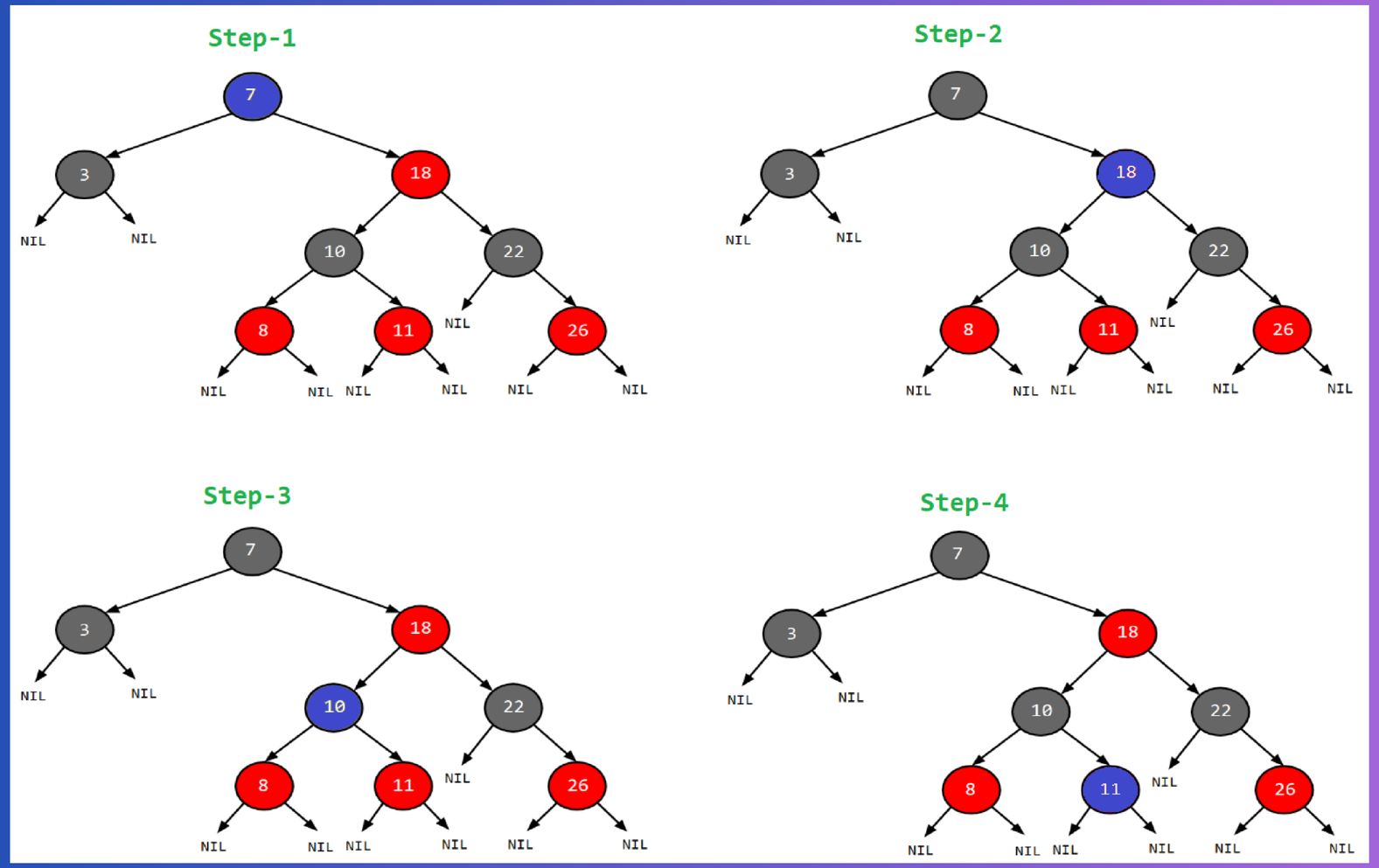
- All the leaves have the same black depth.

Why Red Black Trees?

- Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST.
- The cost of these operations may become O(n) for a skewed Binary tree. If we make sure that the height of the tree remains O(log n) after every insertion and deletion, then we can guarantee an upper bound of O(log n) for all these operations.
- The height of a Red-Black tree is always O(log n) where n is the number of nodes in the tree.

Example : Search for 11

Just follow the blue bubble



Search Algorithm

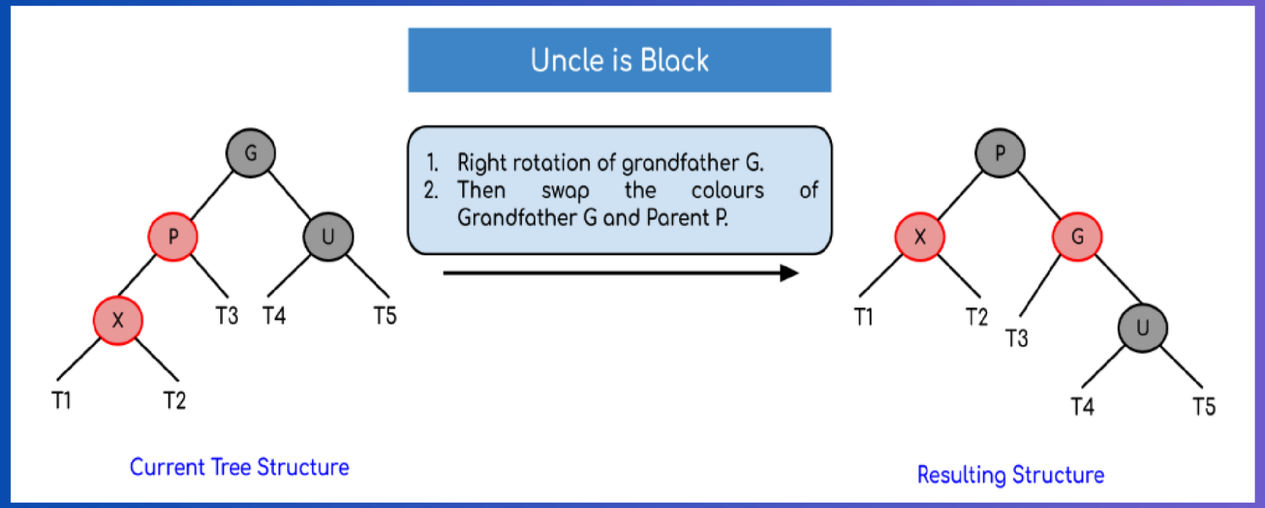
```
SEARCHELEMENT (TREE, VAL)

STEP 1:
IF TREE -> DATA = VAL OR TREE = NULL
    RETURN TREE
ELSE
IF VAL < DATA
        RETURN SEARCHELEMENT (TREE -> LEFT, VAL)
    ELSE
        RETURN SEARCHELEMENT (TREE -> RIGHT, VAL)
    [ END OF IF ]
[ END OF IF ]

STEP 2: END
```

Insertion

In the Red-Black tree, we use two tools to do the balancing.

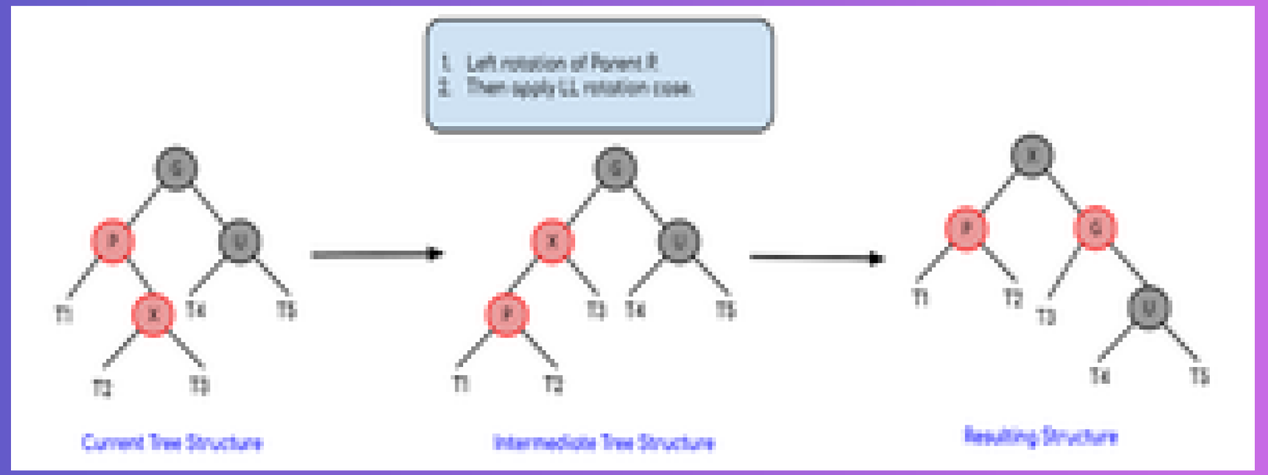
1) Recoloring
2) Rotation

Rotation and Recoloring

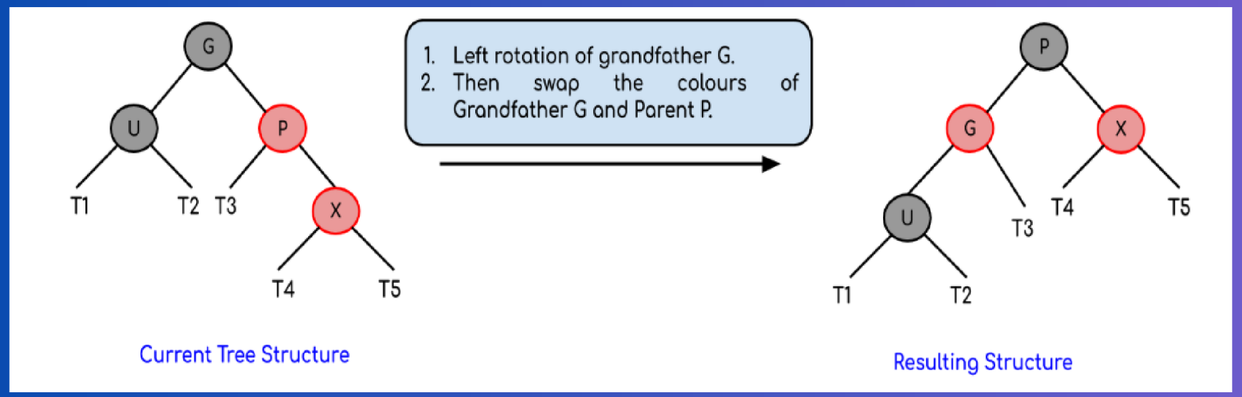


LL
ROTATION

LR
ROTATION

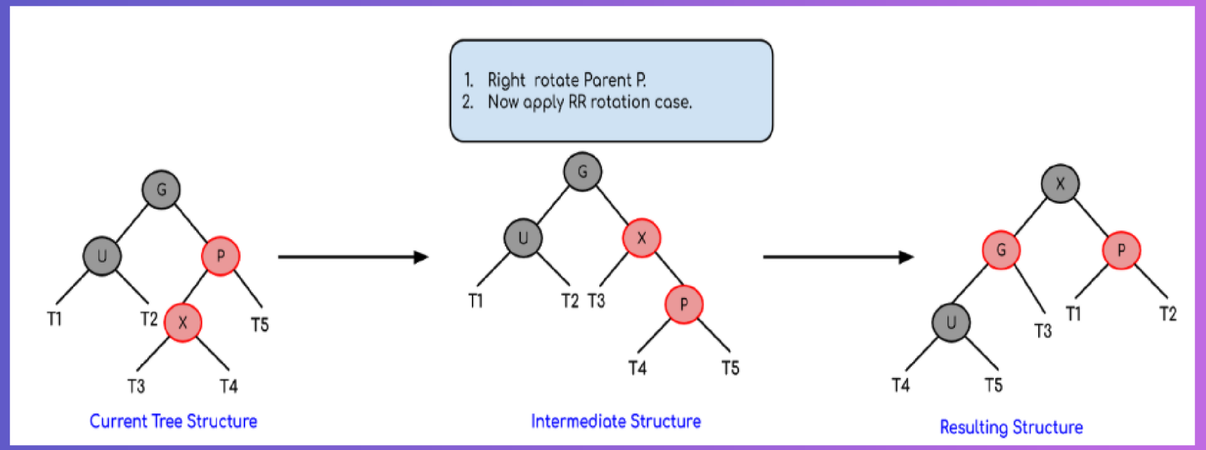


Rotation and Recoloring



RR ROTATION

RL ROTATION



Insertion Algorithm

Let x be the newly inserted node.
Perform standard BST insertion and make the colour of newly inserted nodes as RED.
If x is the root, change the colour of x as BLACK (Black height of complete tree increases by 1).

Do the following if the color of x's parent is not BLACK and x is not the root.
a) If x's uncle is RED (Grandparent must have been black from property 4)

- Change the colour of parent and uncle as BLACK.
- Colour of a grandparent as RED.
- Change x = x's grandparent, repeat steps 2 and 3 for new x.

b) If x's uncle is BLACK, then there can be four configurations for x, x's parent (p) and x's grandparent (g) (This is similar to AVL Tree)

- 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 the right child of p)
- Right Right Case (Mirror of case i)
- Right Left Case (Mirror of case ii)

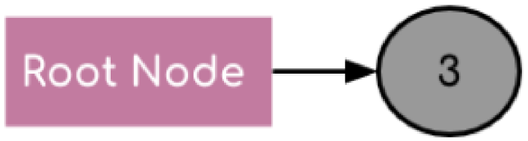
Re-coloring after rotations:

For Left Left Case [3.b (i)] and Right Right case [3.b (iii)], swap colors of grandparent and parent after rotations
For Left Right Case [3.b (ii)]and Right Left Case [3.b (iv)], swap colors of grandparent and inserted node after rotations

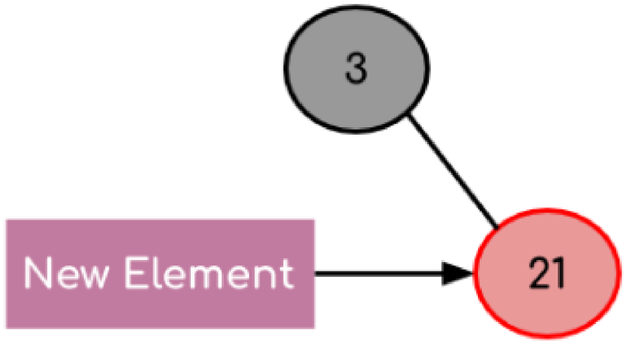
Insertion Example

Creating a red-black tree with elements 3, 21, 32 and 15

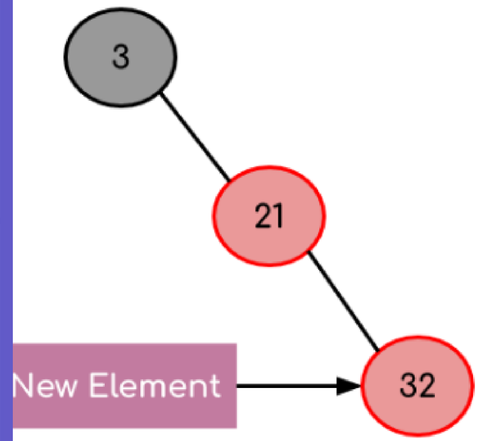
Step 1: Inserting element 3 inside the tree.



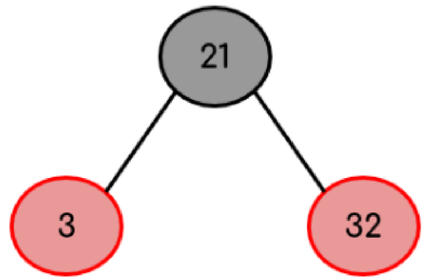
Step 2: Inserting element 21 inside the tree.



Step 3: Inserting element 32 inside the tree.

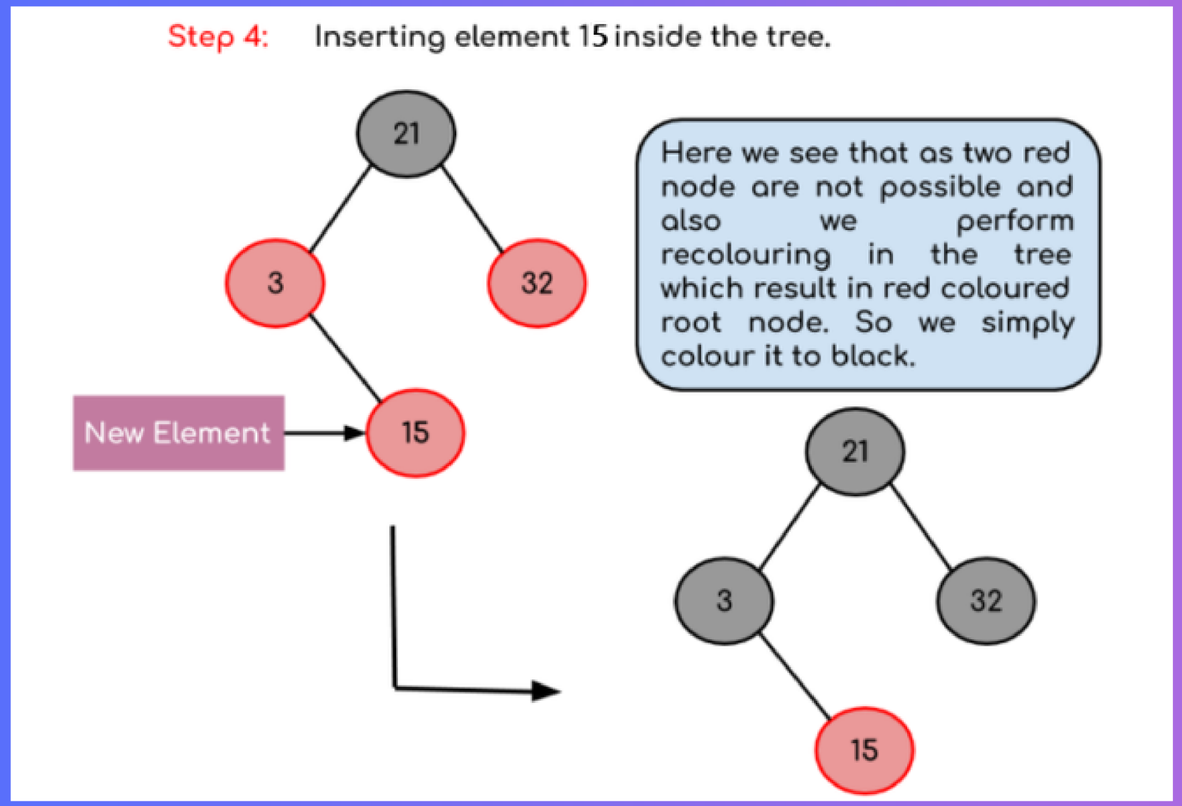


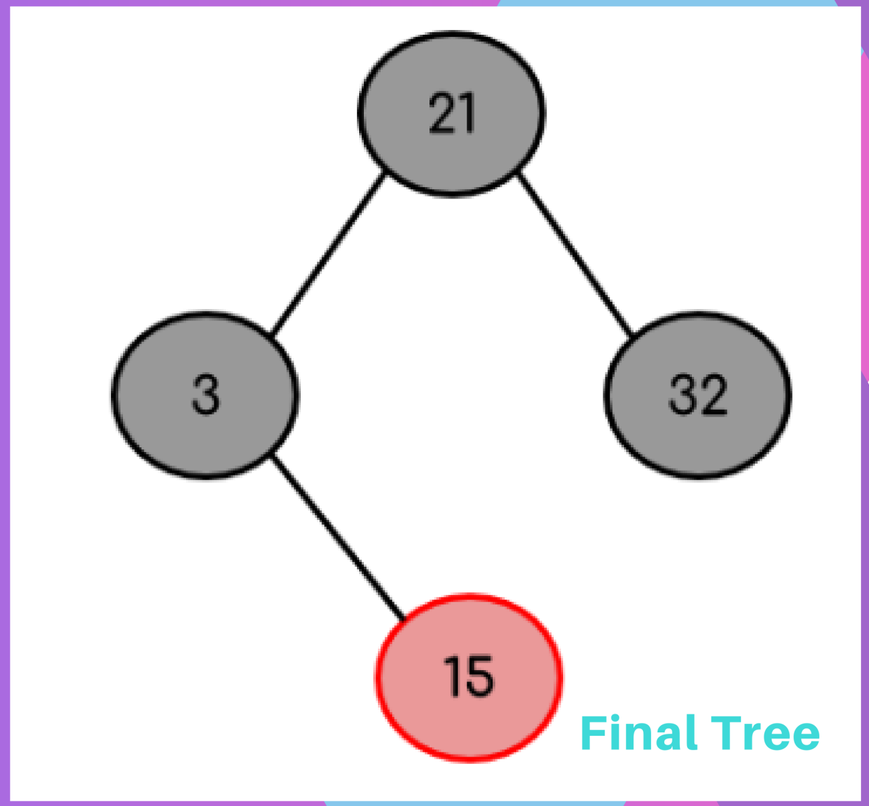
Here we see that as two red node are not possible and also we can see the conditions of RR rotation so it will follow RR rotation and recolouring to balance the tree.



Insertion Example Continued...

Creating a red-black tree with elements 3, 21, 32 and 15





Deletion

Deletion in a red-black tree is a bit more complicated than insertion. When a node is to be deleted, it can either have no children, one child or two children.

Steps Involved in Deletion

- If the node to be deleted has no children, simply remove it and update the parent node.

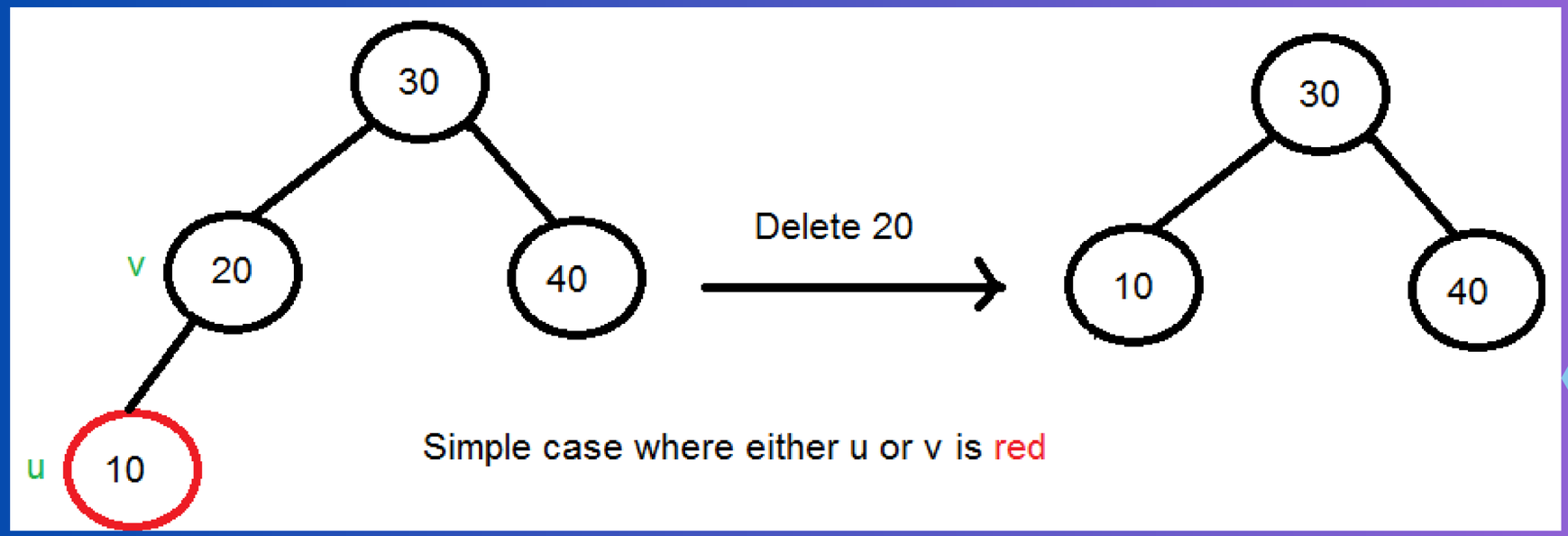
- If the node to be deleted has only one child, replace the node with its child.

- If the node to be deleted has two children, then replace the node with its in-order successor, which is the leftmost node in the right subtree. Then delete the in-order successor node as if it has at most one child.

- After the node is deleted, the red-black properties might be violated. To restore these properties, some color changes and rotations are performed on the nodes in the tree. The changes are similar to those performed during insertion, but with different conditions.
- The deletion operation in a red-black tree takes O(log n) time on average, making it a good choice for searching and deleting elements in large data sets.

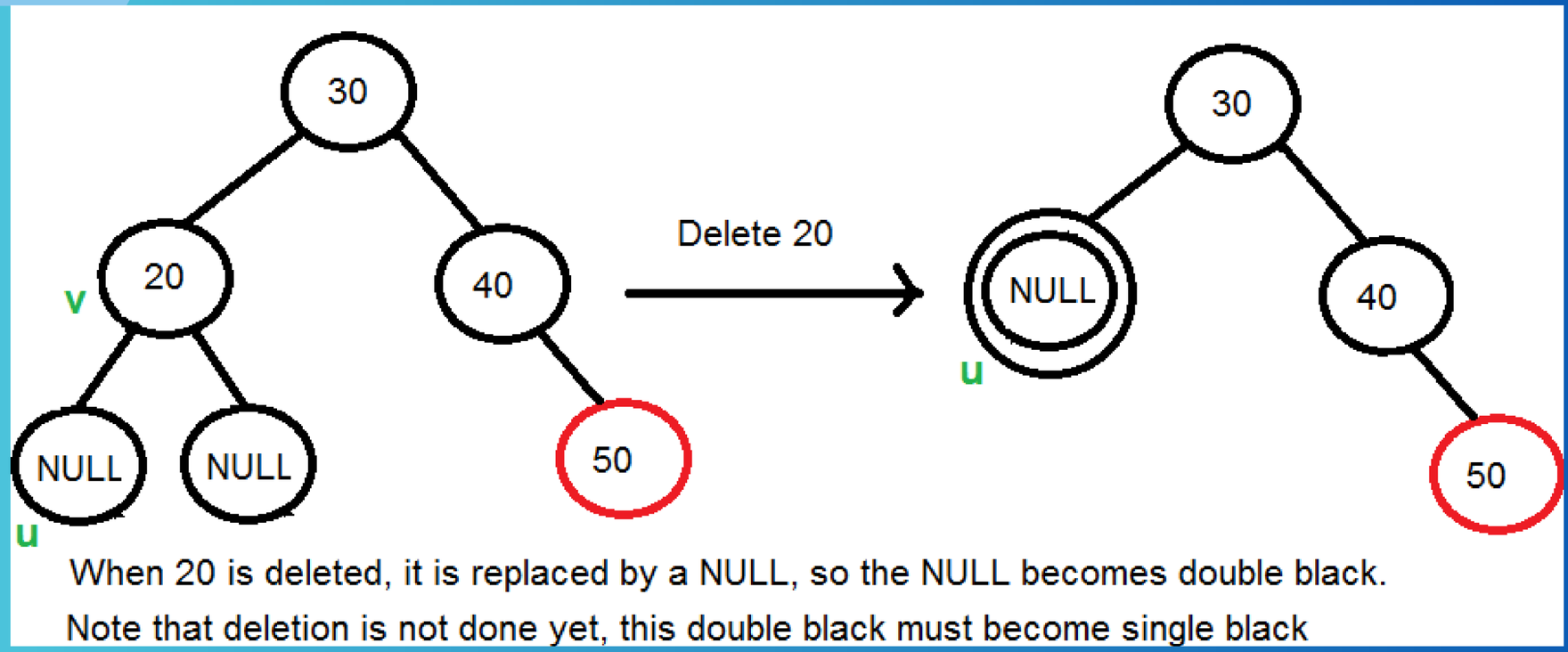
Simple Case

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-black tree.

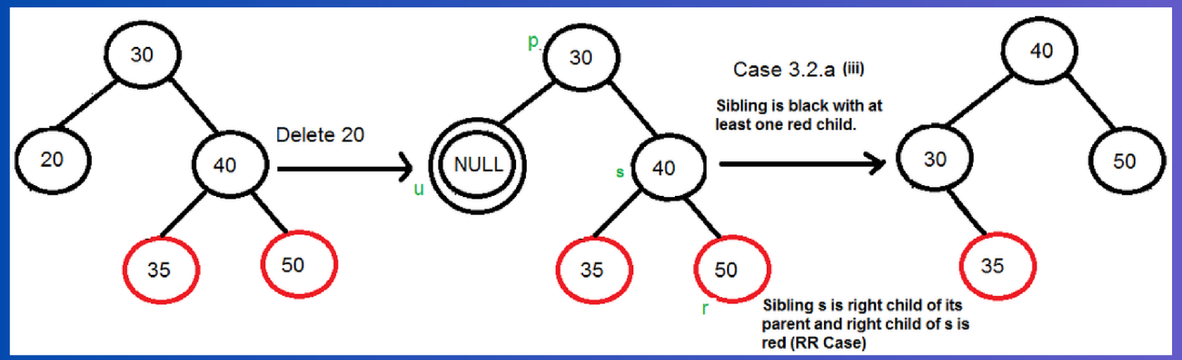


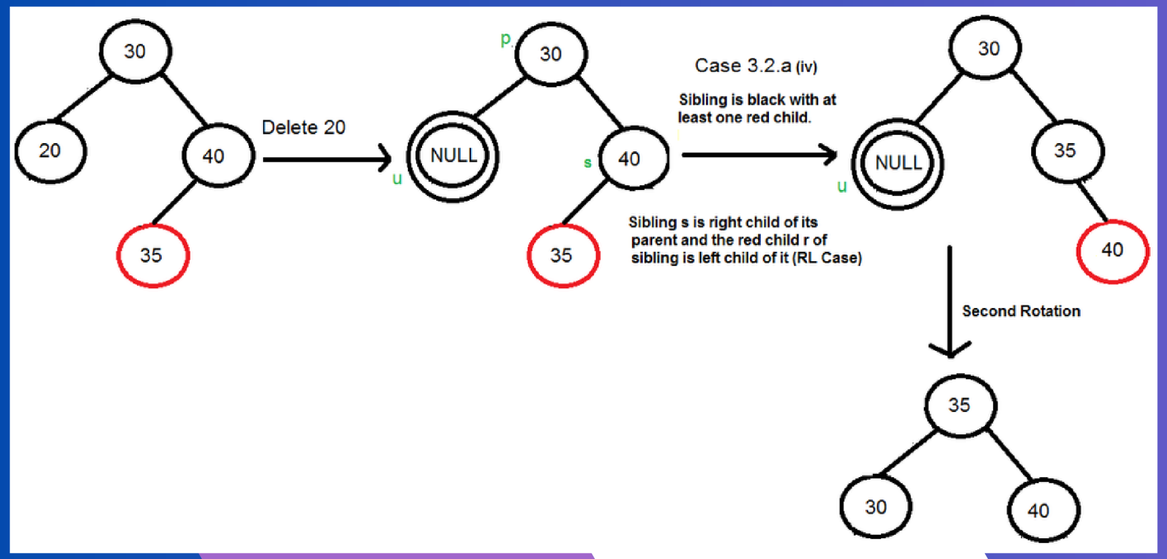
Case when If Both u and v are Black.

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 black. So the deletion of a black leaf also causes a double black.



Case 3.2a





Do the following while the current node u is double black, and it is not the root. Let the 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 into four subcases depending upon the 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). This is mirror of right right case shown in the below diagram.

……(ii) Left Right Case (s is left child of its parent and r is right child). This is mirror of right left case shown in 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)

……(iv) Right Left Case (s is right child of its parent and r is left child of s)

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

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

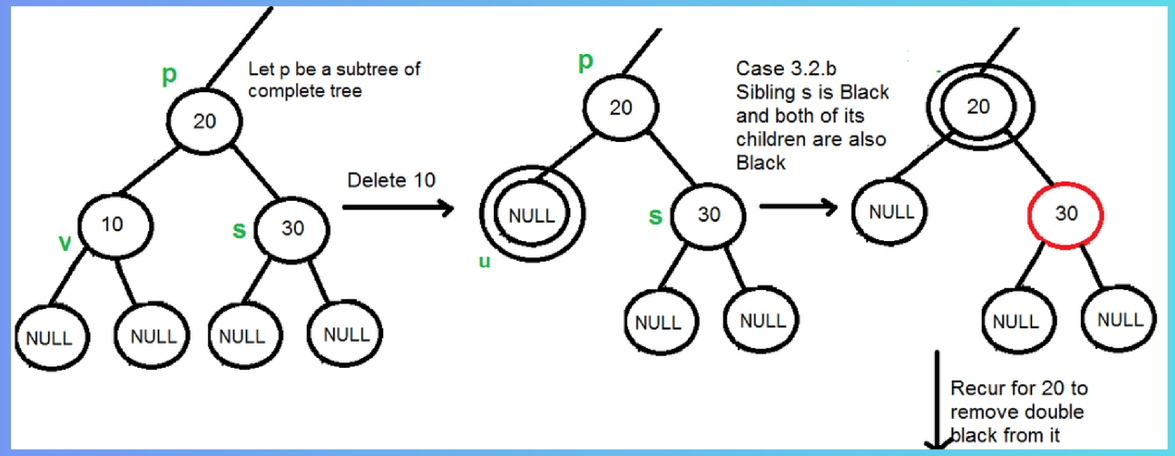
In this case, if parent was red, then we didn't need to recur for parent, we can simply make it black (red + double black = single black)

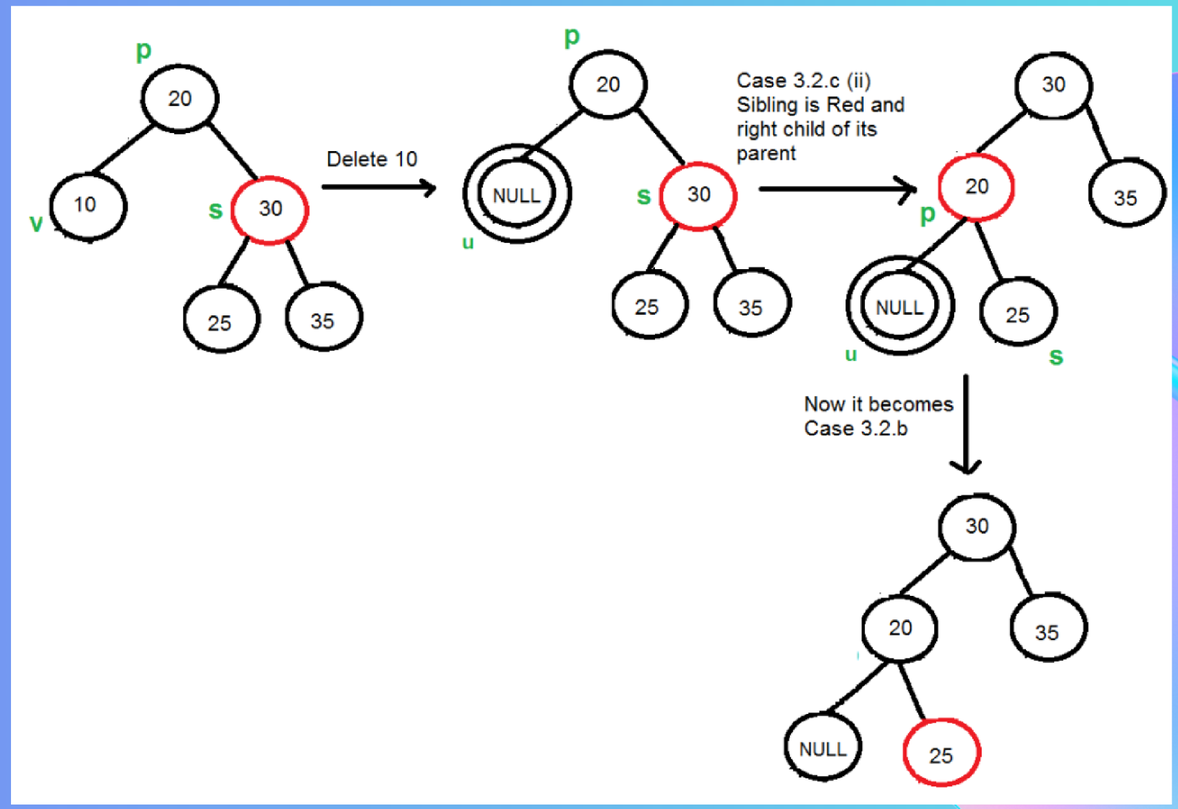
.....(c): If sibling is red, perform a rotation to move old sibling up, recolor the old sibling and parent. The new sibling is always black (See the below diagram). This mainly converts the tree to black sibling case (by rotation) and leads to case (a) or (b). This case can be divided in two subcases.

.....(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.

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

3.2 b and 3.2 c





3.3

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

Thank You