Red Black Tree

Joy Mukherjee

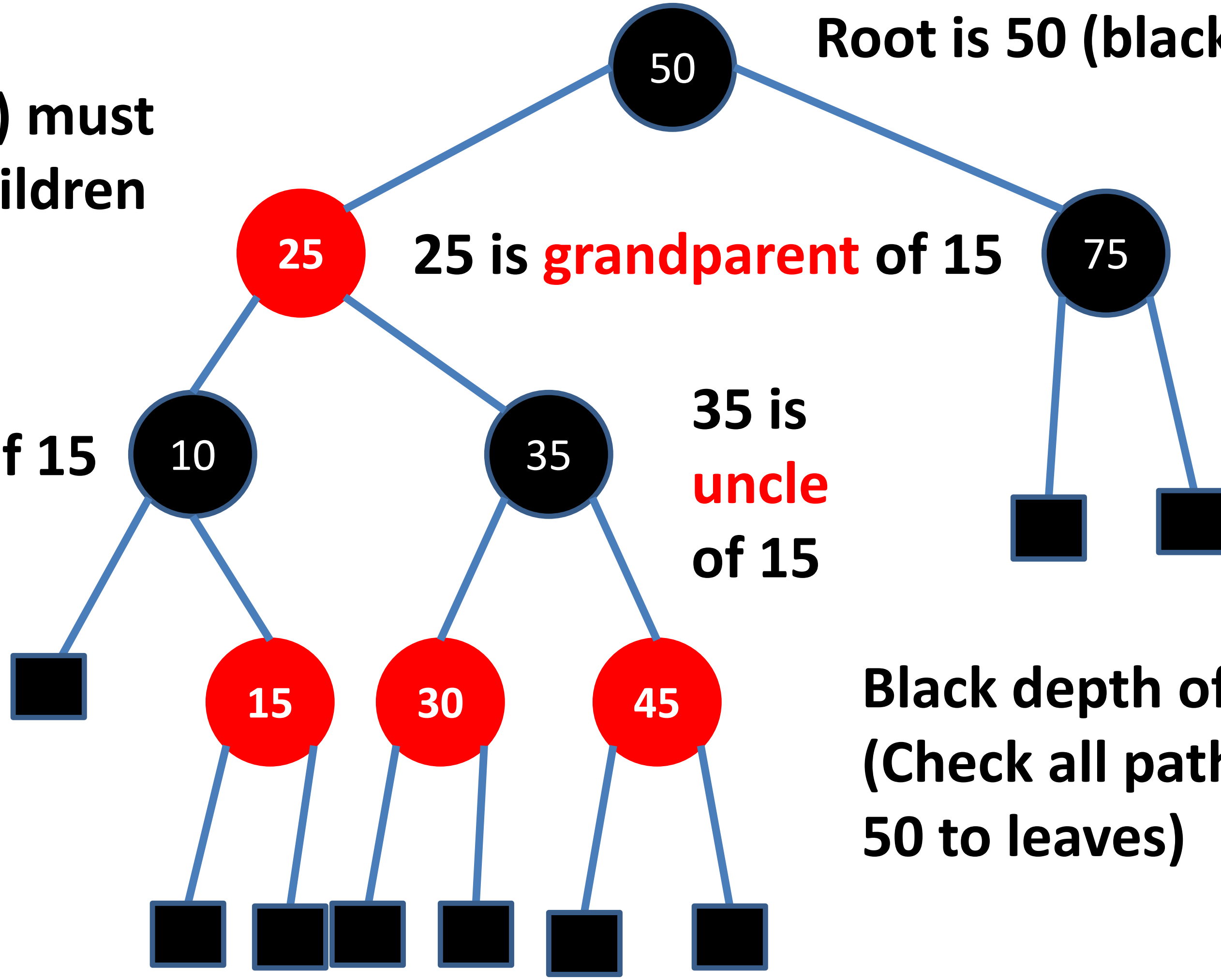
Binary Search Tree

1. It is a binary tree. It may or may not be empty.
2. The root has a key.
3. The keys (if any) in the left subtree are smaller than the key in the root.
4. The keys (if any) in the right subtree are larger than the key in the root.
5. The left subtree and right subtree are also BSTs.

Red-Black Tree

1. It is a Binary Search Tree
2. Every node is either red or black
3. Root node is always black
4. Red node always has black children
5. For each node, black depth is same, i.e., every path from a given node to NULLs contains the same number of black nodes
   - Every path from root to NULL has same number of black nodes.

Example: Red-Black Tree



Root is 50 (black)

Red node (25) must have black children (10 and 35)

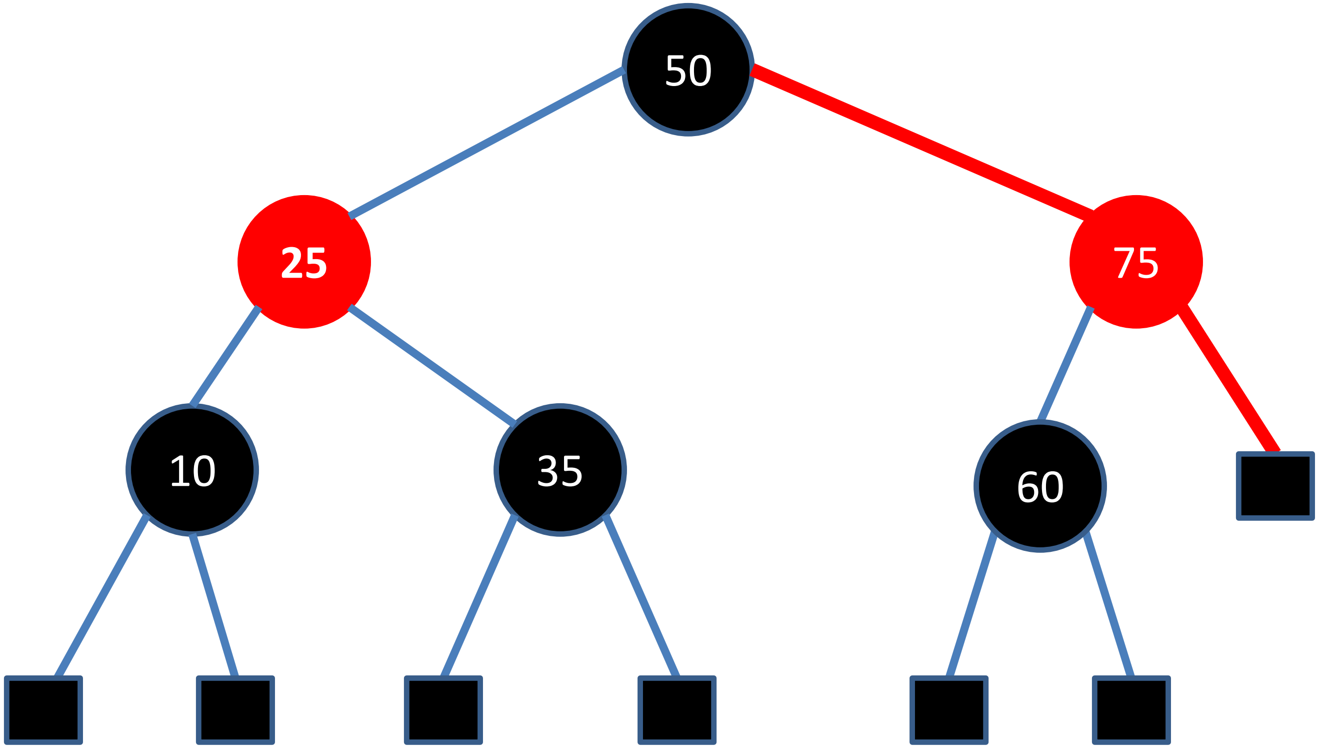
25 is grandparent of 15

10 is parent of 15

35 is uncle of 15

Black depth of 50 is 2 (Check all paths from 50 to leaves)

Not a Red-Black Tree



Black Depth is not same in all Root-NULL paths

Insertion in Red-Black Tree

1. Key is inserted as was done in BST. Make it red.
2. If Key is the root node, then make it black and stop.
3. If parent of key is black, then stop.
4. While parent of key is red, Property 4 violated.
   Uncle is red
   I. Grandparent is the root → Recoloring of uncle and parent to black, and stop.
   II. Grandparent is not the root → Recoloring of uncle and parent to black and grandparent to red. Now consider grandparent as key and Goto 4.

Insertion in Red-Black Tree

Uncle is black/missing

III. Key is the left child of its parent and its parent is the left child of its grandparent

→ Right Rotation & Recoloring; Stop.

IV. Key is the right child of its parent and its parent is the left child of its grandparent

→ Left Rotation, then Right Rotation & Recoloring; Stop.

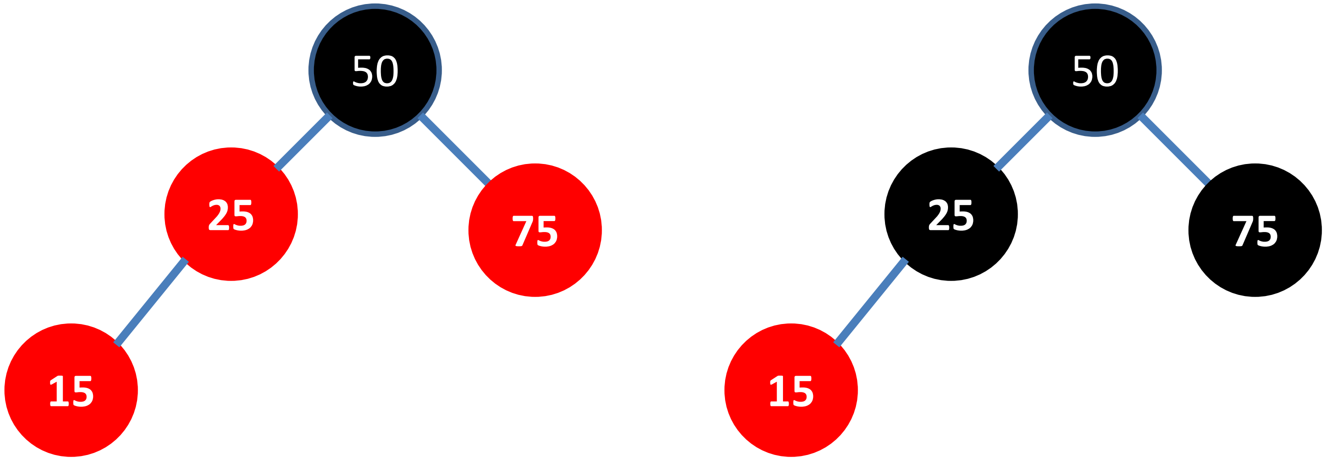
V. Key is the right child of its parent and its parent is the right child of its grandparent

→ Left Rotation & Recoloring; Stop.

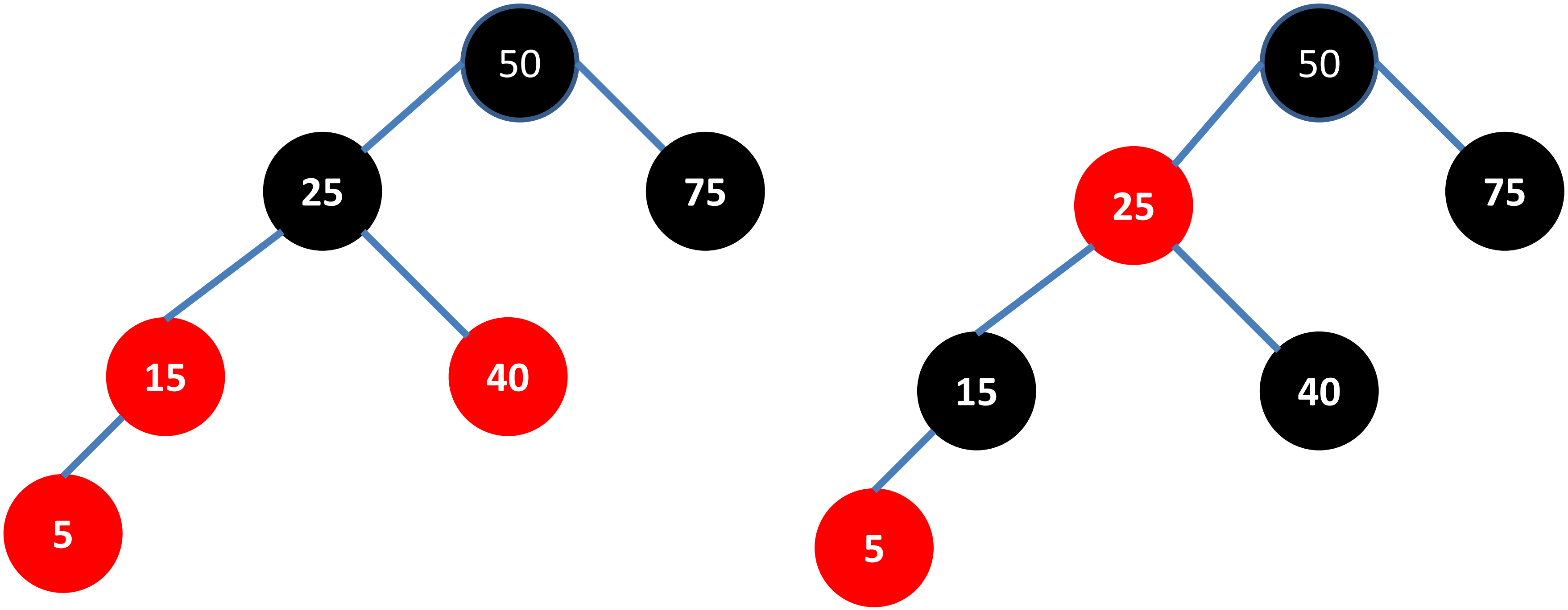
VI. Key is the left child of its parent and its parent is the right child of its grandparent

→ Right Rotation, then Left Rotation & Recoloring; Stop.

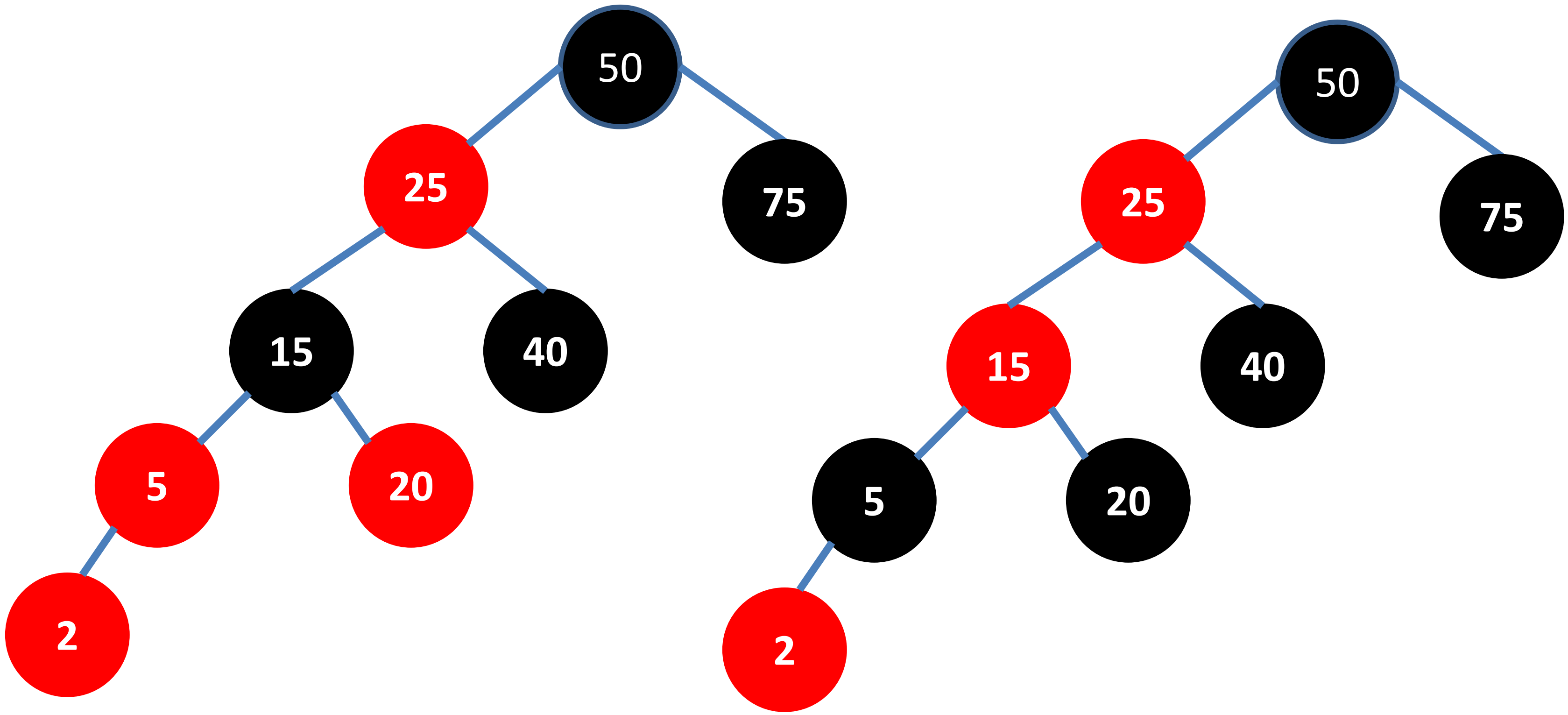
Case I: Insert 15 & Recoloring



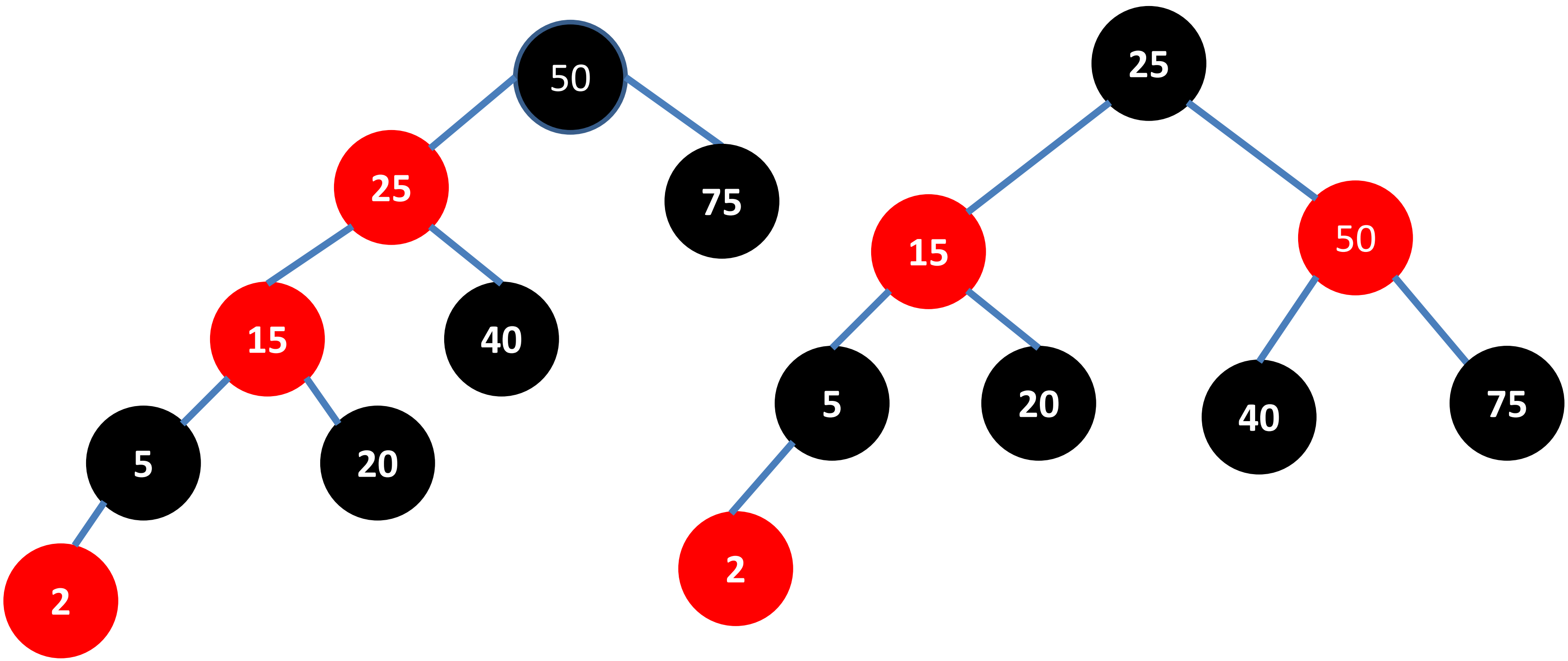
Case II: Insert 5 & Recoloring



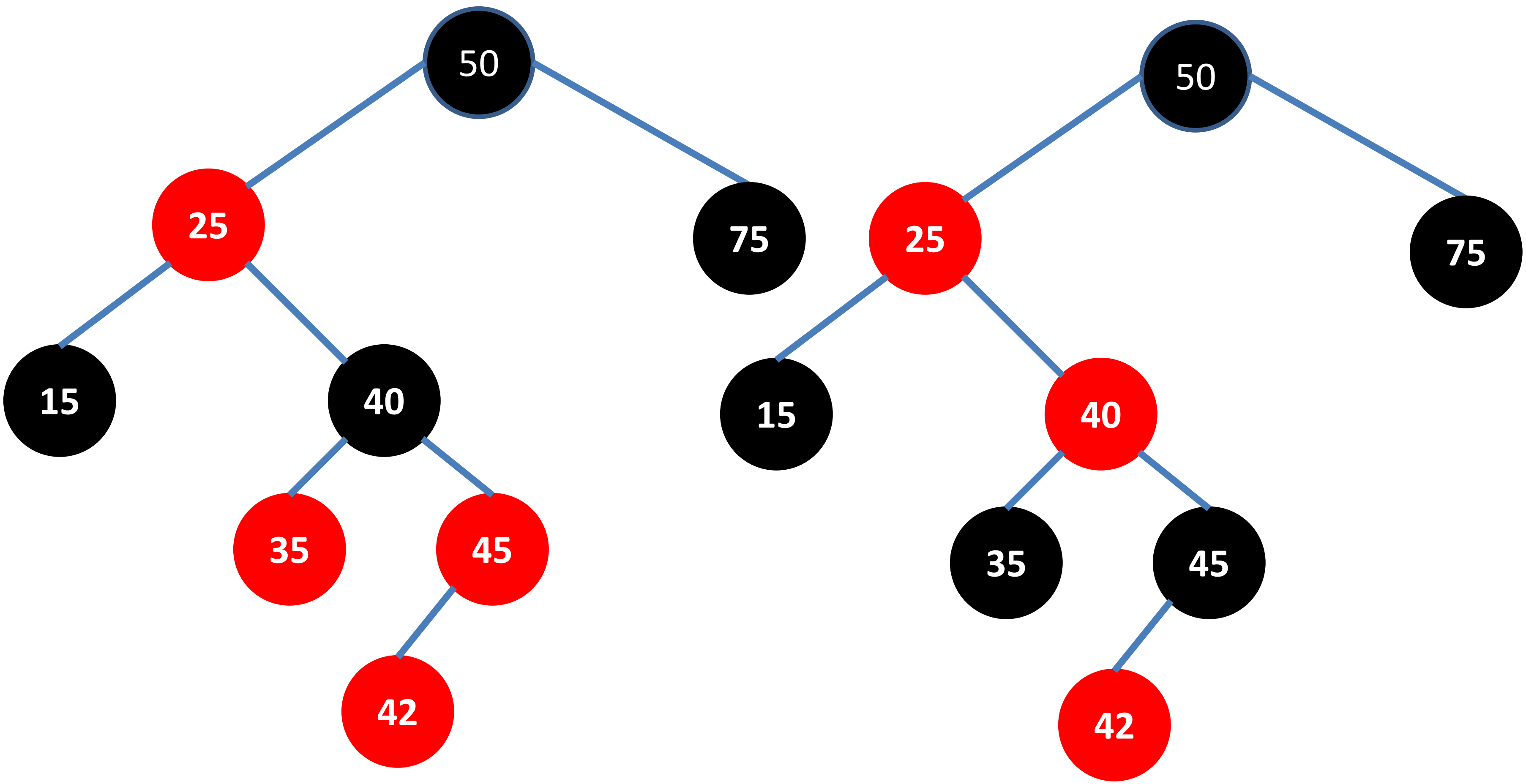
Case II: Insert 2 & Recoloring



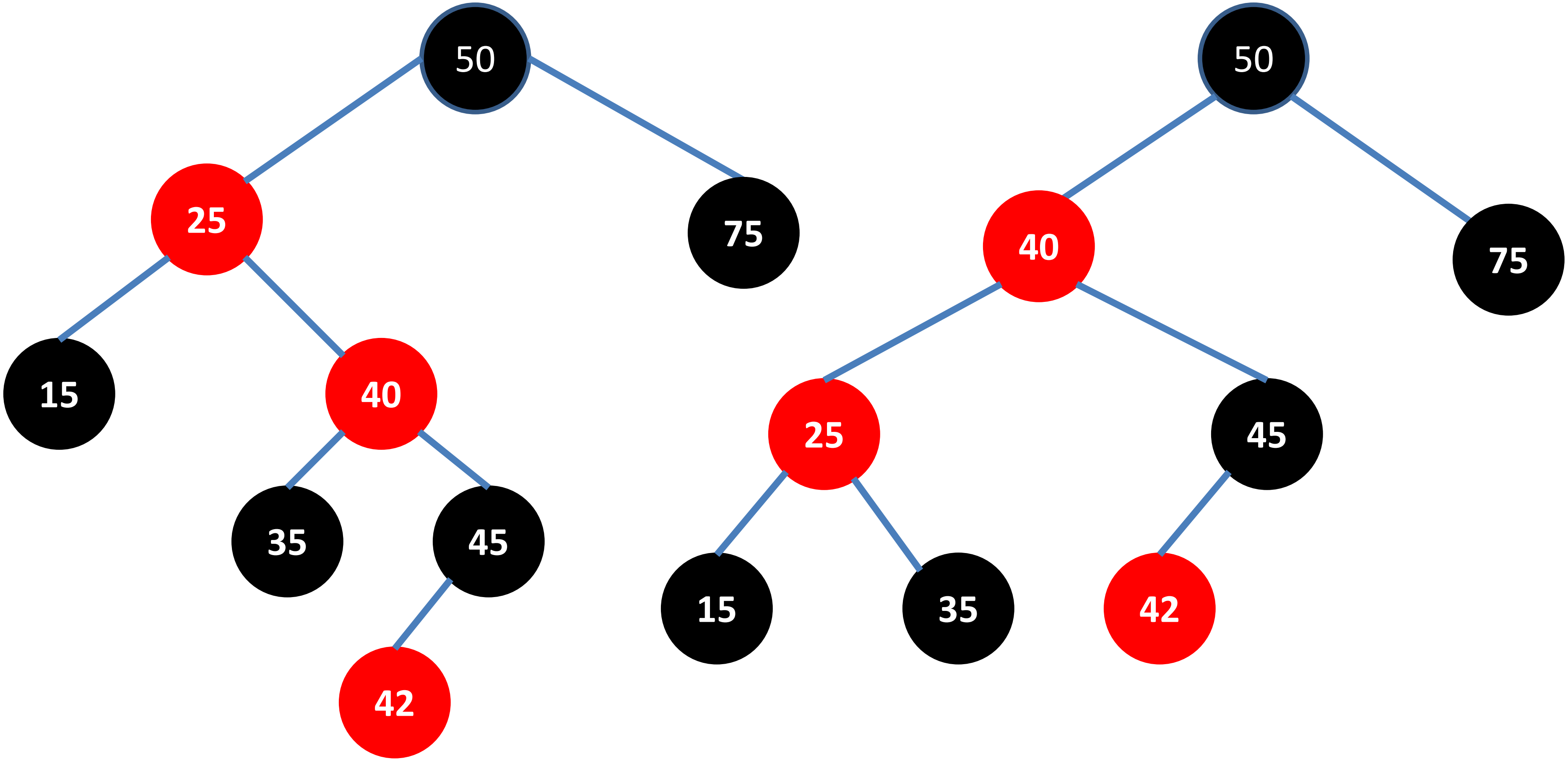
Case III: Right Rotation & Recoloring



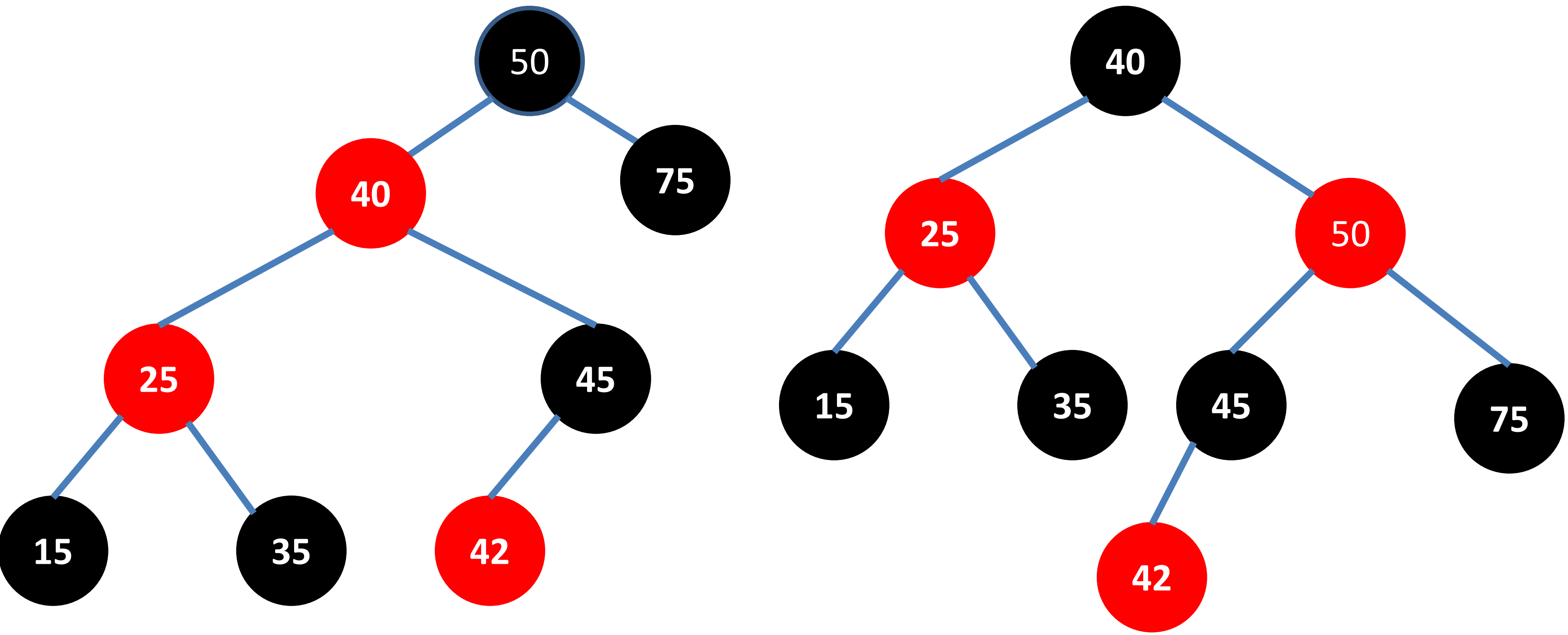
Case II: Insert 42 & Recoloring



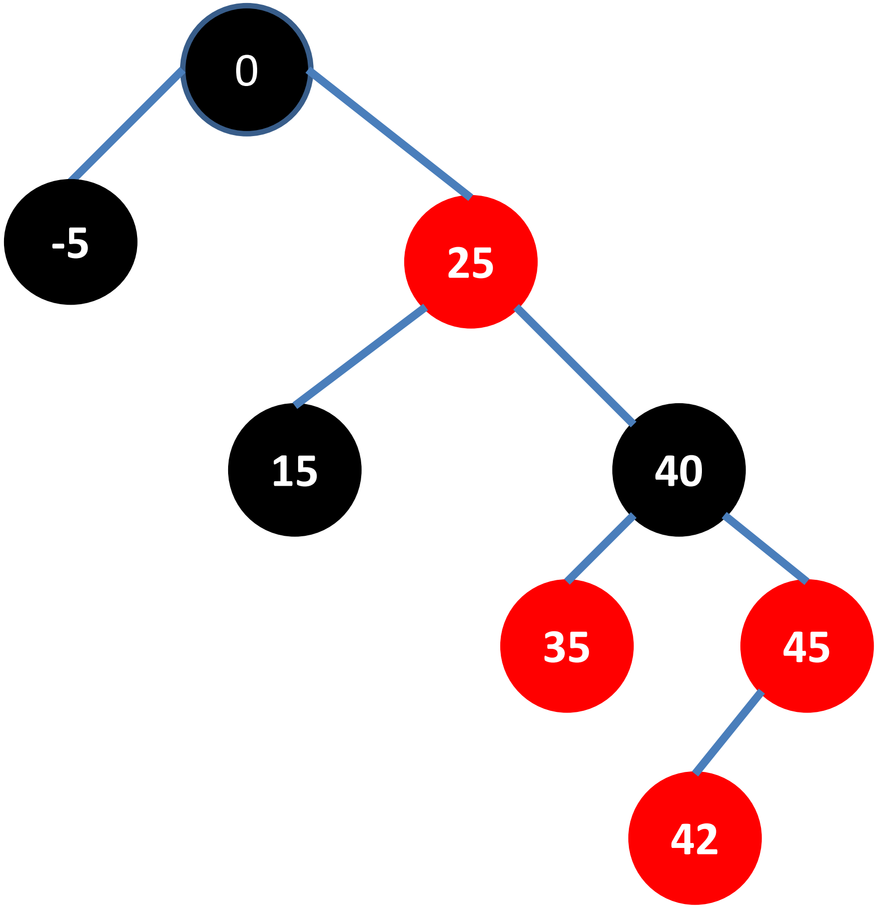
Case IV: Left Rotation

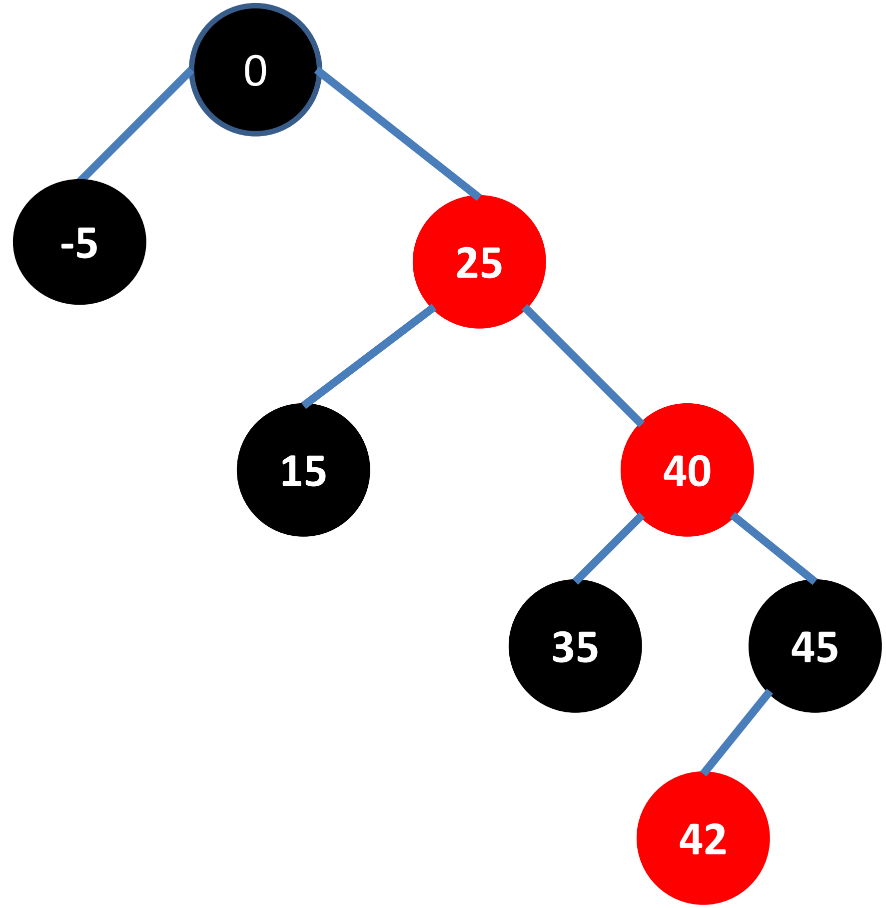


Case IV: Right Rotation & Recoloring

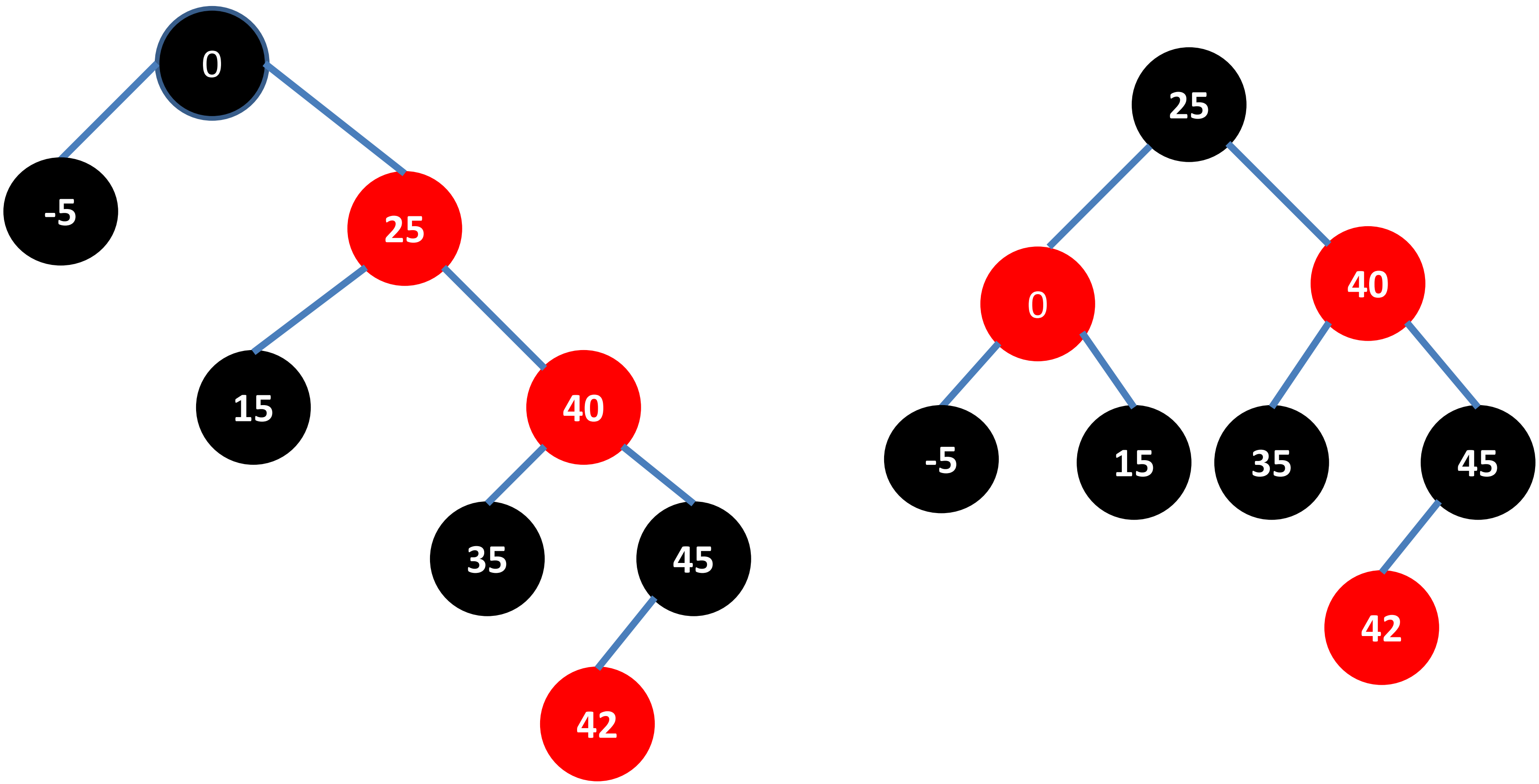


Case II: Insert 42 & Recoloring

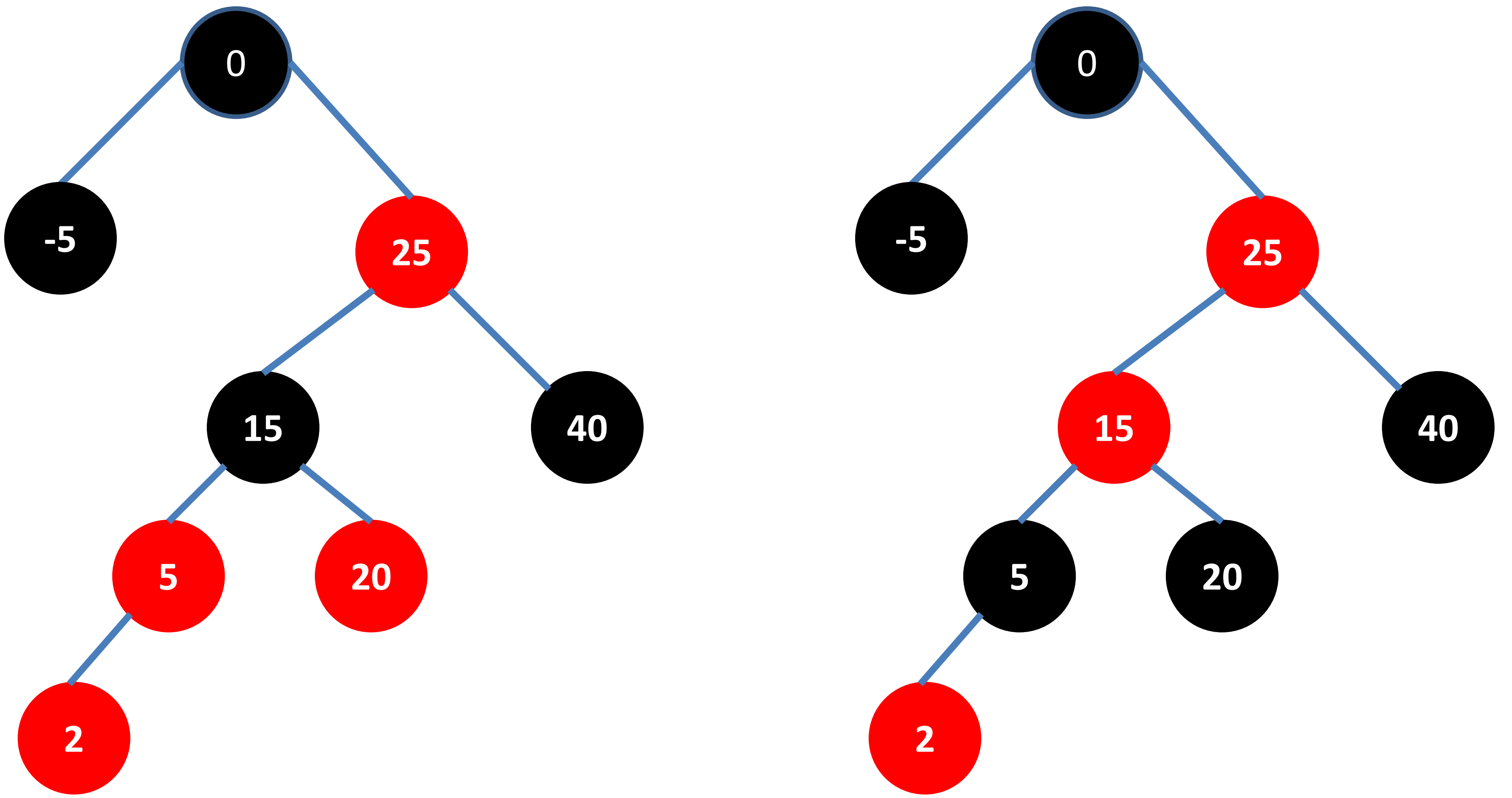




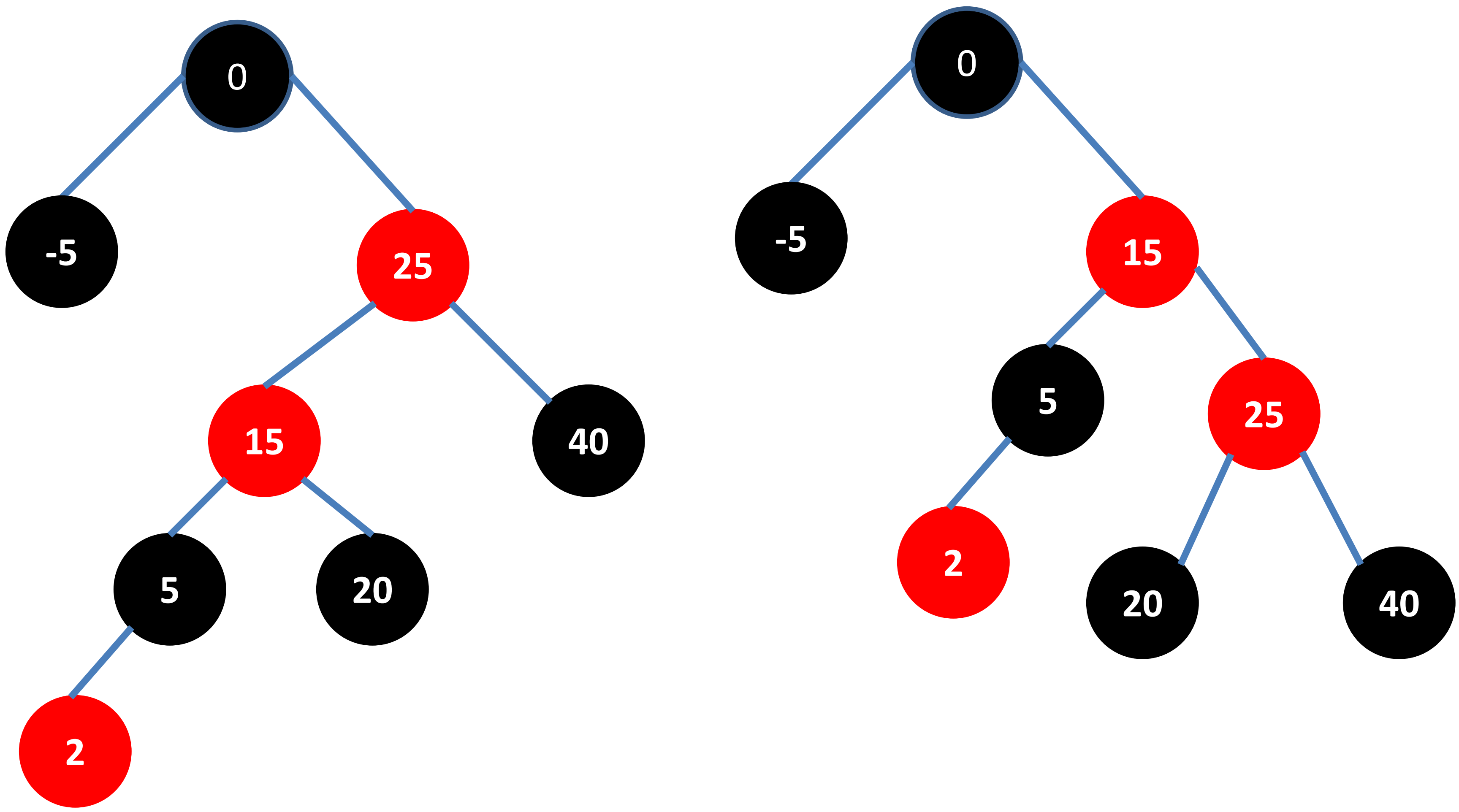
Case V: Left Rotation & Recoloring



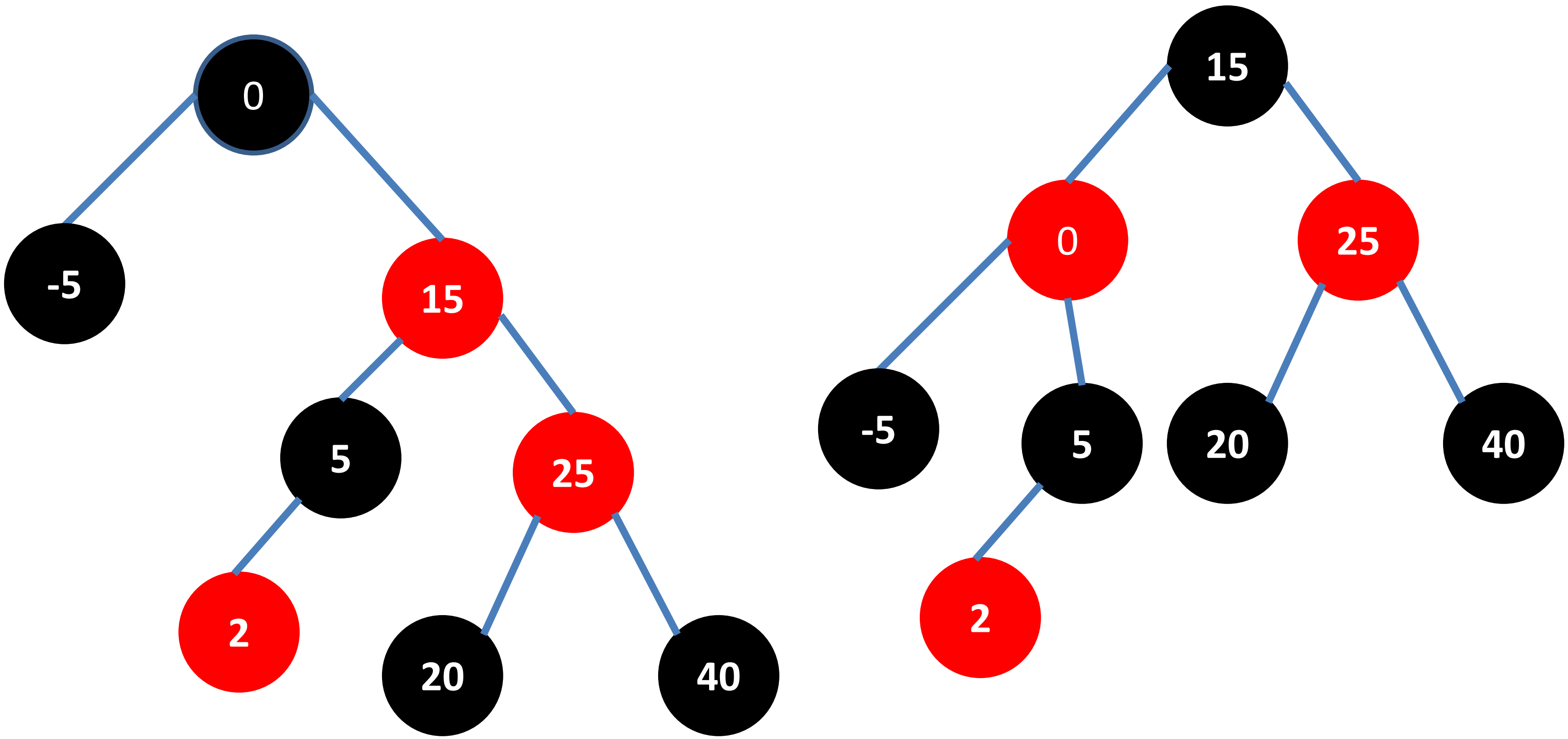
Case II: Insert 2 & Recoloring



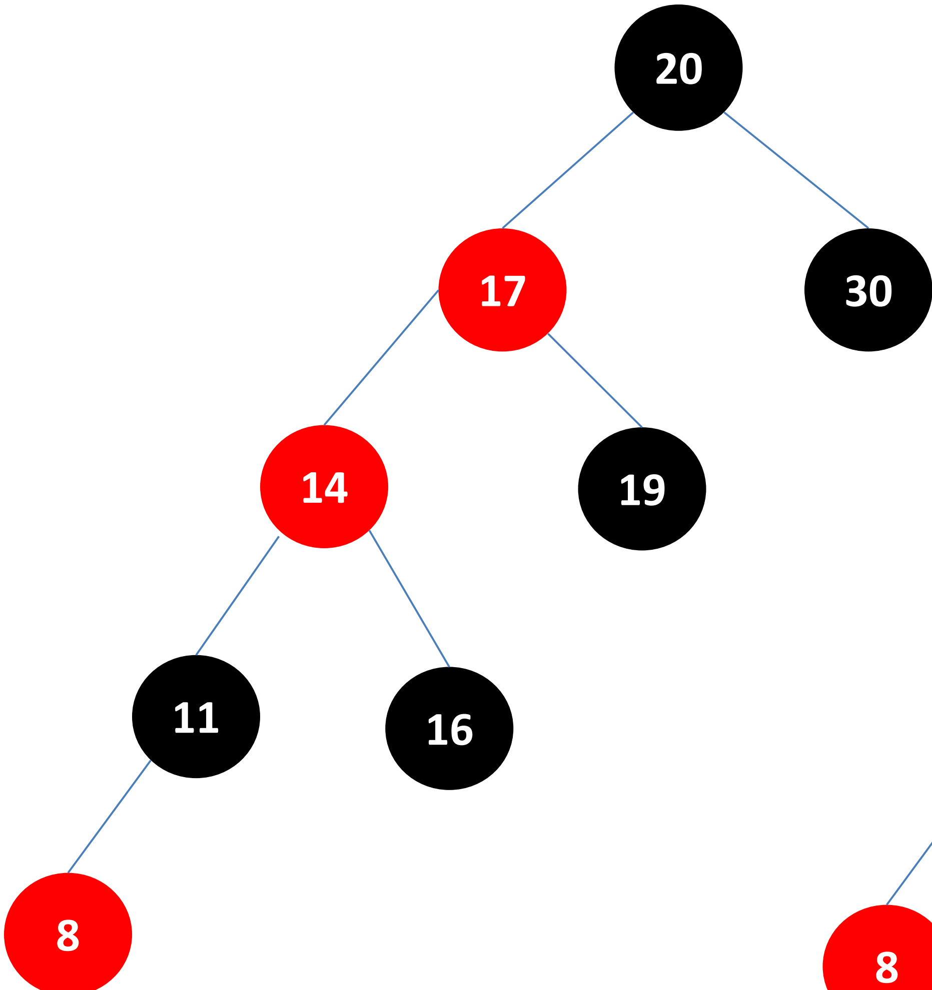
Case VI: Right Rotation

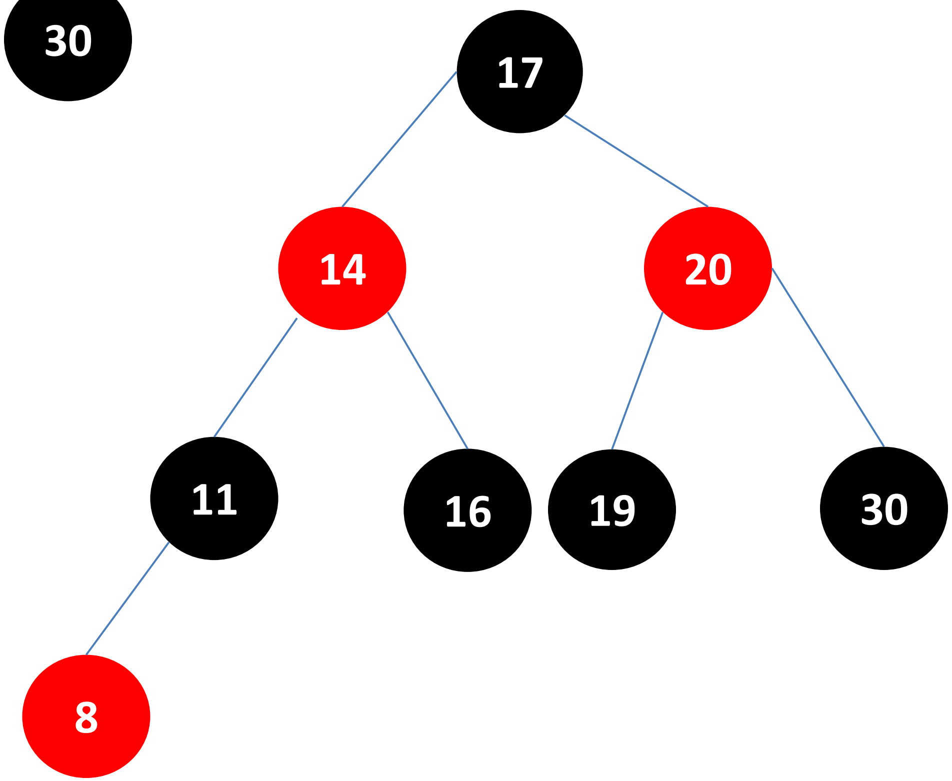


Case VI: Left Rotation & Recoloring

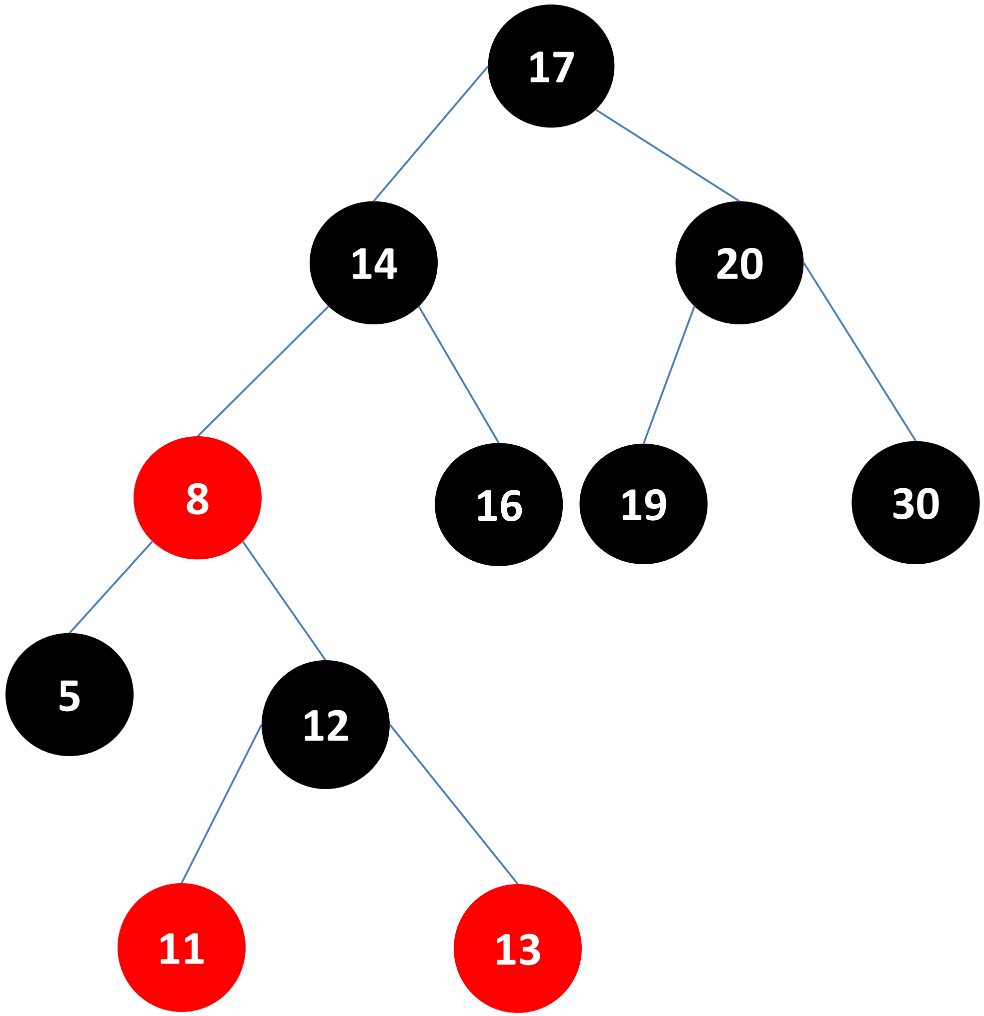


20 17 30 14 19 11 16 8

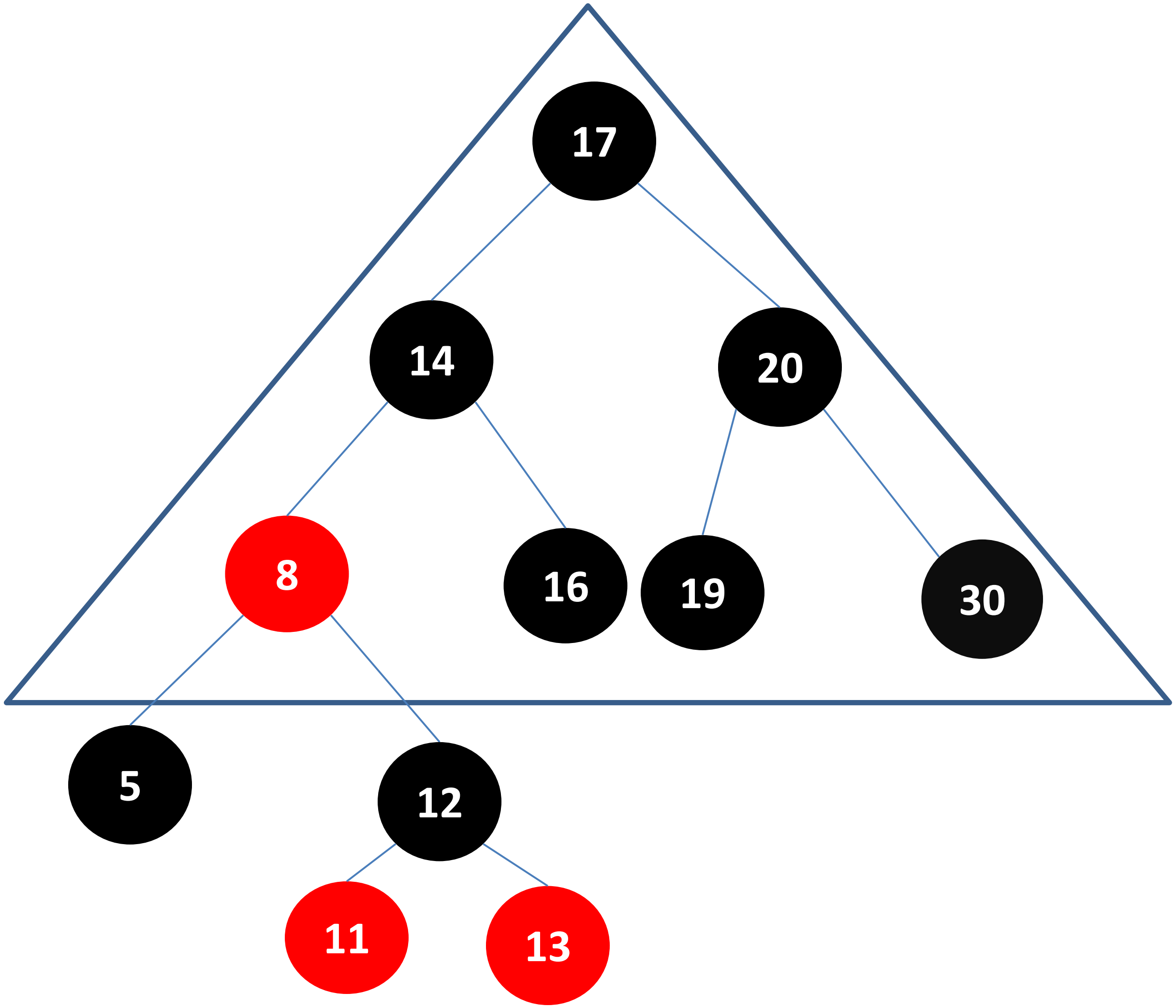




20 17 30 14 19 11 16 8 5 13 12



20 17 30 14 19 11 16 8 5 13 12



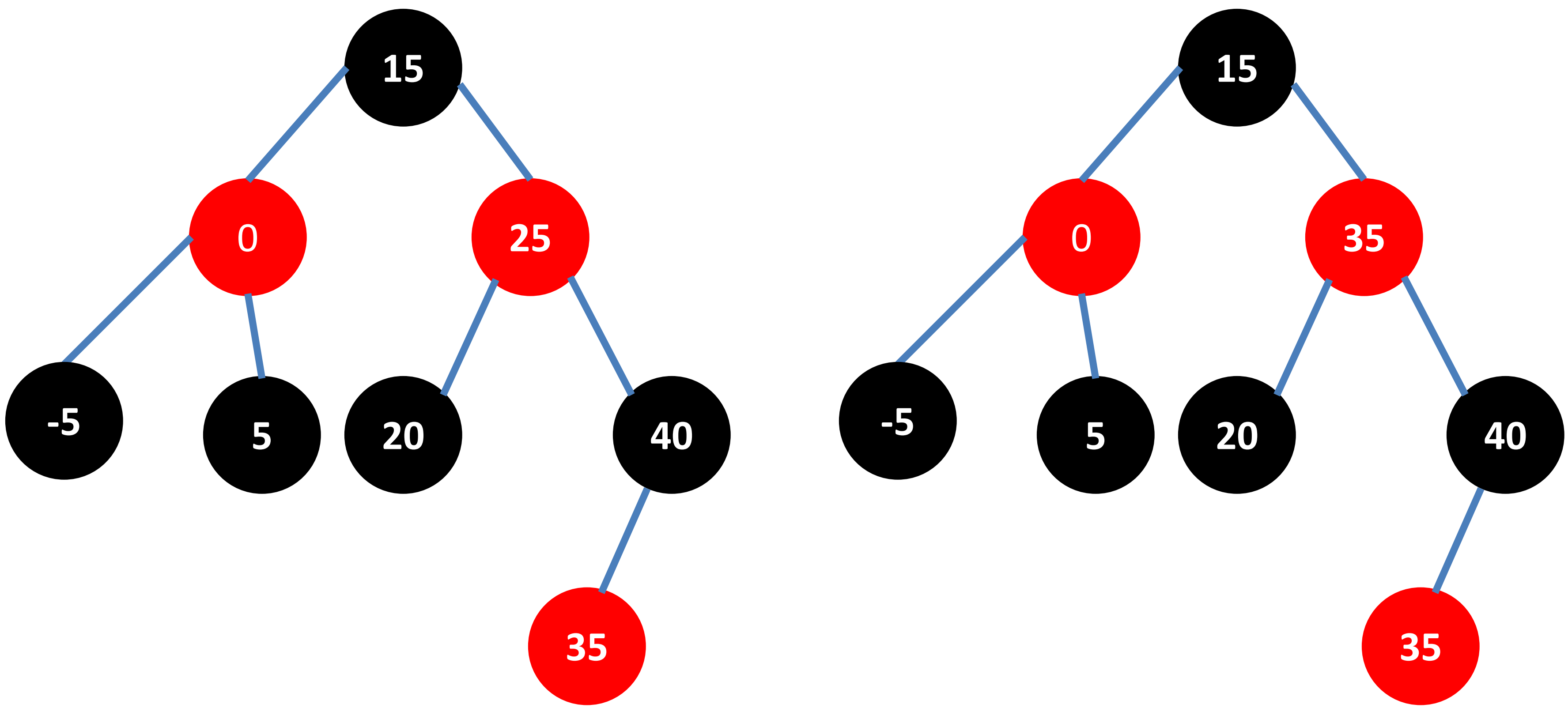
Maximum Depth of a Red-Black Tree

- The black depth is same.
- The number of black nodes in all root-to-NULL paths is same.
- Consider the shortest root-to-NULL path of length k edges, i.e., k+1 nodes.
- All the levels from 0 to k in the red-black tree is full, i.e., it is a complete binary tree.
- The number of nodes in the complete binary tree is $n = 2^{k+1} – 1$.
- The black depth of the tree = $k+1 = \log_2(n+1)$
- The maximum depth of the tree = $2. \log_2(n+1)$ [In the longest path, red and black nodes may interleave in the worst case, since red nodes have black children]
- If N is the total number of nodes in the red-black tree, then $n \leq N$
- The maximum depth of the tree = $2. \log_2(n+1) \leq 2. \log_2(N+1)$

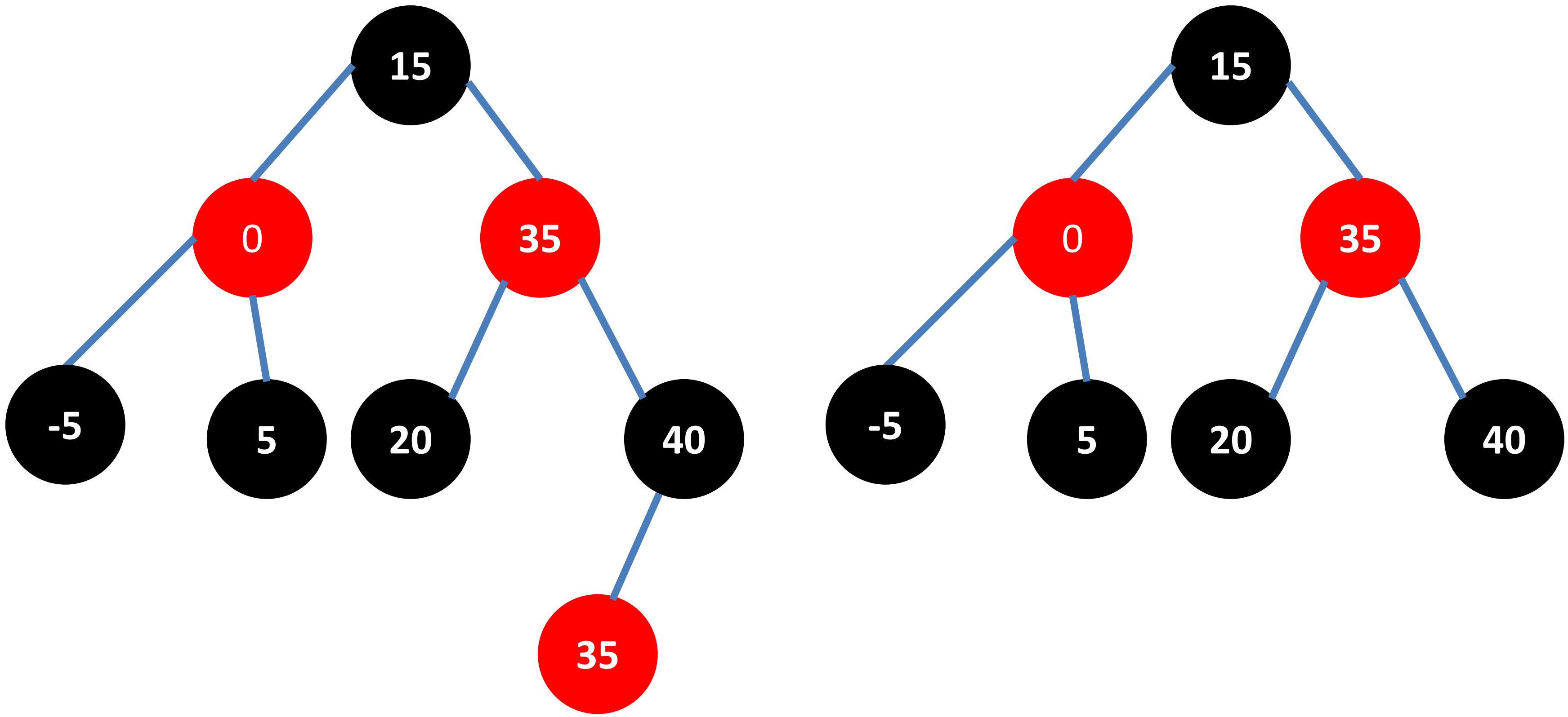
Deletion in Red-Black Tree

1. Like BST, if a key to be deleted has 2 children, then convert to a case where the key to be deleted is a leaf or has a single child.
2. If the node to be deleted is red, then delete and stop.
3. If the node to be deleted has a red child, then delete the node, color the child black, and stop.
4. If the node to be deleted is black, then there are six possible cases.

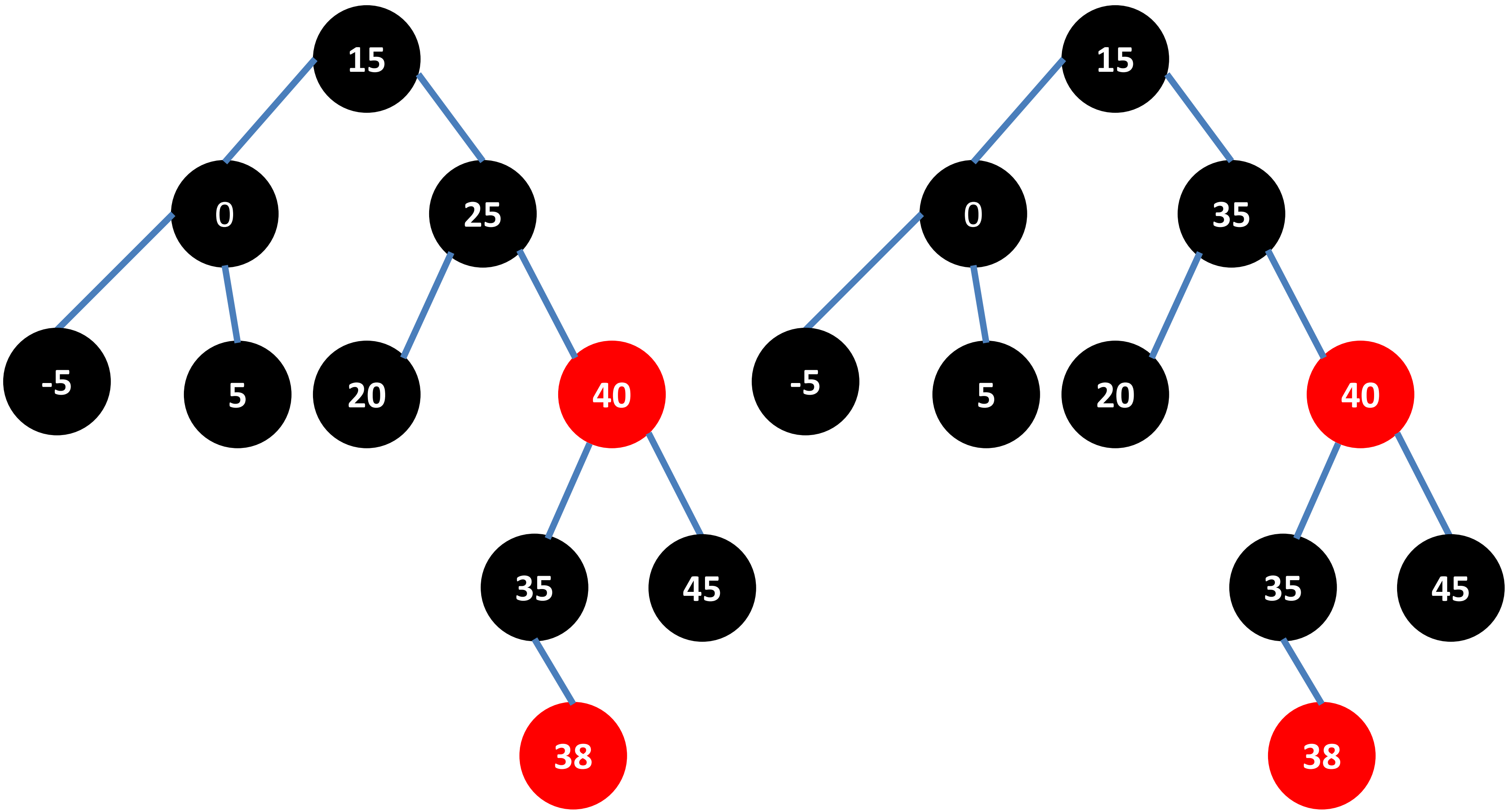
Delete 25: Replace by Inorder Successor



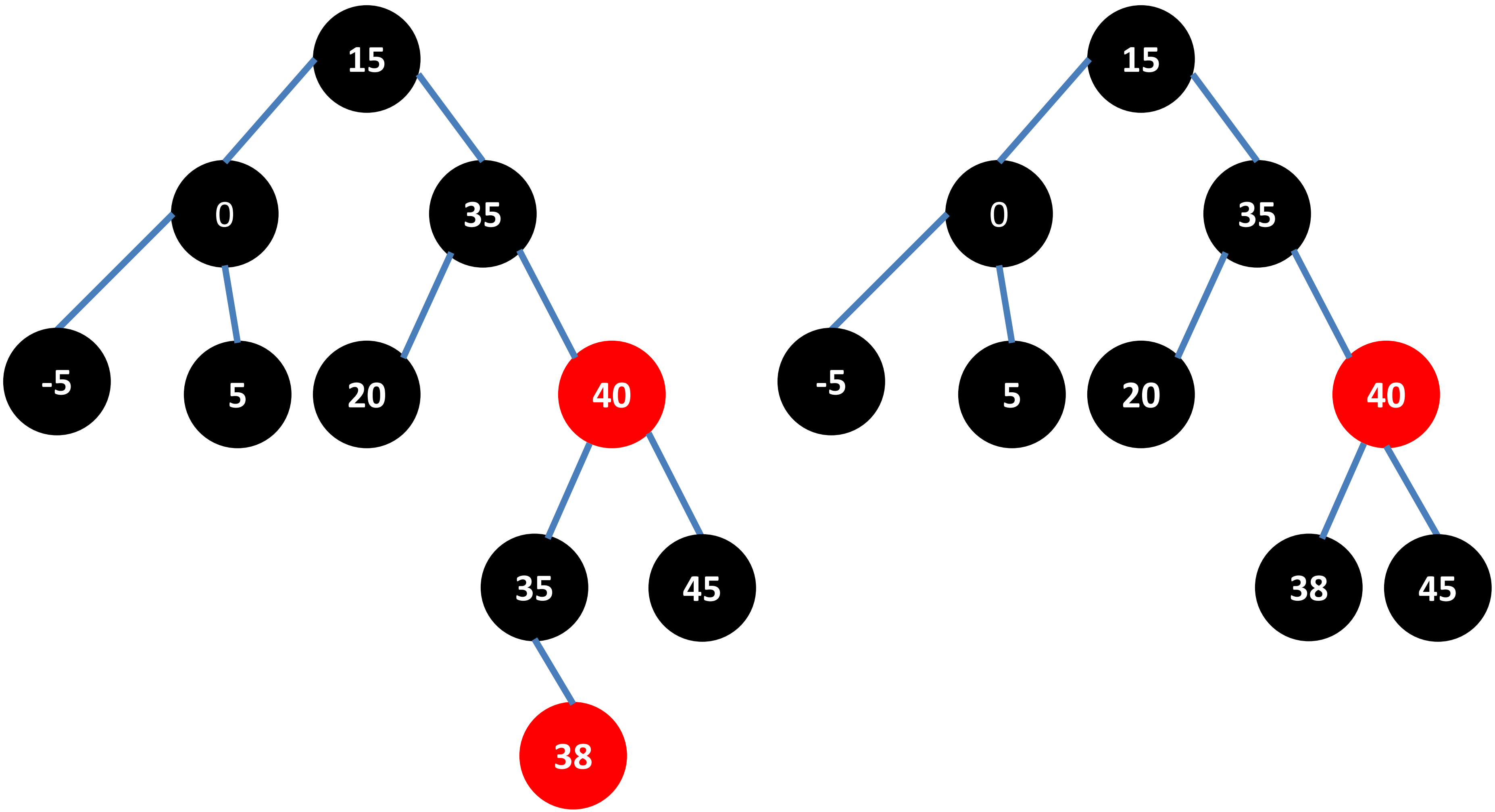
Delete 35



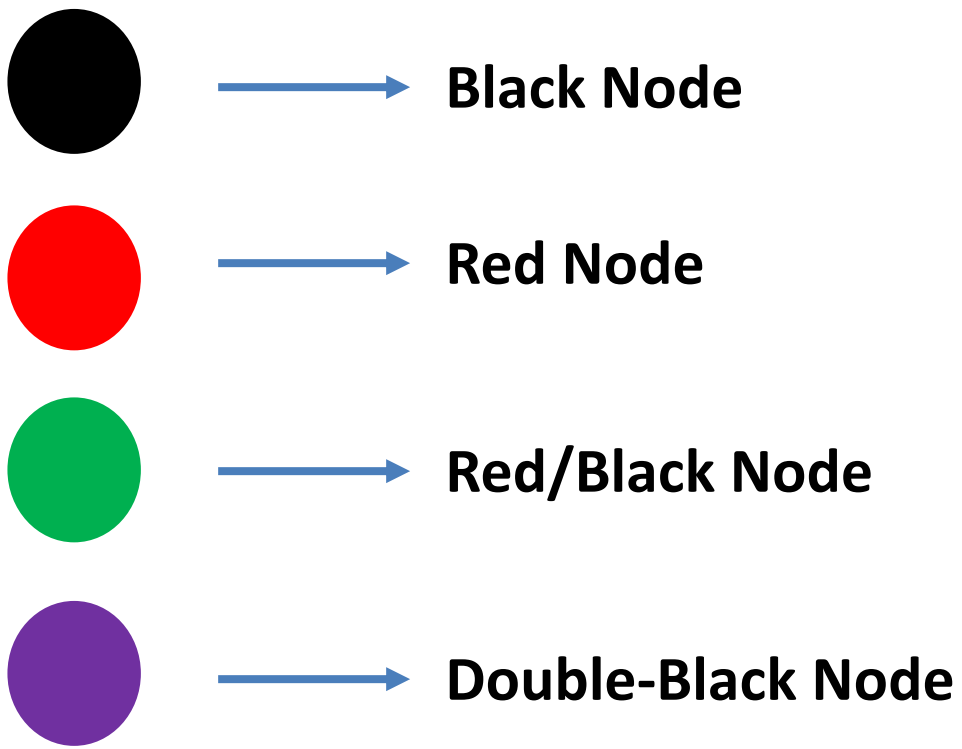
Delete 25: Replace by Inorder Successor



Delete 35

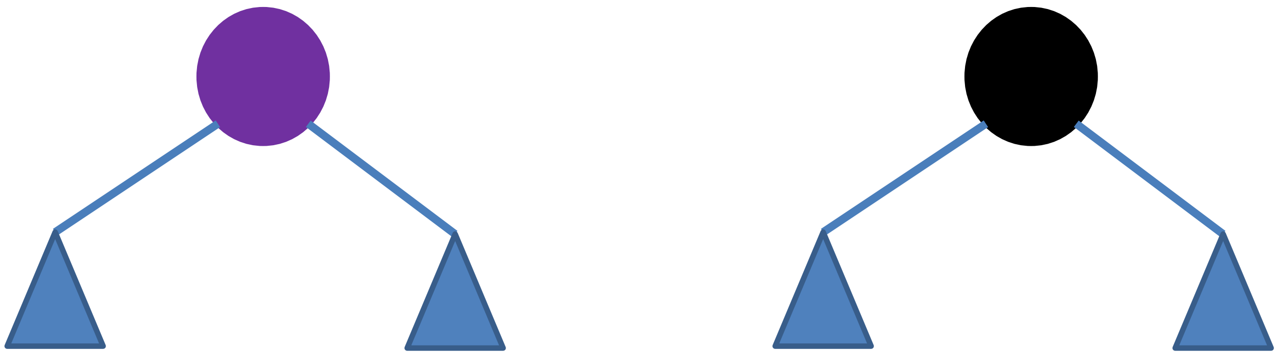


Deletion in Red-Black Tree: Six Cases



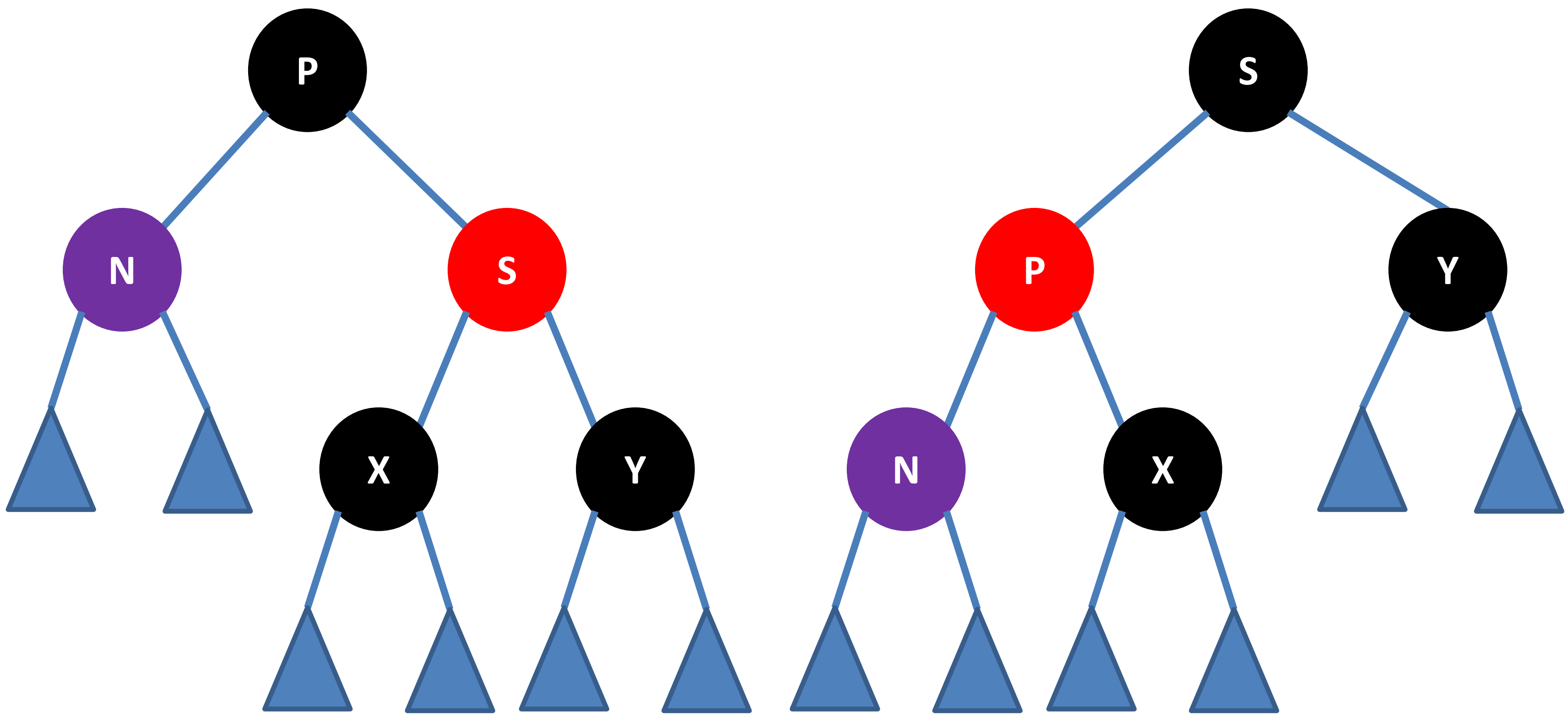
Case I: Root is double-black with two subtrees

→ Make the root black and stop.



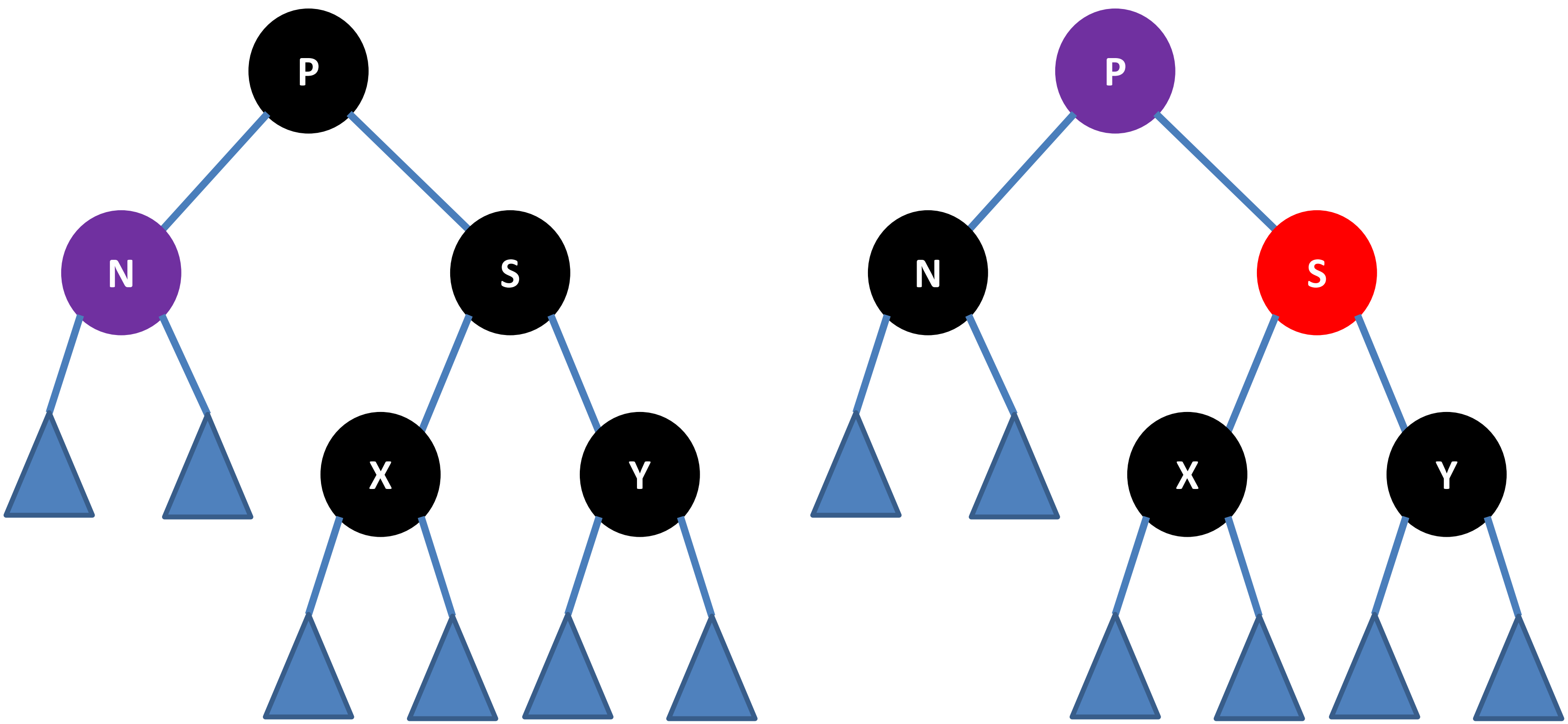
Case II: Double-black node has a black parent and a red right sibling, and the red sibling has two black children

→ Recolor parent as red and right sibling as black, rotate left around parent, check for double black cases.



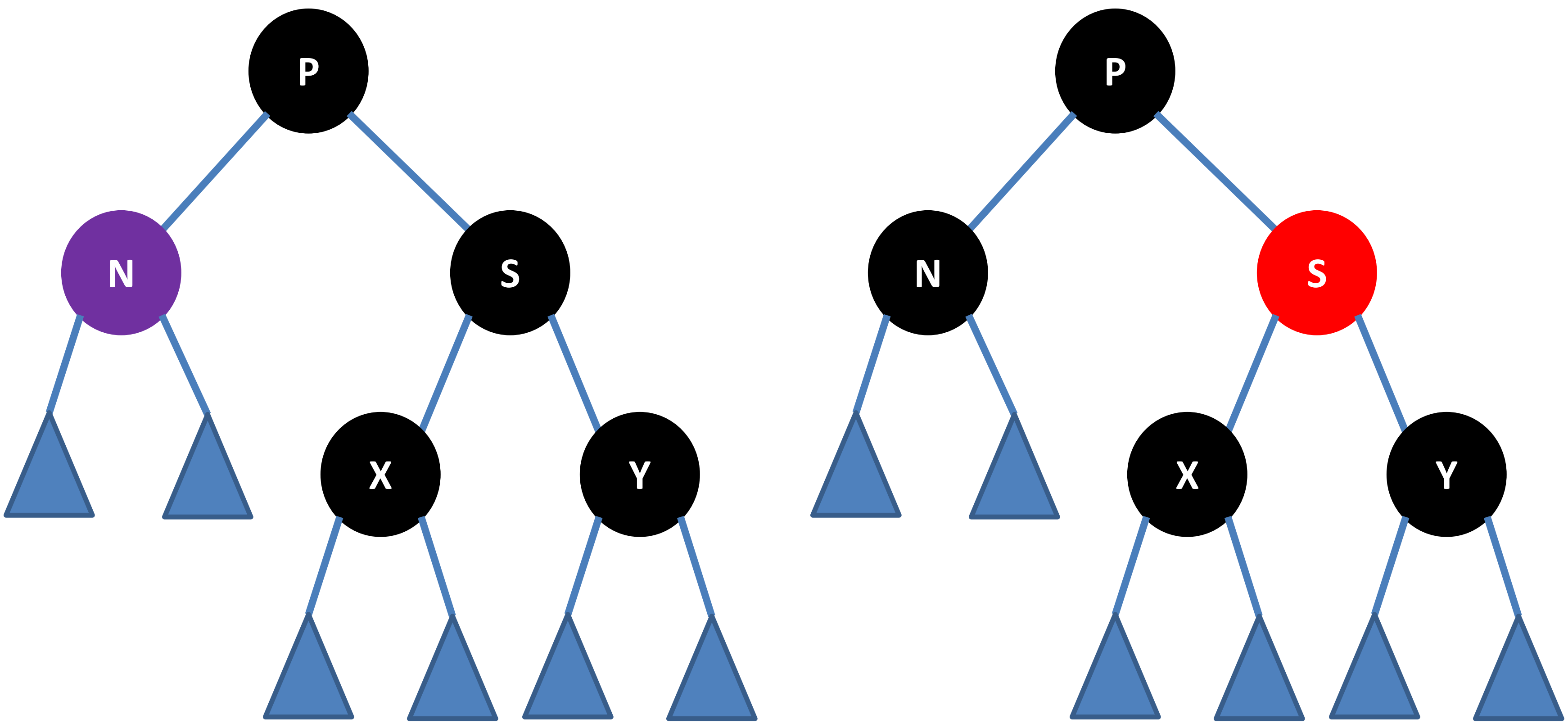
Case III: Double-black node has a black parent and a black right sibling, and the black sibling has two black children

→ Recolor parent as red and right sibling as black, rotate left around parent, check for double-black cases.



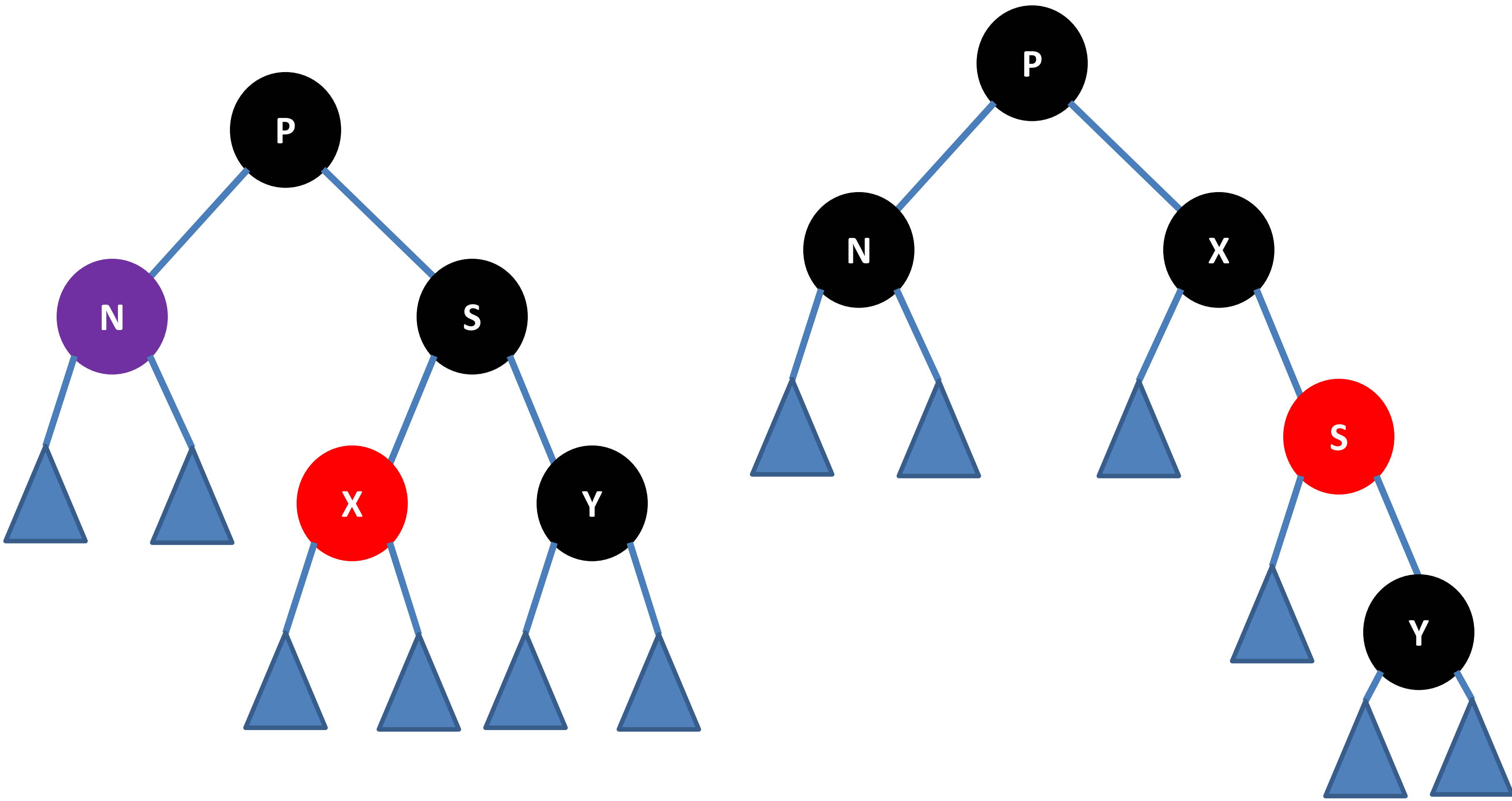
Case IV: Double-black node has a red parent and a black right sibling, and the black sibling has two black children

→ Recolor parent as red and right sibling as black, rotate left around parent, and stop.



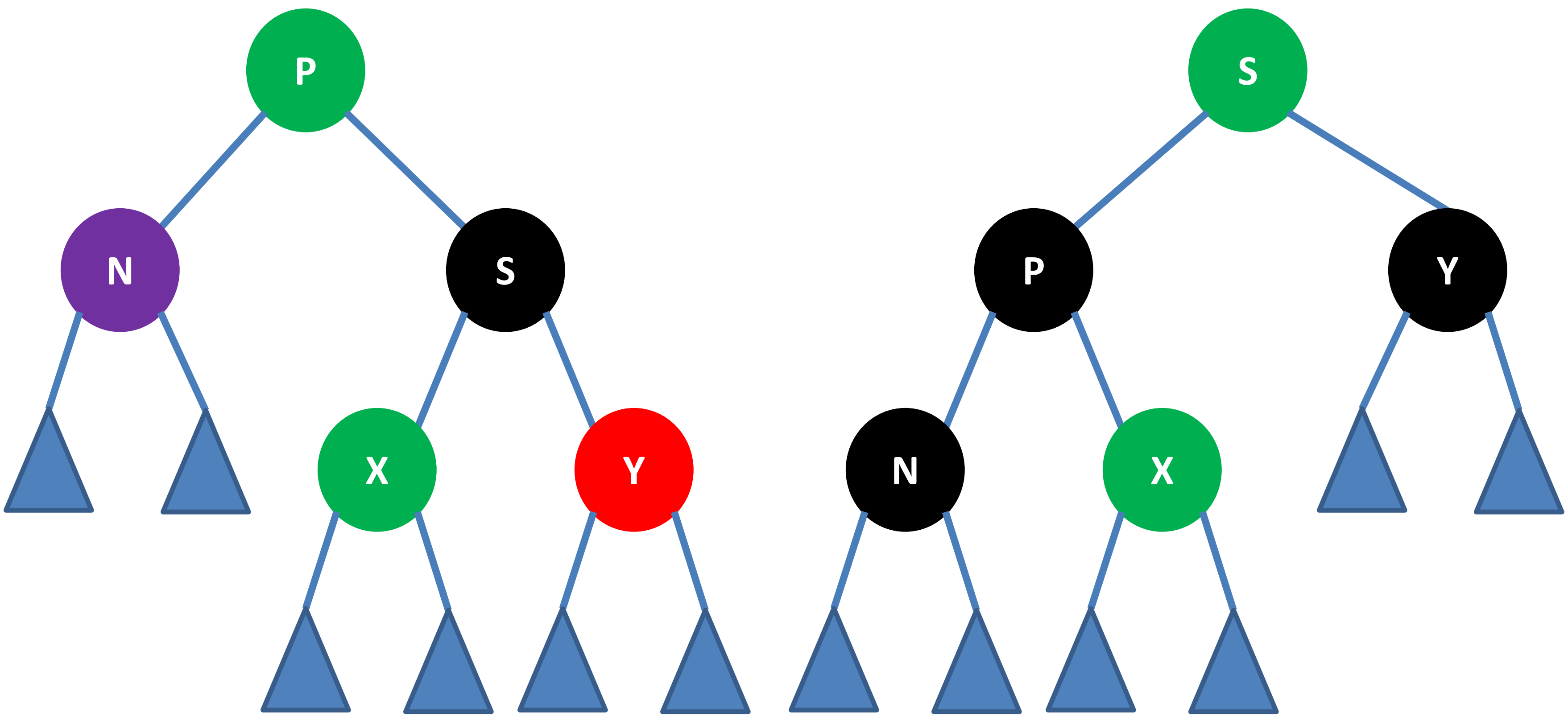
Case V: Double-black node has a black parent and a black right sibling, and the right sibling has a red left child and a black right child

→ Recolor right sibling as red and left child of right sibling as black, rotate right around right sibling, check for double-black cases.

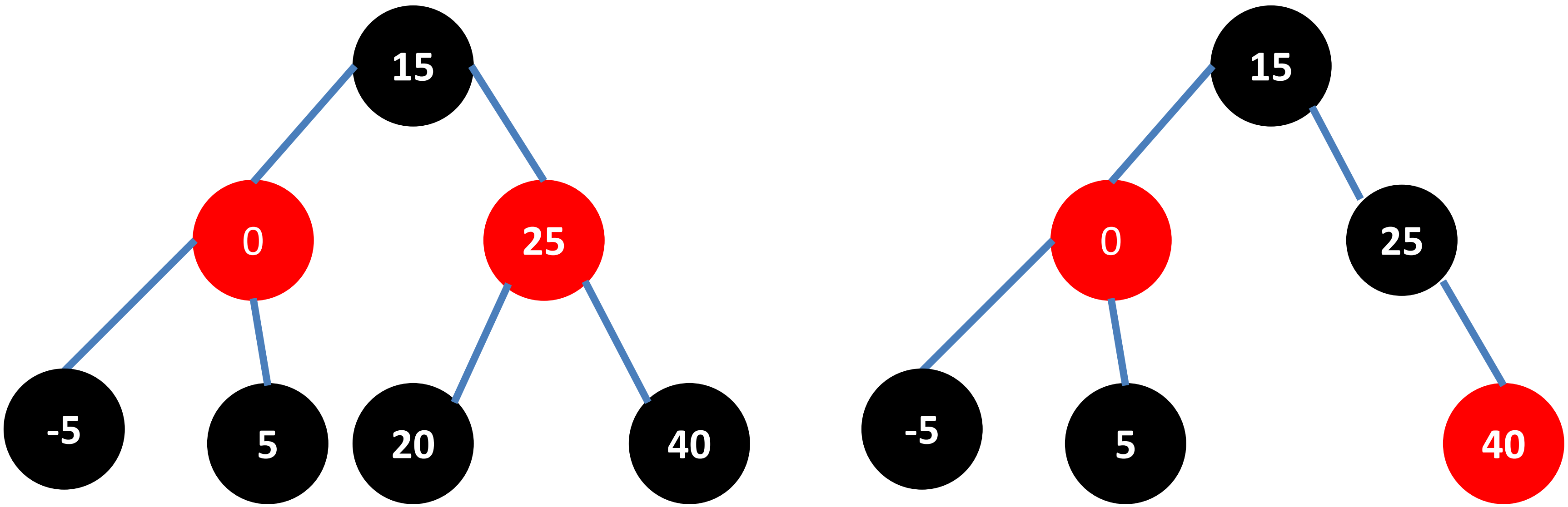


Case VI: Double-black node has a red/black parent and a black right sibling, and the black sibling has a red/black left child and a red right child

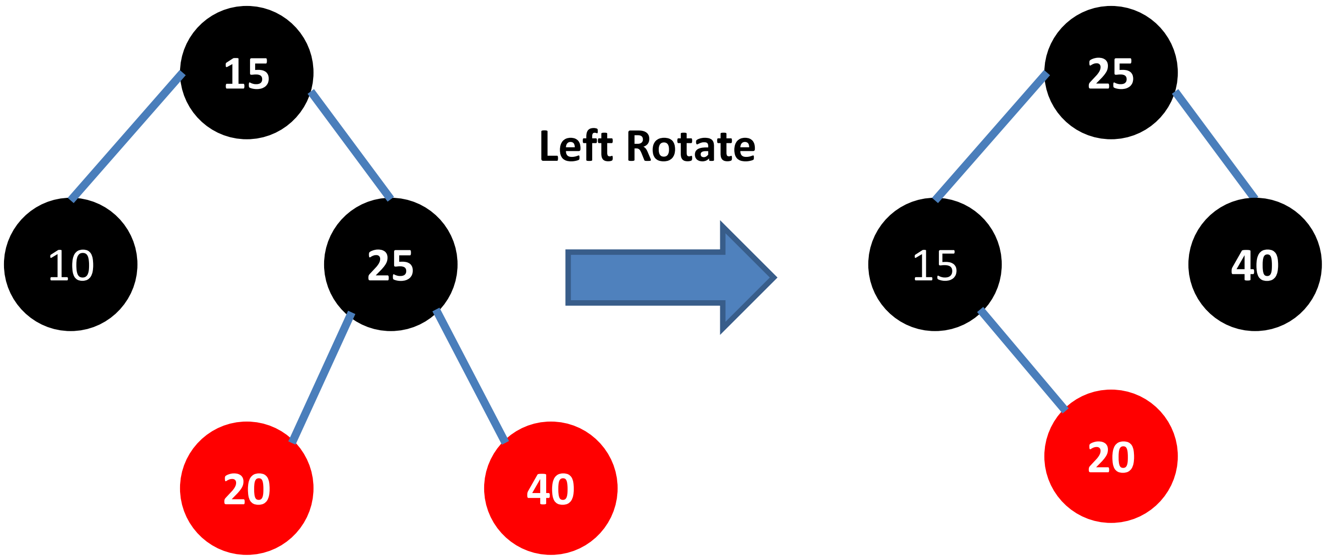
→ Recolor right sibling same as parent color, parent as black, right child of right sibling as black, rotate left around parent, and stop.



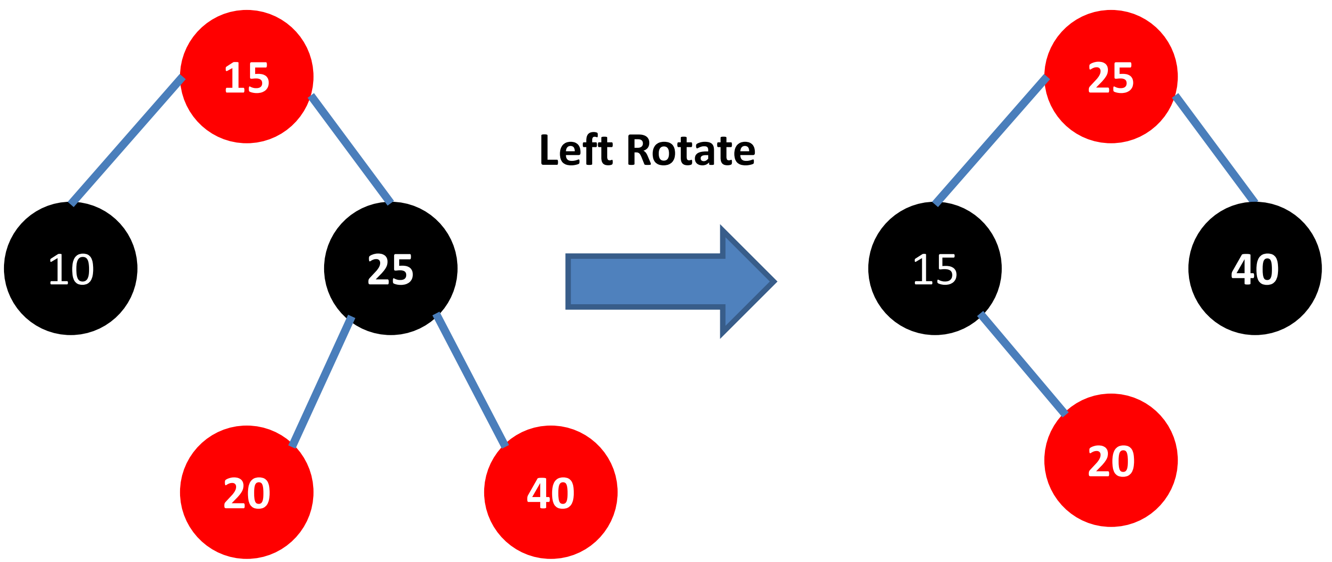
Case IV: Delete 20



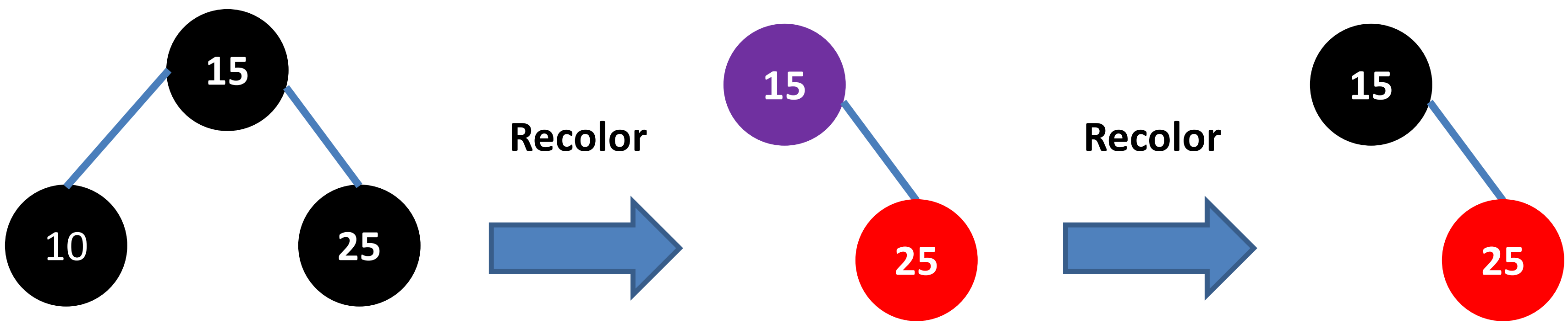
Case VI: Delete 10



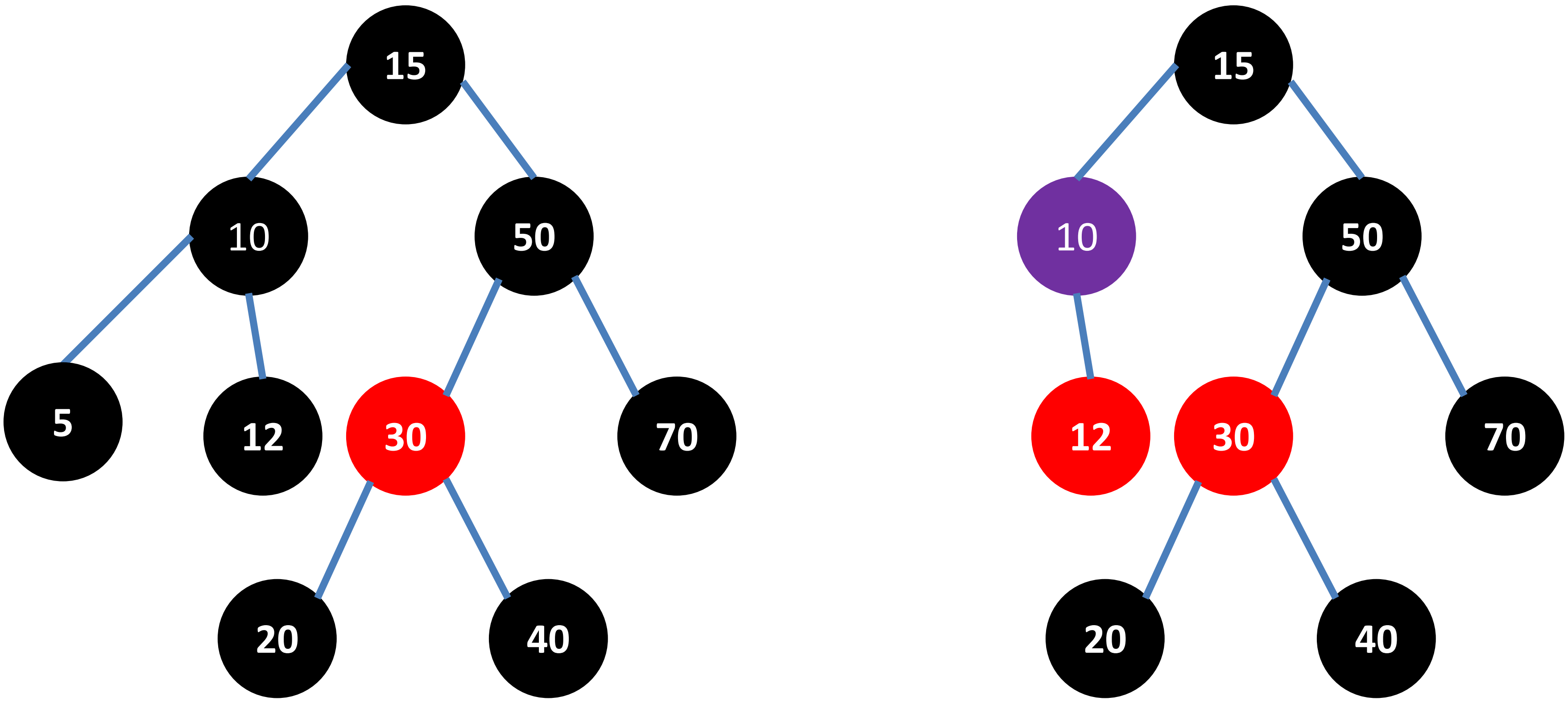
Case VI: Delete 10



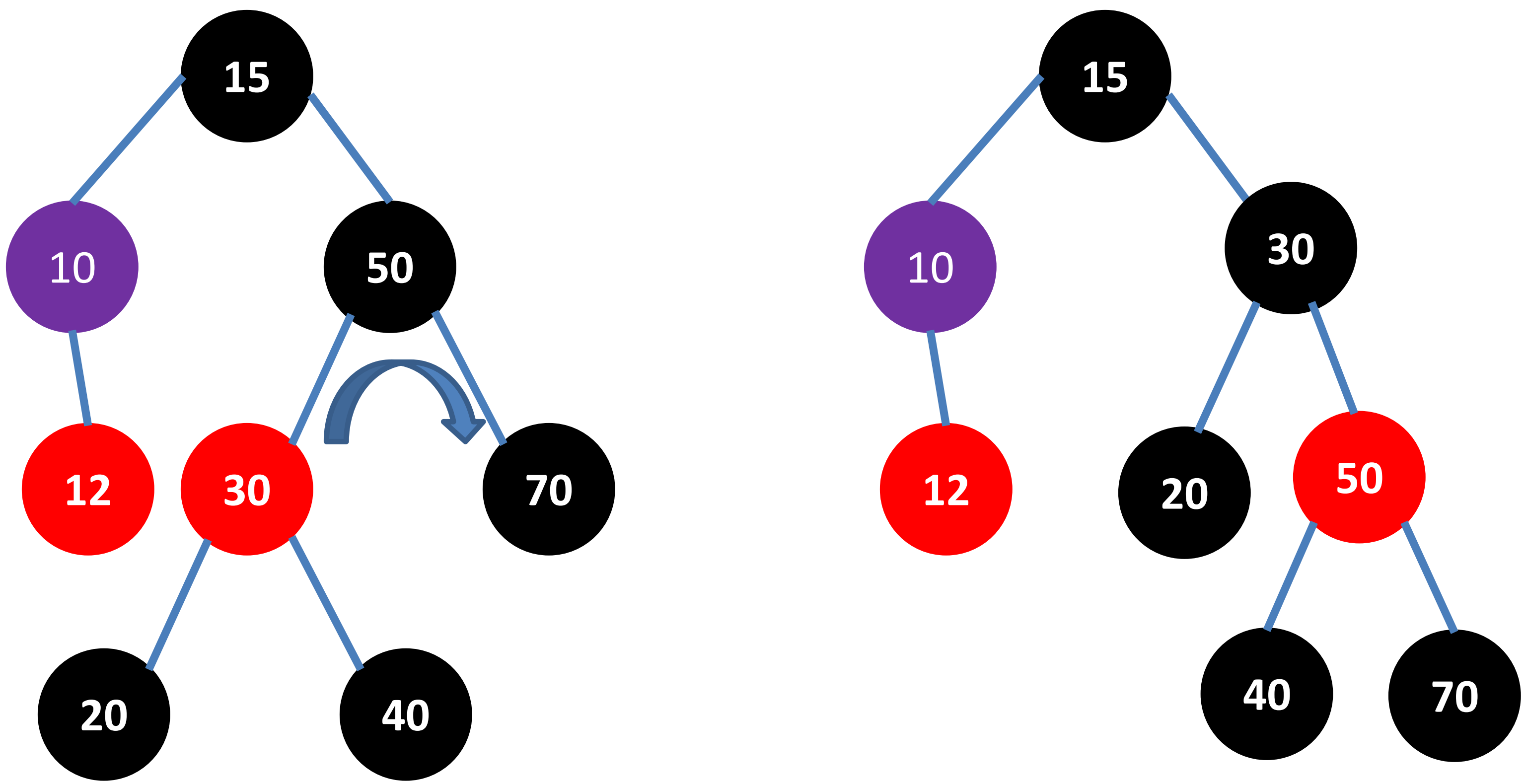
Case III: Delete 10



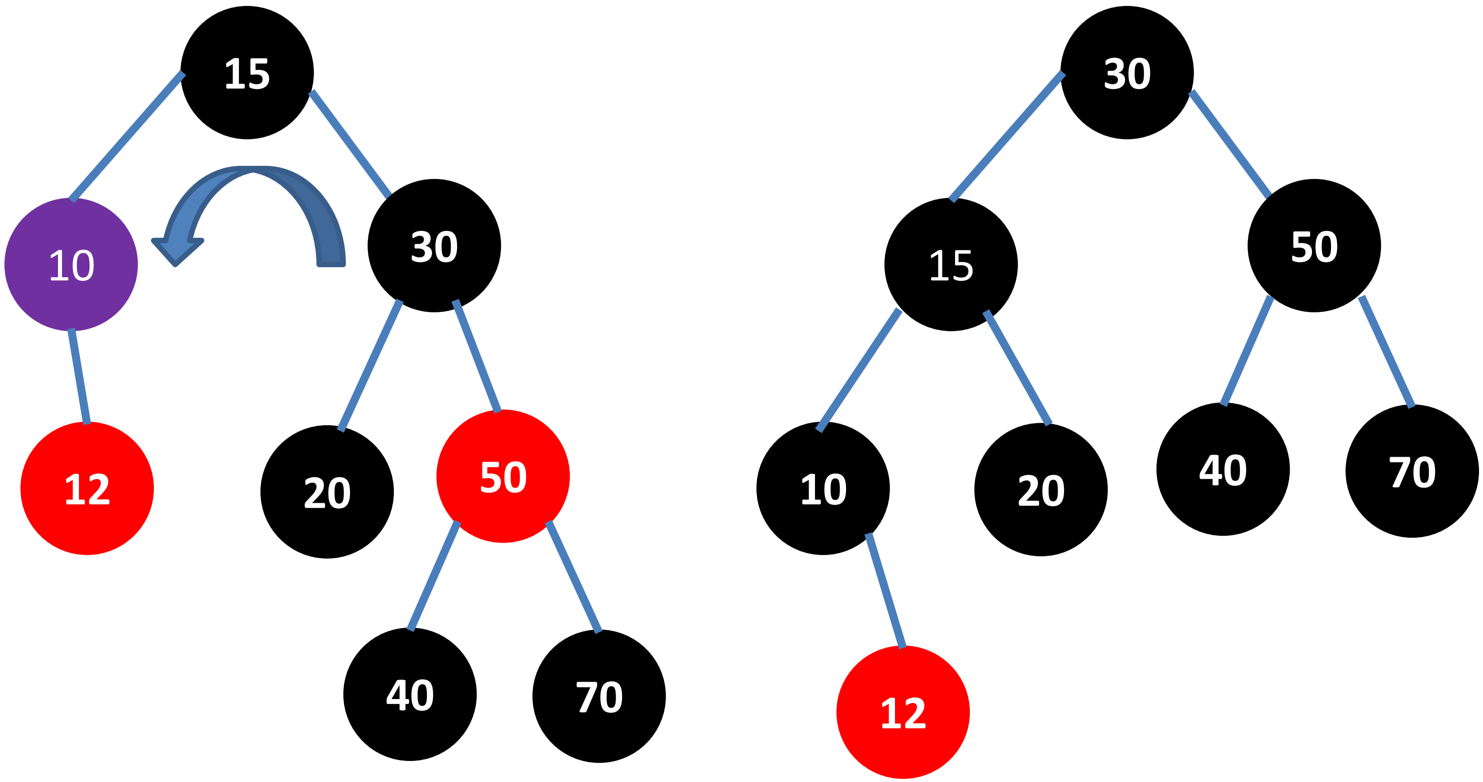
Case III: Delete 5



Case V: Right Rotate



Case VI: Left Rotate



Case II: Delete 30