RED BLACK TREE

Deletion

Sambeg Shrestha 45
Bibek Mishra 33

RED BLACK TREE

Table of Contents

01
Introduction

02
Operations

03
Delete Operation

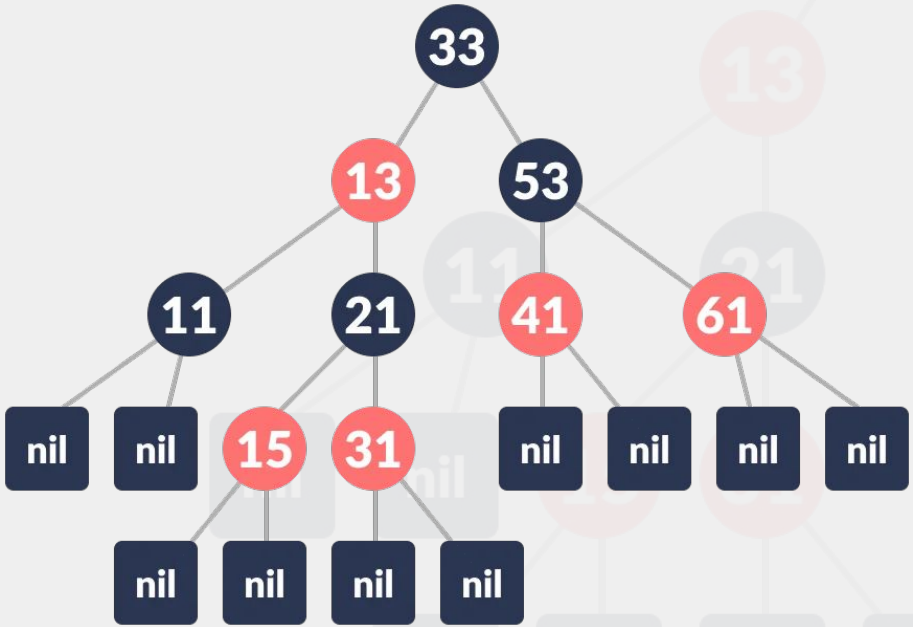
04
Extra

01
INTRODUCTION

RED BLACK TREE

A **Red-Black Tree** is a **Binary Search Tree** with following additional properties:

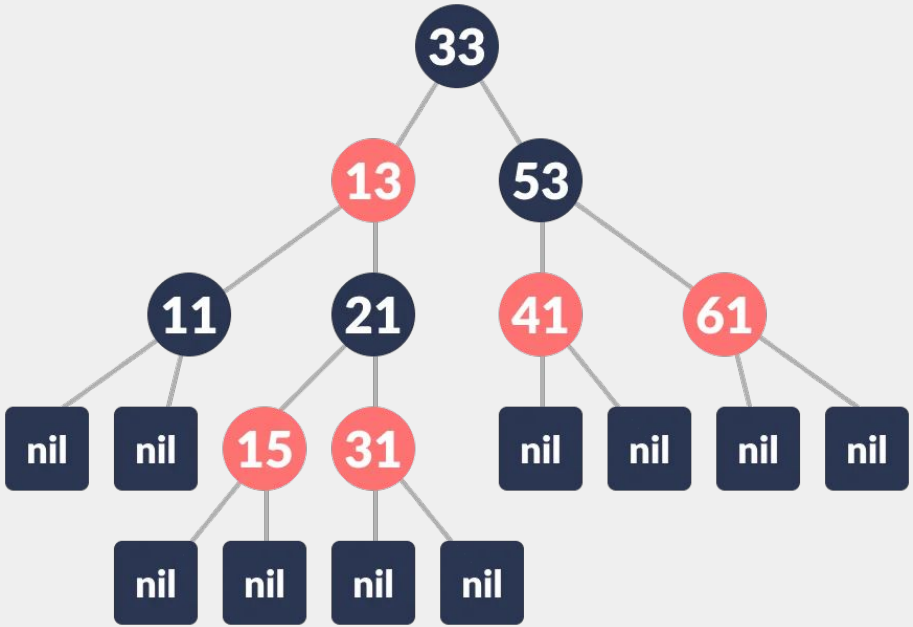
1. **BST**: Tree must be self-balancing **BST**.
2. **Red/Black Property**: Every node is colored, either **red** or **black**.
3. **Root Property**: The *root* is **black**.
4. **Leaf Property**: *Every leaf (NIL)* is **black**.
5. **Red Property**: If a **red node** has *children* then, the *children* are always **black.**
6. **Depth Property**: Every path from a given *node* to any of its *descendant NIL* nodes *contains* the *same number* of *black nodes.*



RED BLACK TREE

A **Red-Black Tree** is a **self-balancing Binary Search Tree** each node of the binary tree has an extra bit to represent the color (red or black) of the node.

These color bits are used to ensure the tree remains approximately balanced during insertions and deletions.

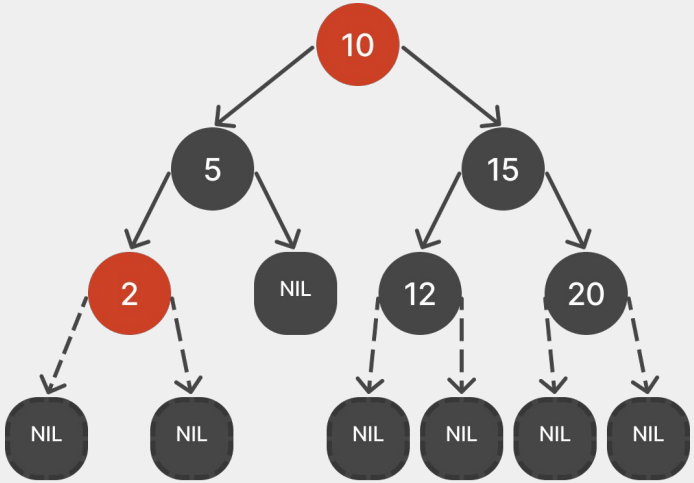


Attributes of each node:

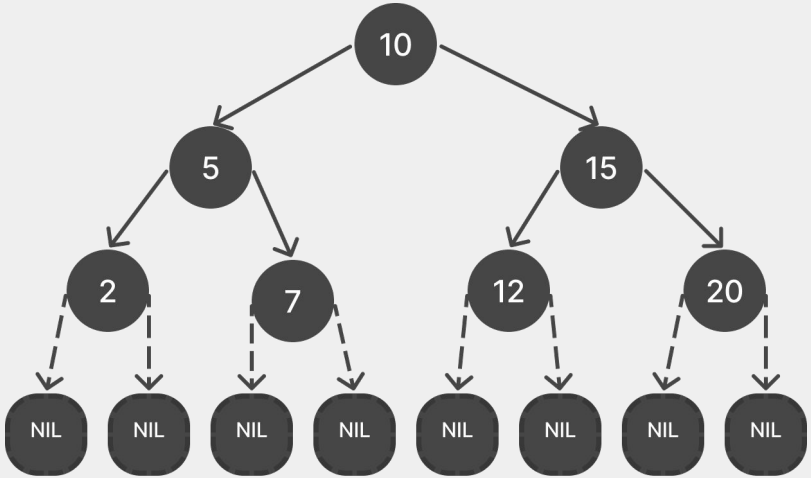
- color
- key
- leftChild
- rightChild
- parent (except root node)

RED BLACK TREE

Examples



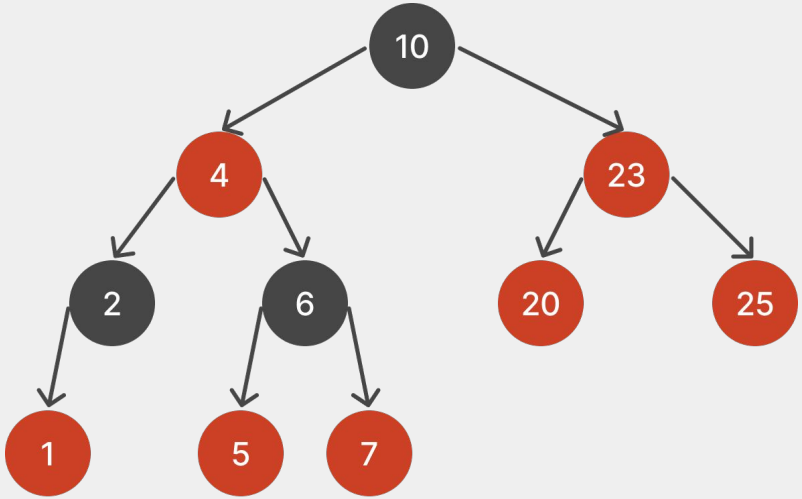
✗ RBT
✗ Root Property



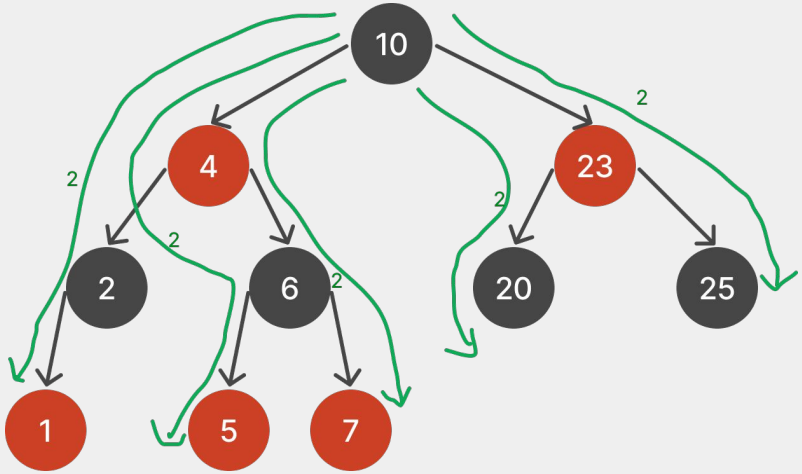
✓ RBT
✓ All Properties

RED BLACK TREE

Examples



✗ RBT
✗ Red Property



✓ RBT
✓ All Properties

RED BLACK TREE

02
OPERATIONS

RED BLACK TREE

OPERATIONS ON RBT

1. **Recolor:** Change the color of one node from black(0) to red(1) or vice versa.
2. **Rotation:** In rotation operation, the positions of the nodes of a subtree are interchanged.
    a. **Left Rotate:** In left-rotation, the arrangement of the nodes on the right is transformed into the arrangements on the left node.
    b. **Right Rotate:** In right-rotation, the arrangement of the nodes on the left is transformed into the arrangements on the right node.
    c. **Left-Right and Right-Left Rotate:** In left-right rotation, the arrangements are first shifted to the left and then to the right and vice versa for right-left.
3. **Inserting an element into a Red-Black Tree:** Inserting a node into the RBT. It includes inserting a node similar to BST then maintaining RBT property by either Recoloring or rotating or both.
4. **Deleting an element from a Red-Black Tree:** Like insertion, deletion also takes place like in BST. Then, to maintain RBT property it is either recolored or rotated or both.

03
DELETE OPERATION

RED BLACK TREE DELETION ON RBT

DELETING A NODE

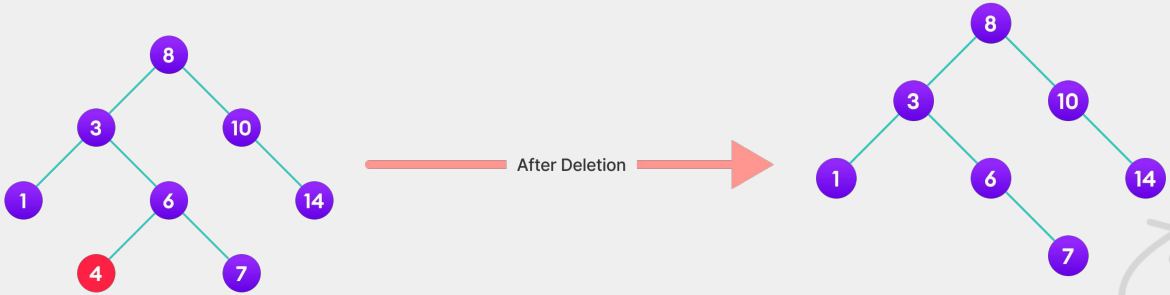
The delete operation works like the delete operation in BST. A node is deleted from this operation and later red black property is maintained.

Step 1: Perform BST Deletion

Recalling BST Deletion

A. Deleting leaf node

Simply delete the node

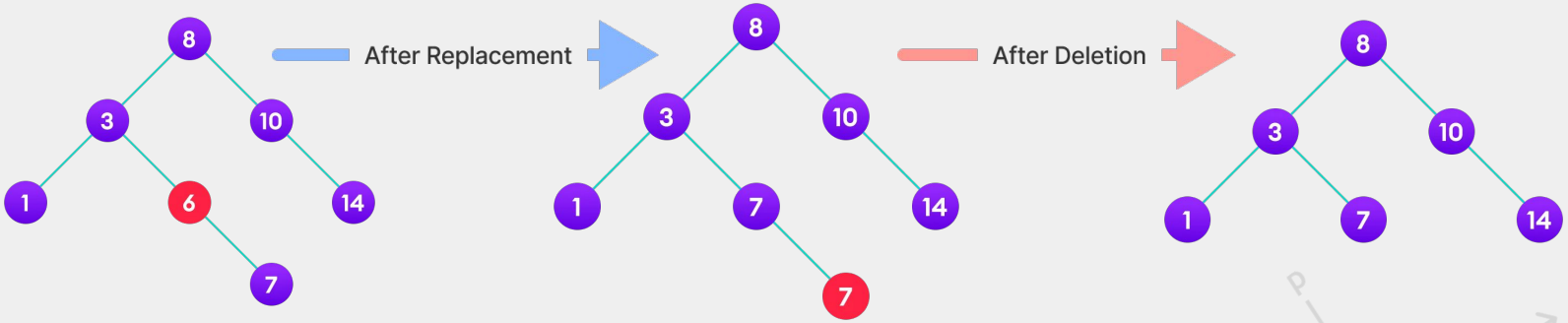


RED BLACK TREE DELETION ON RBT

1. Perform BST Deletion

B. **Deleting node with single child**

Replace the child with the node and delete the child from original position

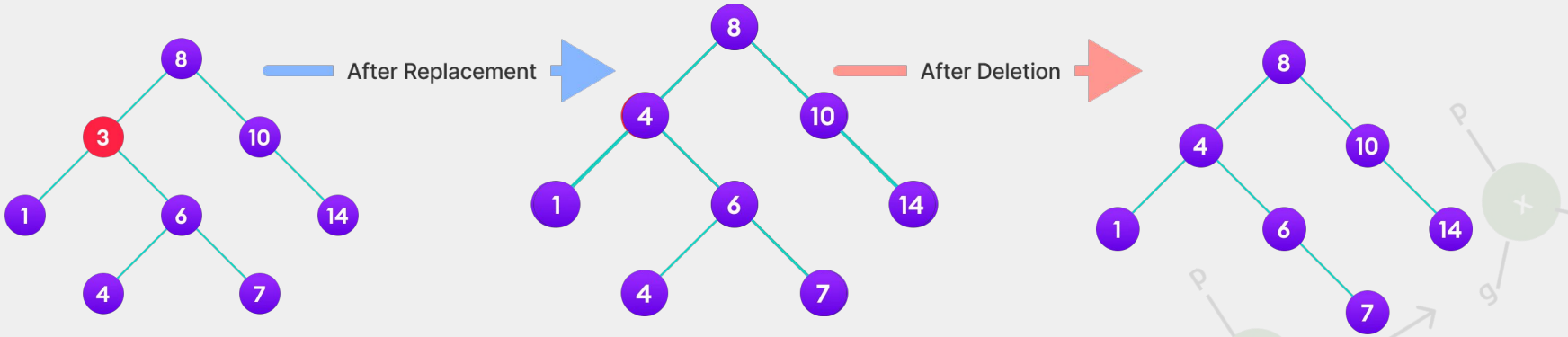


RED BLACK TREE DELETION ON RBT

1. Perform BST Deletion

C. Deleting node with double child

Replace the node with its inorder successor and delete the successor from original position



RED BLACK TREE

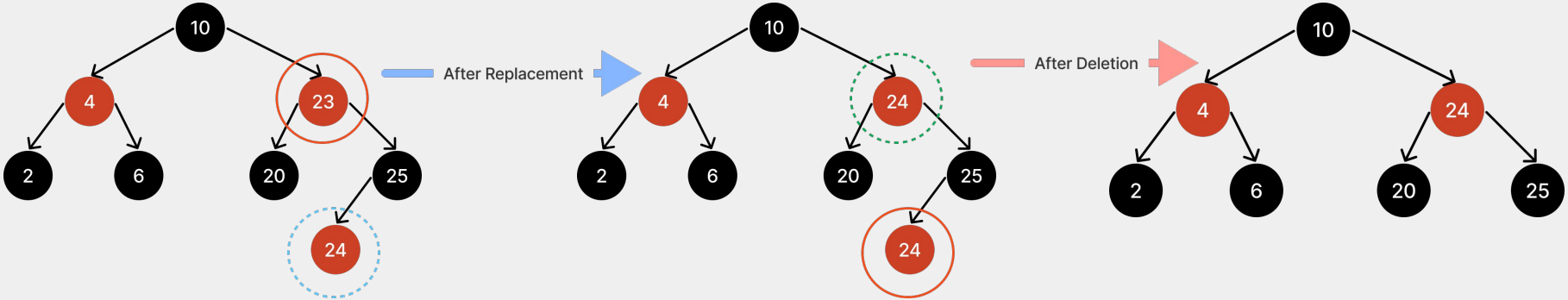
DELETION ON RBT: CASE 1

CHECKING AND BALANCING TREE

Step 2: Checking and Balancing the tree

It is necessary to check the colors of nodes to maintain RB property and balance the tree appropriately while deleting. There are some cases that can help us analyze and assess that.

Case 1: If node to be deleted is red, delete it.



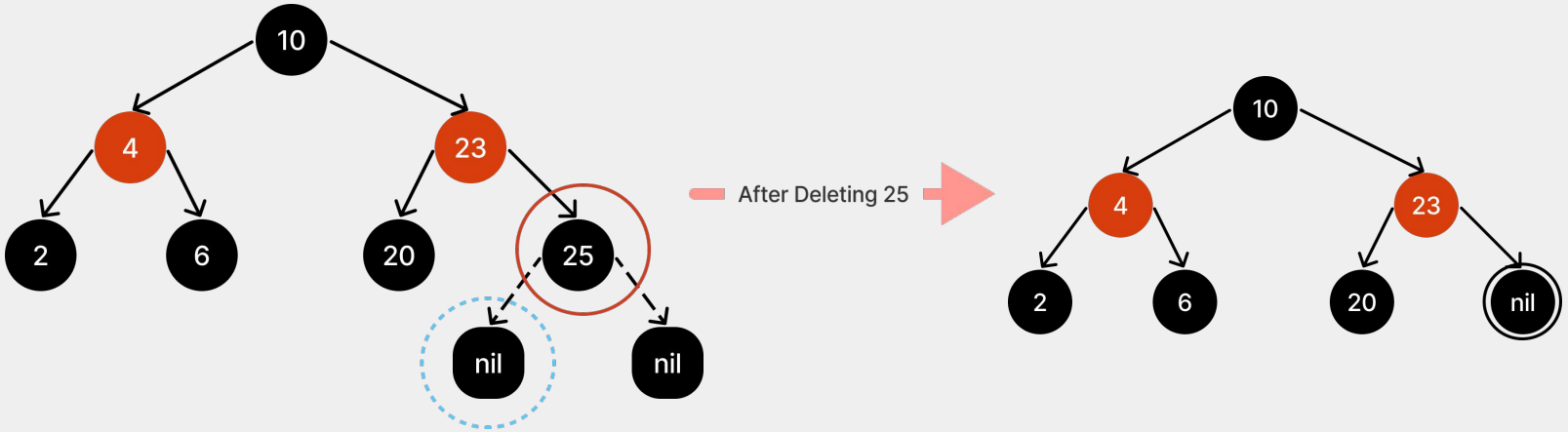
RED BLACK TREE

DELETION ON RBT: DB

2.

Understanding Double Black (DB)

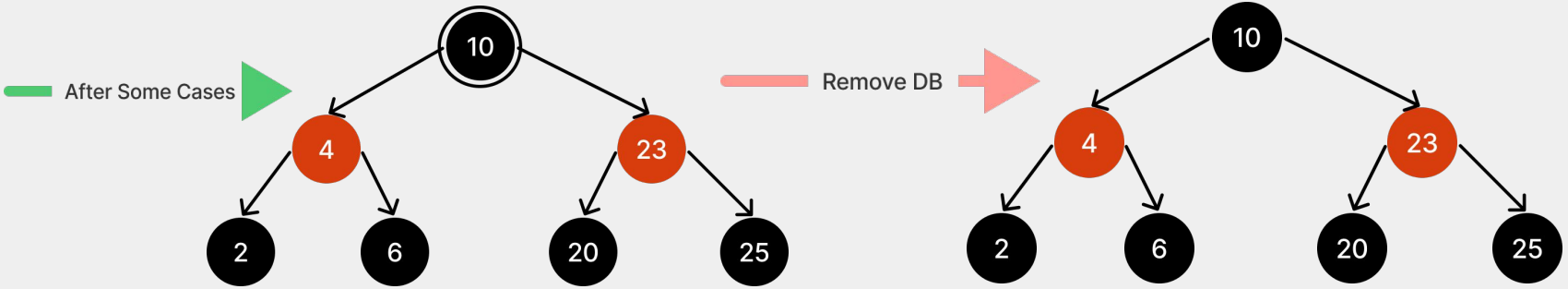
When Deleting a Black node, if it is replaced by another black node (inorder successor or predecessor), the node to delete becomes Double Black.



RED BLACK TREE

DELETION ON RBT: CASE 2

2. Case 2: If root is double black (DB), just remove double black.



RED BLACK TREE

DELETION ON RBT: CASE 3

2. **Case 3: If DB's sibling is black and both its children are black**

   a. Remove **DB**

   b. Recolor it's Parent(P) to Black and it's Sibling(S) to Red. Depending on P's color another two cases may arise:

      i. If P was originally Red, it becomes Black.
      ii. If P was originally Black, it becomes **Double Black.**

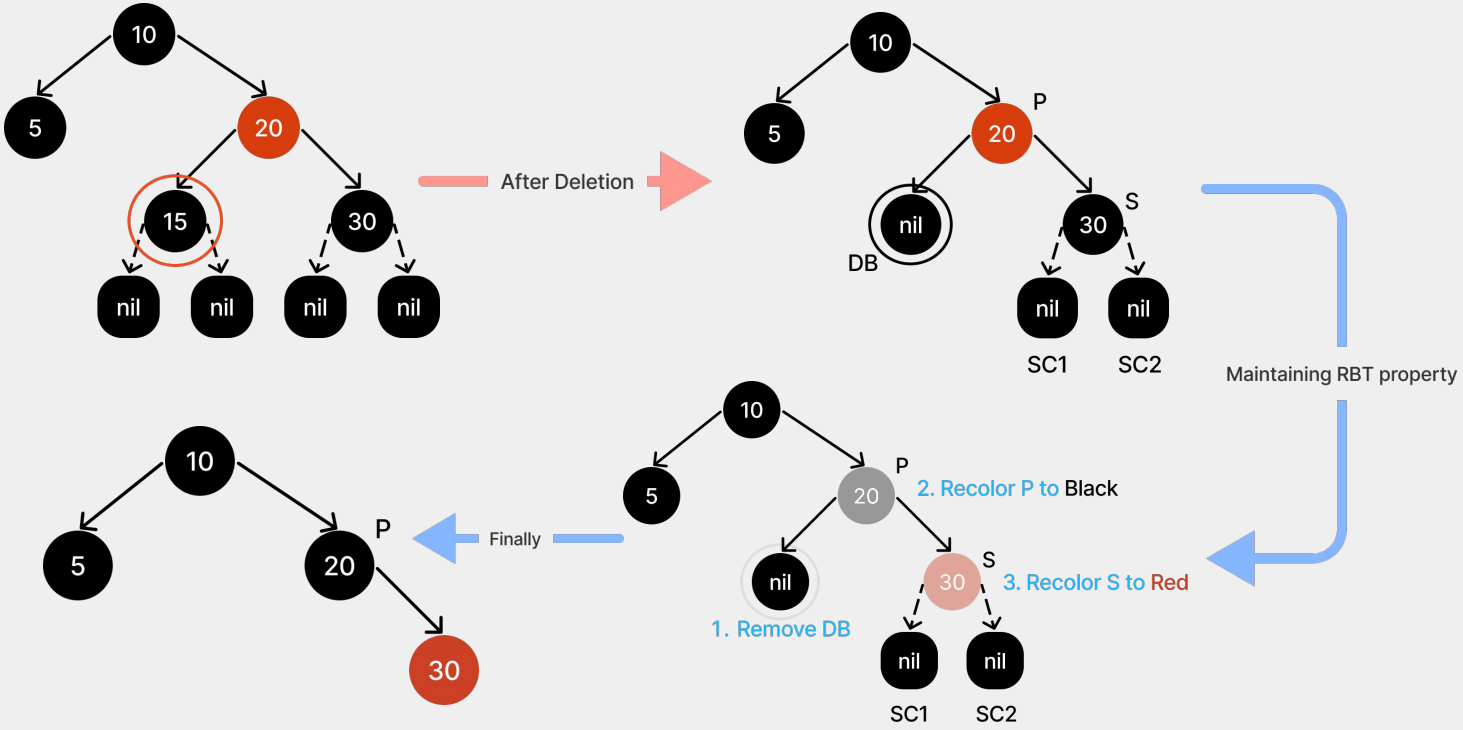
   c. Recolor **DB**'s sibling to Red.

   d. If still **DB** exists, e.g. when P becomes **DB,** other **CASES** should be applied**.**

RED BLACK TREE

DELETION ON RBT: CASE 3

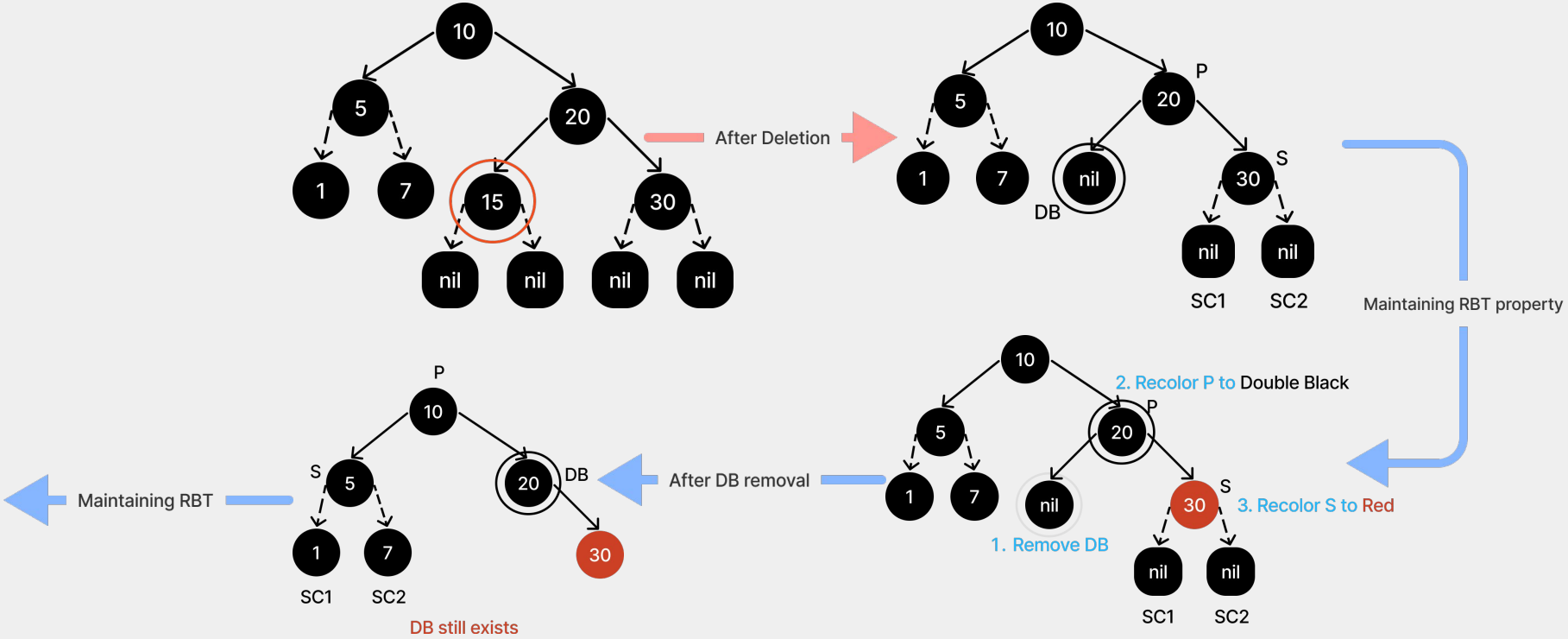
2. Case 3: If DB's S is black and S's both children's are black and P is red.



RED BLACK TREE

DELETION ON RBT: CASE 3

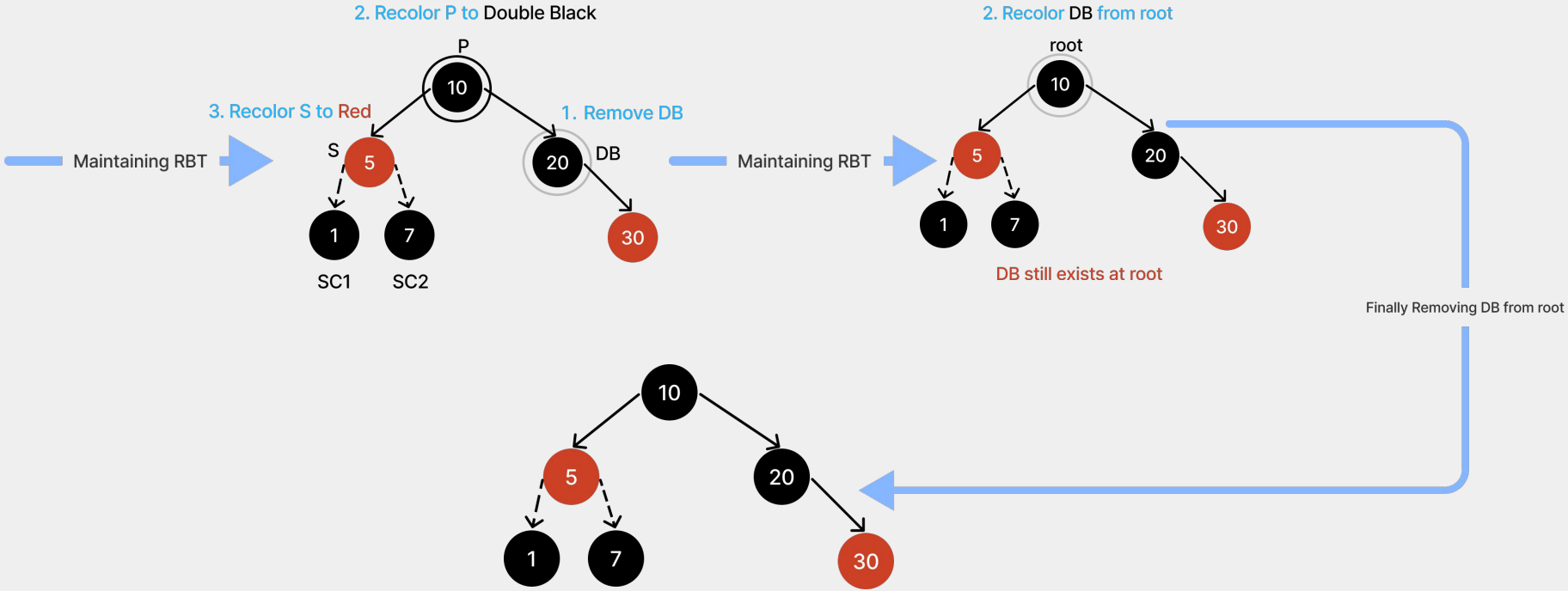
2. Case 3: If DB's S is black and S's both children's are black and P is also black.



RED BLACK TREE

DELETION ON RBT: CASE 3

2. Case 3: If DB's S is black and S's both children's are black and P is also black.

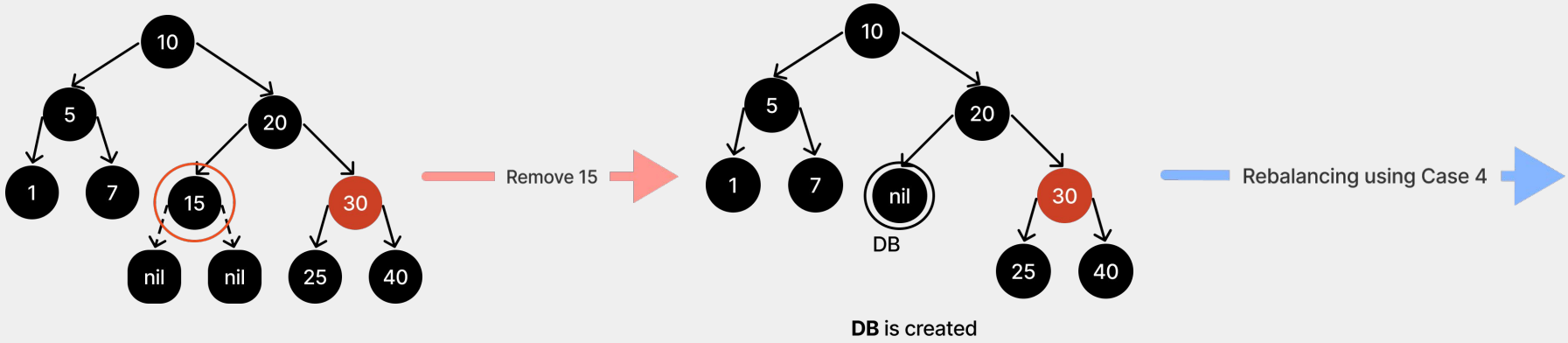


RED BLACK TREE

DELETION ON RBT: CASE 4

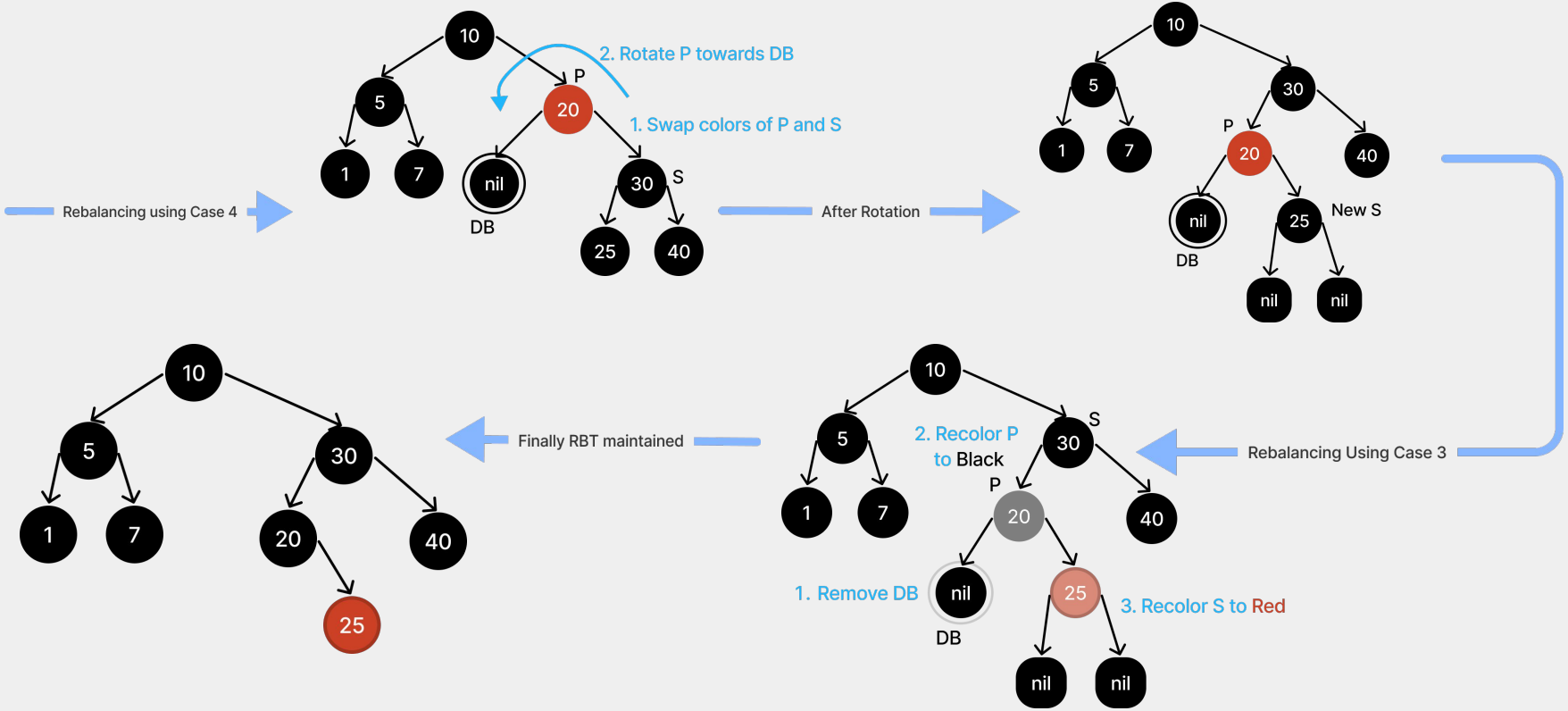
2. **Case 4: If DB's sibling(S) is red,**

a. Swap Colors of Parent(P) and Sibling(S)
b. Rotate the Parent in **DB** direction
c. Re-apply other cases until RBT maintained.



RED BLACK TREE

DELETION ON RBT: CASE 4

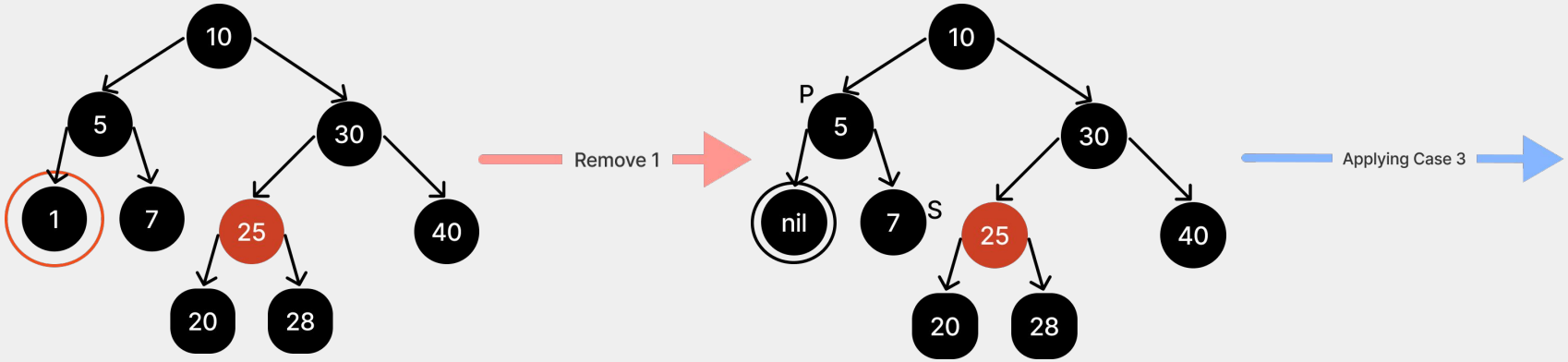


RED BLACK TREE

DELETION ON RBT: CASE 5

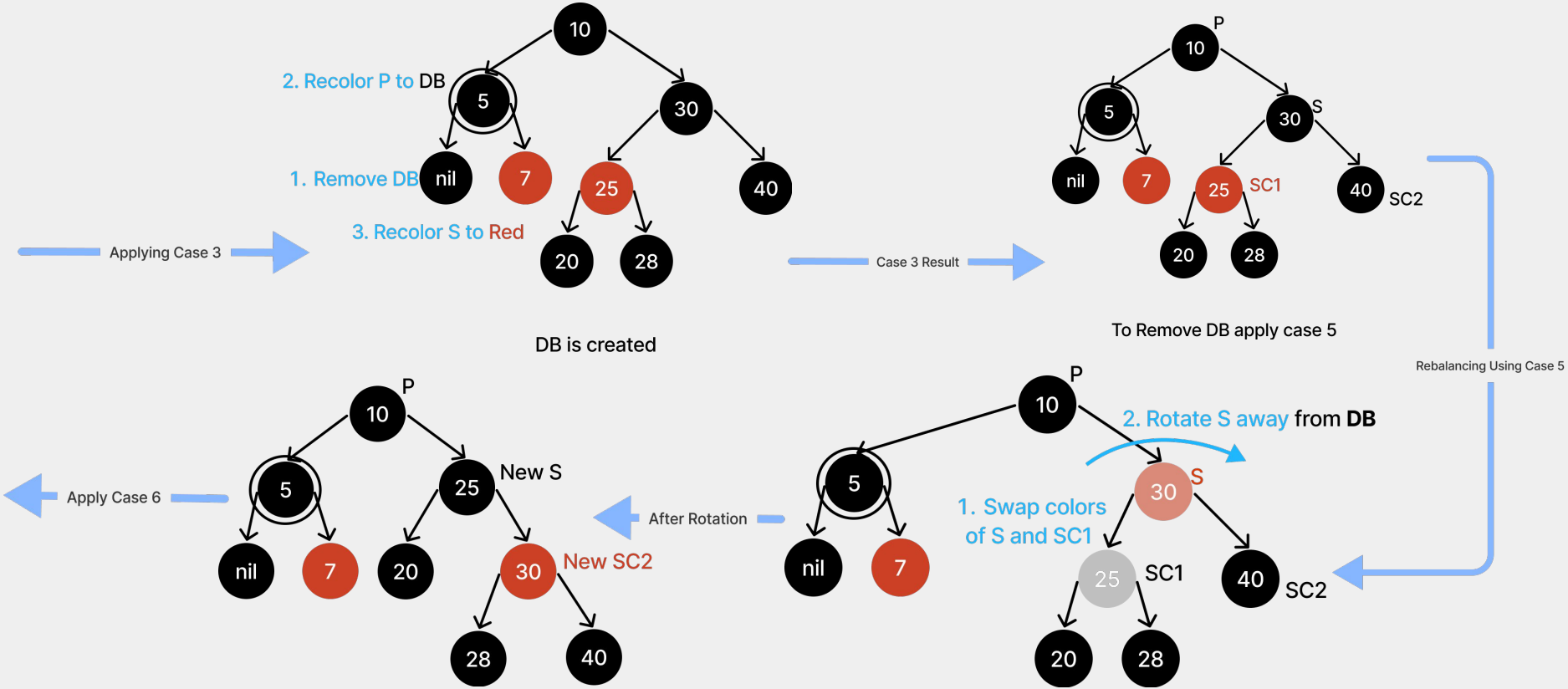
2. **Case 5: DB's sibling(S) is Black, sibling's child which is far from DB(SC2) is black but the child which is near (SB1) is Red**

a. Swap the color of S and SB1 i.e. Sibling and Near Child of Sibling
b. Rotate the sibling in opposite direction to DB
c. Apply case 6



RED BLACK TREE

DELETION ON RBT: CASE 5

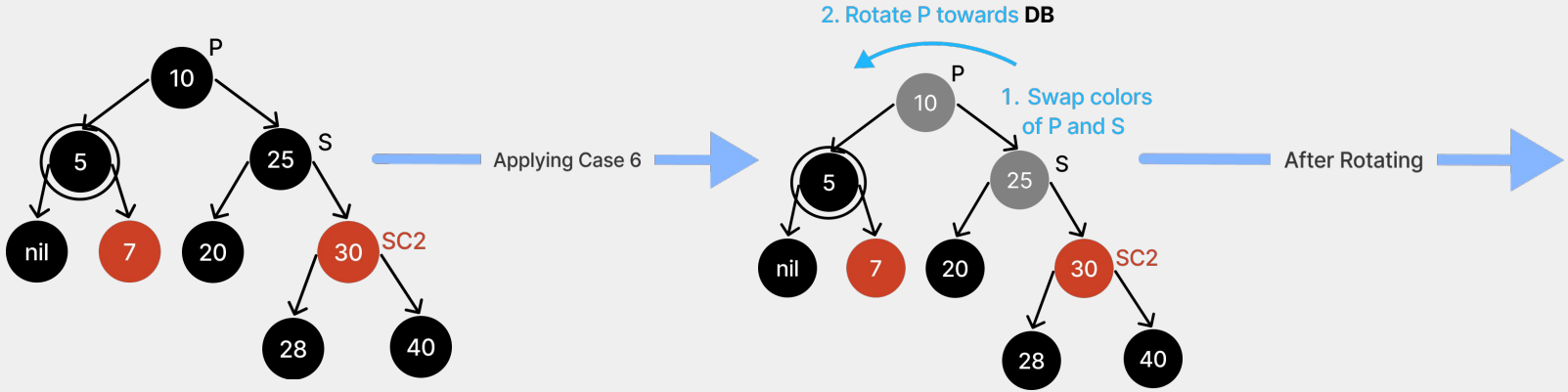


RED BLACK TREE

DELETION ON RBT: CASE 6

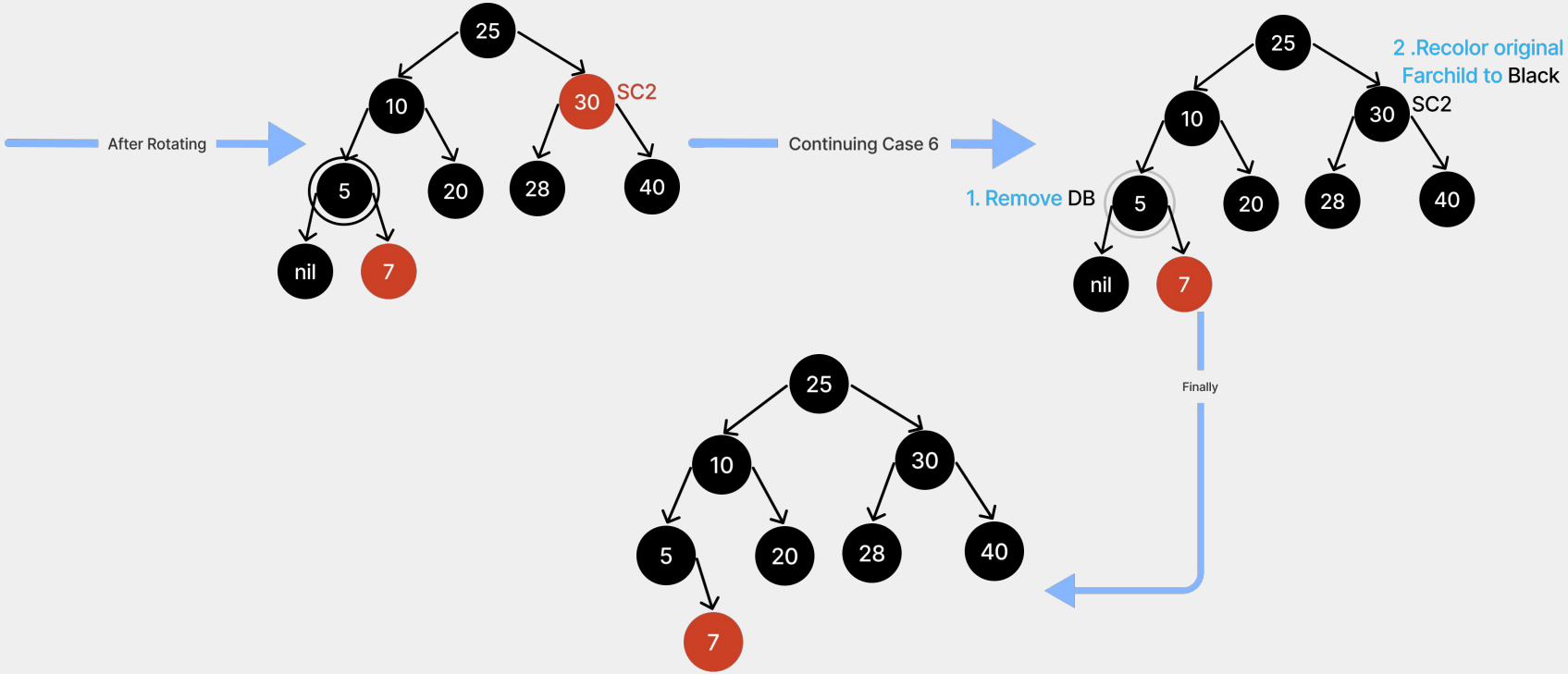
2. **Case 6: DB's sibling(S) is Black, sibling's child which is far from DB(SC2) is red**

a. Swap the color of P and S i.e. Parent and Sibling.
b. Rotate the Parent towards **DB**.
c. Remove **DB**.
d. Change color of Red child (SC2) to black.



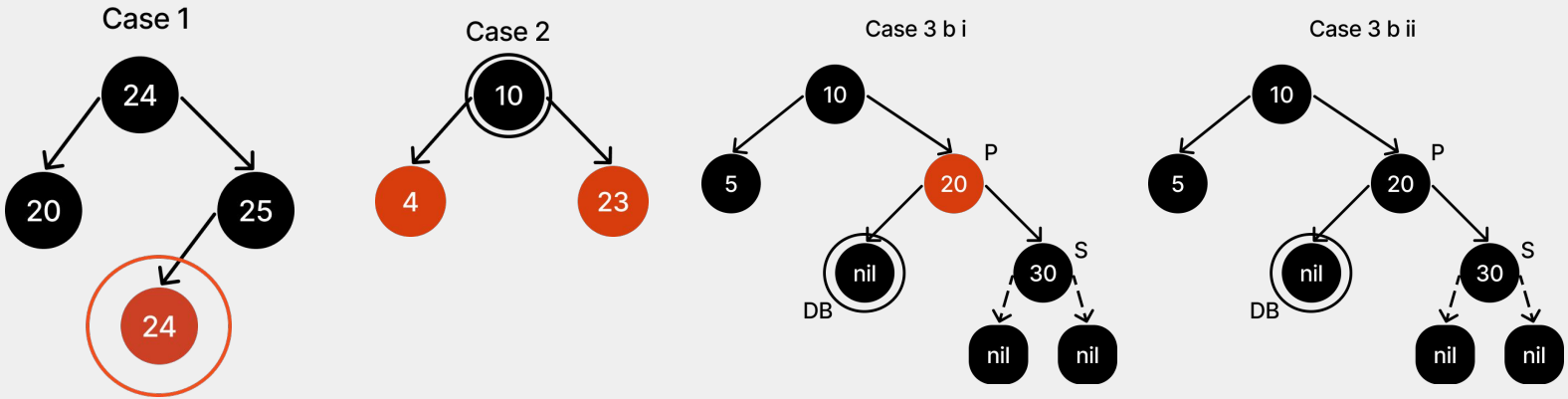
RED BLACK TREE

DELETION ON RBT: CASE 6



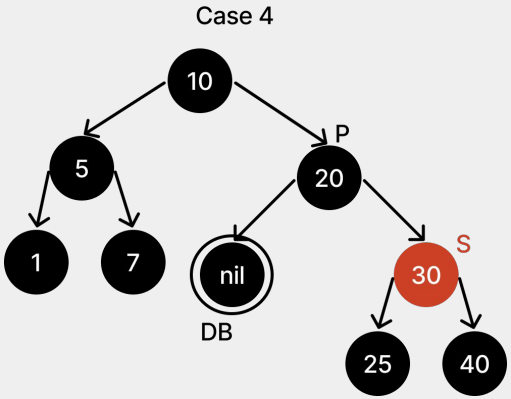
RED BLACK TREE

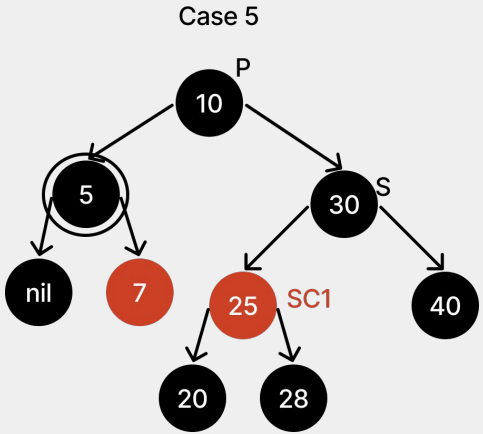
DELETION ON RBT: SUMMARY OF SCENARIOS FOR CASES

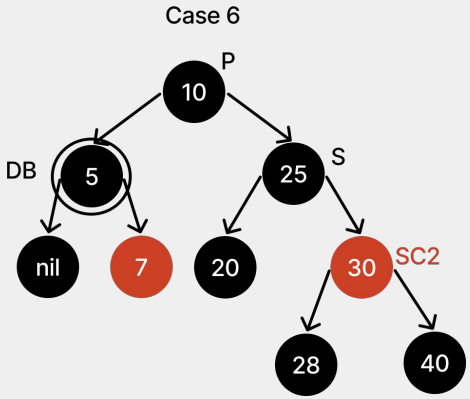


RED BLACK TREE

DELETION ON RBT: SUMMARY OF SCENARIOS FOR CASES







RED BLACK TREE

DELETION TIME AND SPACE COMPLEXITY

Time

- Finding the delete node plus the left-most right successor is proportional to the tree's height, hence it is O(log n)
- The swapping and deletion are both O(1)
- Each particular fix (rotation, for example) is O(1). In the worst-case scenario, a double-black might be transmitted up to the root. Because each rotation takes the same amount of time, this is proportional to the tree's height and so O(log n)

So, **Best= O(log n) Worst = O(log n)**

Space

The average and worst space complexity of a red-black tree is the same as that of a Binary Search Tree and is determined by the total number of nodes: O(n). Each node takes up O(1) space. As a result, if the tree has n total nodes, the space complexity is n times O(1), which is O(n)

So, **Space complexity = O(n)**

RED BLACK TREE

04
EXTRA

RED BLACK TREE

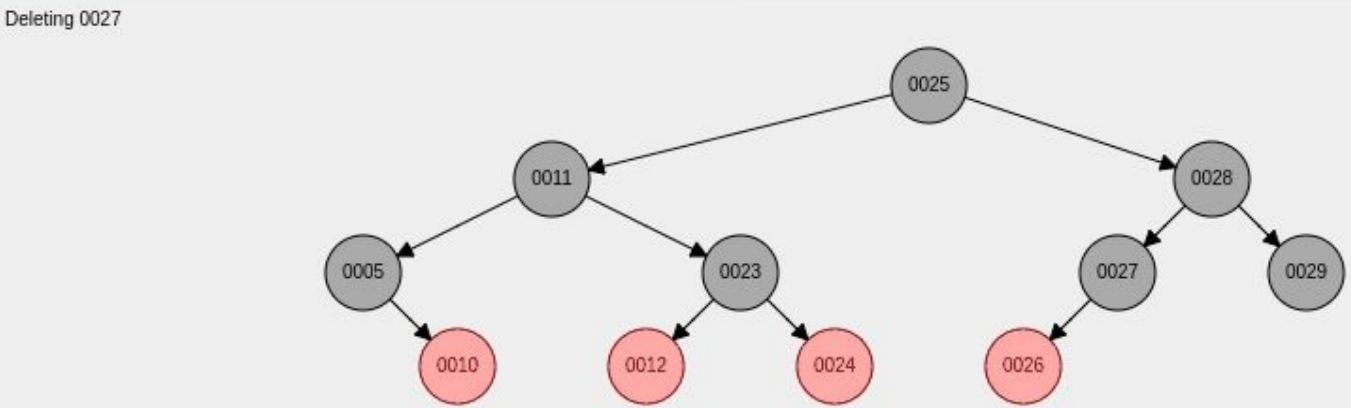
EXTRA: PLAYGROUND

Scan QR Code to go to Red Black Tree visualizer

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

RED BLACK TREE

EXTRA: PLAYGROUND



RED BLACK TREE

REFERENCES

"5.18 Red Black Tree Deletion | Data Structure." *YouTube*, uploaded by Jenny's lectures CS/IT NET&JRF, 10 Nov. 2019, www.youtube.com/watch?v=w5cvkTXY0vQ.

"Red-Black Tree." *Programiz*, www.programiz.com/dsa/red-black-tree. Accessed 10 Feb. 2022.

Singh, Harshita. "Time and Space Complexity Analysis of Red Black Tree." *OpenGenus IQ: Computing Expertise & Legacy*, 28 Dec. 2021, iq.opengenus.org/time-and-space-complexity-of-red-black-tree.

"Unit 2: Red Black Tree", https://www.lkouniv.ac.in/site/writereaddata/siteContent/20200406191957998 1anshu_singh_engg_Red_black_tree.pdf

THANK YOU !