Data Structures

COMP2321

Research Paper No. 1

Red-Black Tree

Instructor: Dr. Radi Jarrar

Name: Anas Nimer

ID: 1180180

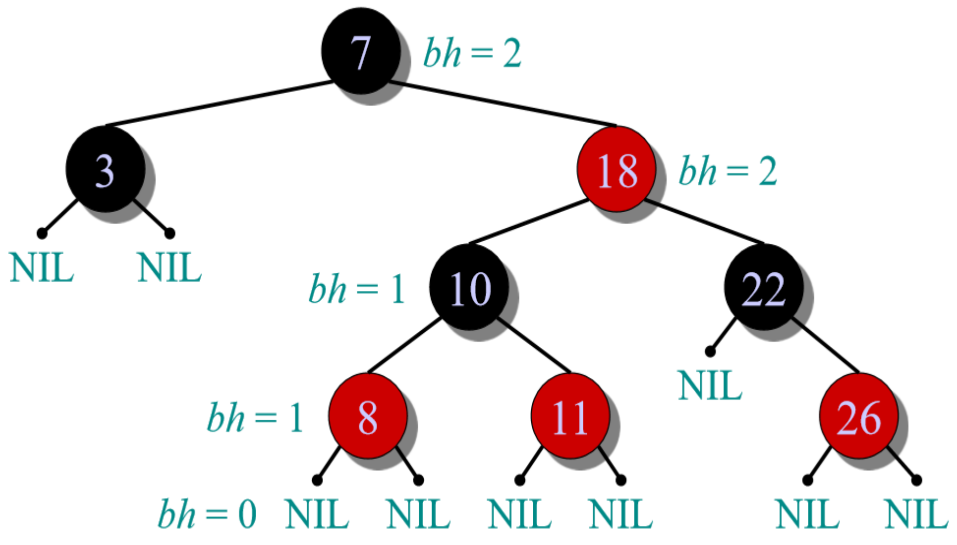
Section: 4

Contents:

- ## **Introduction:**

The red and black tree is a form of a self-balancing binary search tree. Each binary tree node contains an additional bit, and these bits represent the color of the node (red or black). These color bits are used to ensure the tree remains roughly balanced during insertions and deletions. The balance is maintained by painting each tree node in one of two colors, in a way that matches these characteristics:

1- The root must have the black color.
2- If a node is red, then both its children are black.
3- Every path from a given node to any of its goes through the same number of black nodes.
4- The shortest path must contain all black nodes.
5- The longest path has a different sequence of red and black nodes.
6- The path from the root to the farthest leaf is no more than twice the length of the path from the root to the nearest leaf.



Red _black tree

- ## **When it is best to use:**

The AVL trees are more balanced compared to Red-Black Trees, but they may cause more rotations during insertion and deletion. So if your application involves many frequent insertions and deletions, then Red Black trees should be preferred. And if the insertions and deletions are less frequent and search is a more frequent operation, then AVL tree should be preferred over Red-Black Tree.

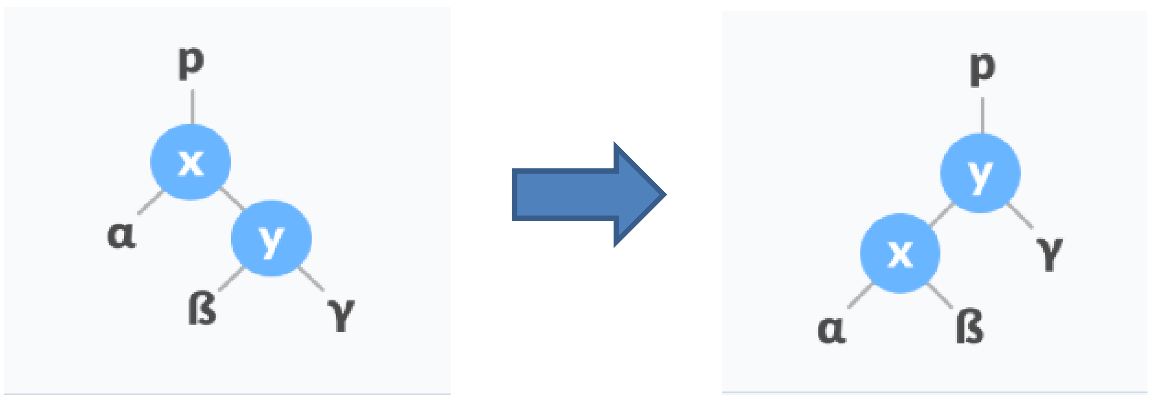
- **<u>Rotation:</u>**

In rotation operation ,The location of the nodes in the sub-tree are interchanged such that the rotation process is used to maintain the properties of the black red tree while performing certain operations, such as insertion and deletion.

There are 2 rotation types:

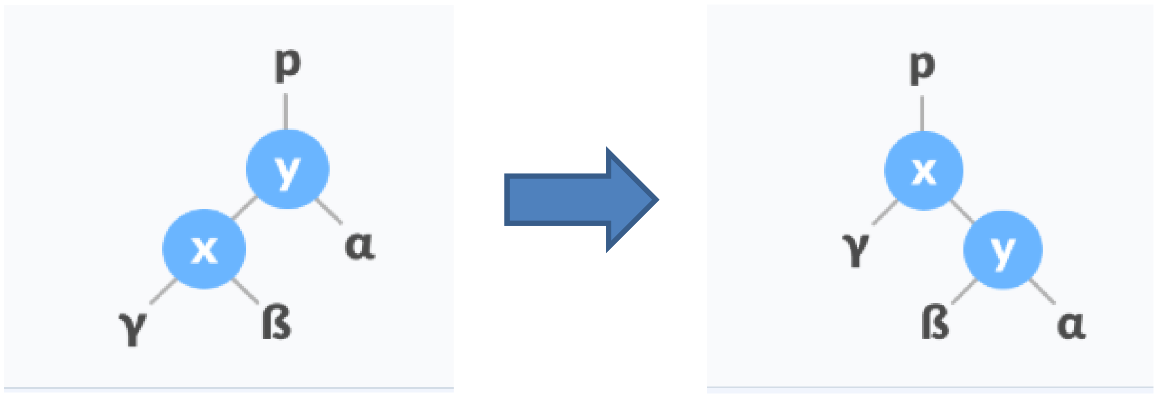
1- **<u>Left Rotation:</u>**

In the left spin, the arrangement of the nodes on the right is converted to the arrangements on the left node so that her right child becomes the mother of the node, and her left child becomes her new right child.



2- **<u>Right Rotation:</u>**

In the right spin, the arrangement of the nodes on the left is converted to the arrangements on the right node where the left child becomes a parent of the node, and the right child becomes the new left child.



- **Operation:**

Like many other trees, the red and black tree has many operations such as adding, deleting and searching. Performing these operations may require a difference in the colors of the nodes to preserve the characteristics of the red and black tree.

1. Insertion:

The process of inserting into the black red tree is performed using the following steps :

Step 1: If the tree is empty, enter the new node as the black root node.

Step 2: If the tree is not empty, enter the new node as a red leaf node, if the root of the inserted node is black. The process is exited, while if its origin is red, it must be changed to black.

Step 3: If the inserted node is red and the grandfather of this node has a child in red, then we change the color of the grandfather (if isn't root) to red and the children to black.

Step 4: If the node is inserted from the side (left of the right) of the root or from the side (right of the left) of the root, we must rotate the node in the opposite direction of the insertion process with the necessary operations preserving the properties of the tree, for example (changing the colors and locations of the nodes).

Step 5: If the inserted node is inserted in the left from the left or the right from the root, then we must rotate the grandfather in the direction opposite to the inserted node so that the parent takes the place of the grandfather, the grandfather becomes the left or right child of the parent, so in this step we need switch the colors of the parents and grandparents After completing the rotation.

2. **Find**:

Searching for an item in a red-black tree does not differ from searching for an item in the binary tree, so if the number we want to search for is greater than the node, we search for it in the right branch of the tree and we continue searching until it is found and returned, while if it is not found we know that the element is not found in The tree, and also if the object we want to search for is smaller than the node it is searched for in the left branch of the tree.

3. **Deletion**:

When performing the process of deleting an element from a red black tree in this case it can lead to the lack of verification of a characteristic of a red black tree so we must do some things in order to achieve all these properties.

Step 1: perform Binary search tree deletion.

Step 2:

Case 1: if node of be deletion is red just delete it.

Case 2: if root is double black just remove double black.

Case 3: if double black's sibling is black and both it's children are black then :

- **a)** remove double black.
- **b)** add black to it's parent such that if parent is red recolor it black but if parent is black it become double black.
- **c)** make sibling red .
- **d)** if parent still double black exist so we need to apply other cases.

Case 4: If double black's sibling is red

- **a)** we swap colors of the parent and it's sibling.
- **b)** rotate parent in the double blacked direction.
- **c)** Replay this cases.

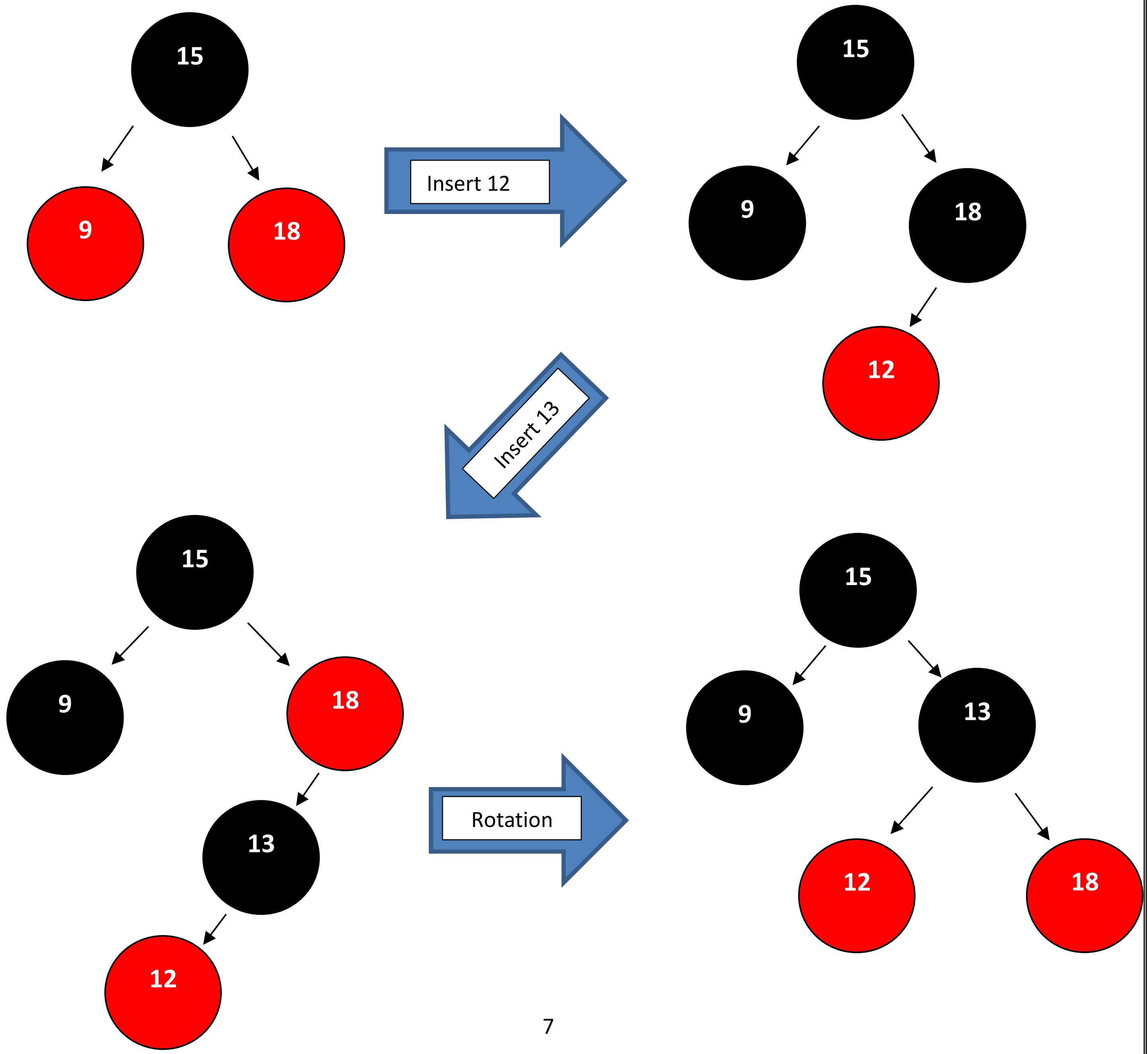
Case 5: if double blacks sibling is black Sibling children who is far from double black is black , but the children near the double black is red.

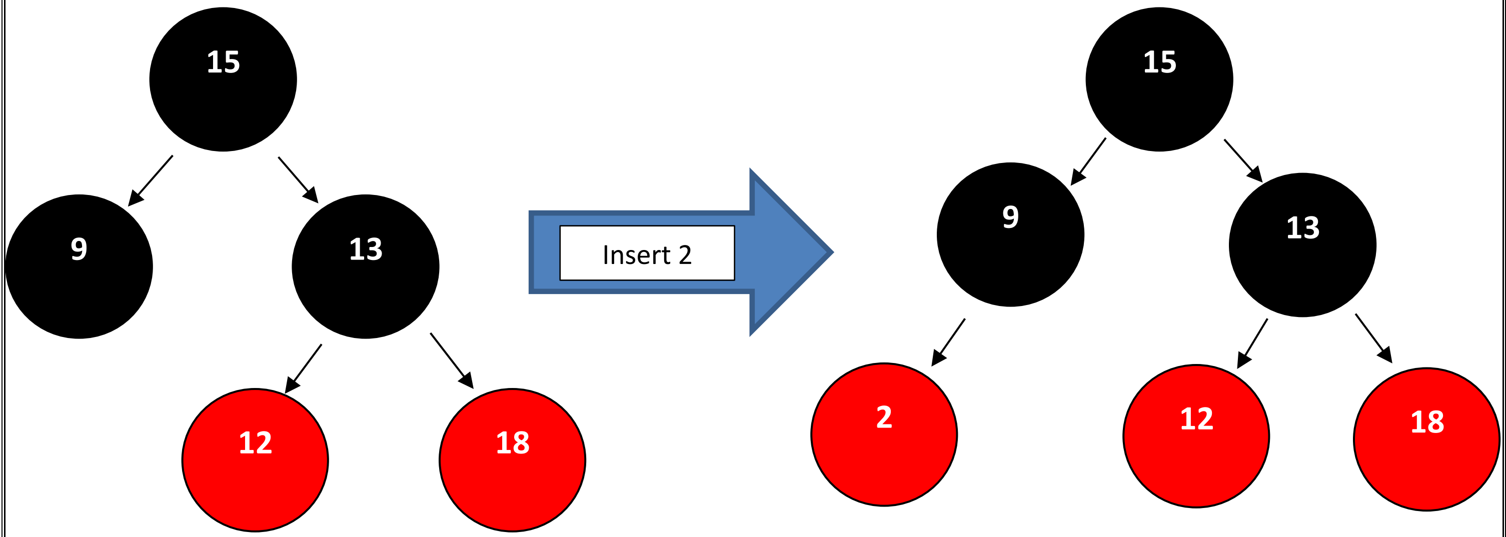
<u>Case 6</u>: if double black sibling is black and the far children is red then we need :

a) Swap color of parent and sibling.
b) Rotate parent in double black direction.
c) Remove double black.
d) Chang color of red children to black.

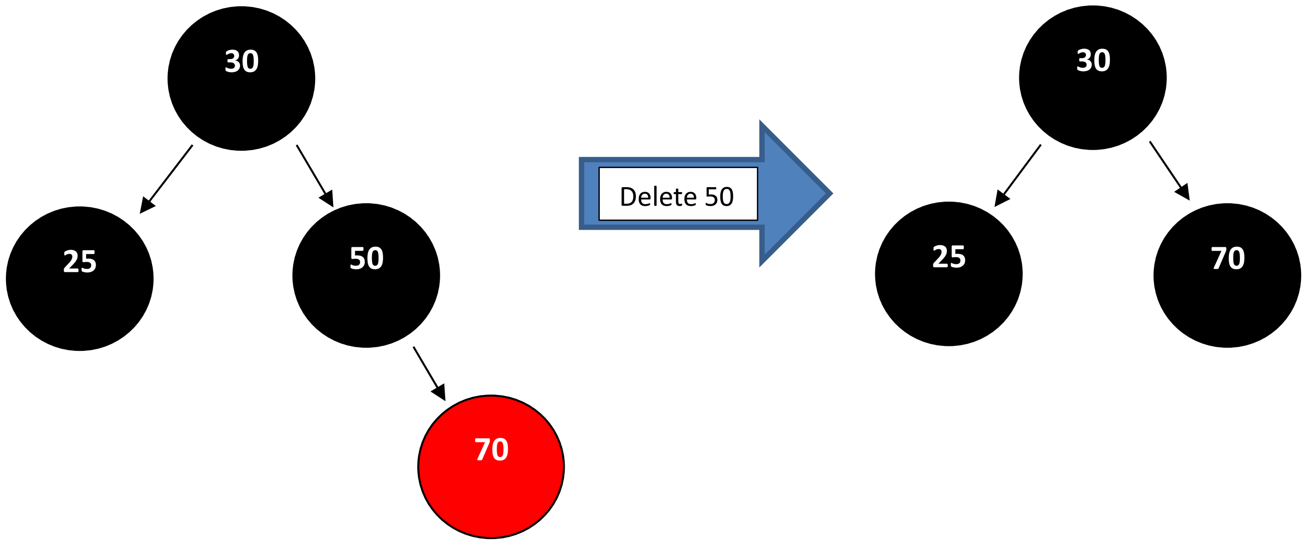
- **<u>Example</u>** :

1- <u>Insert</u>:

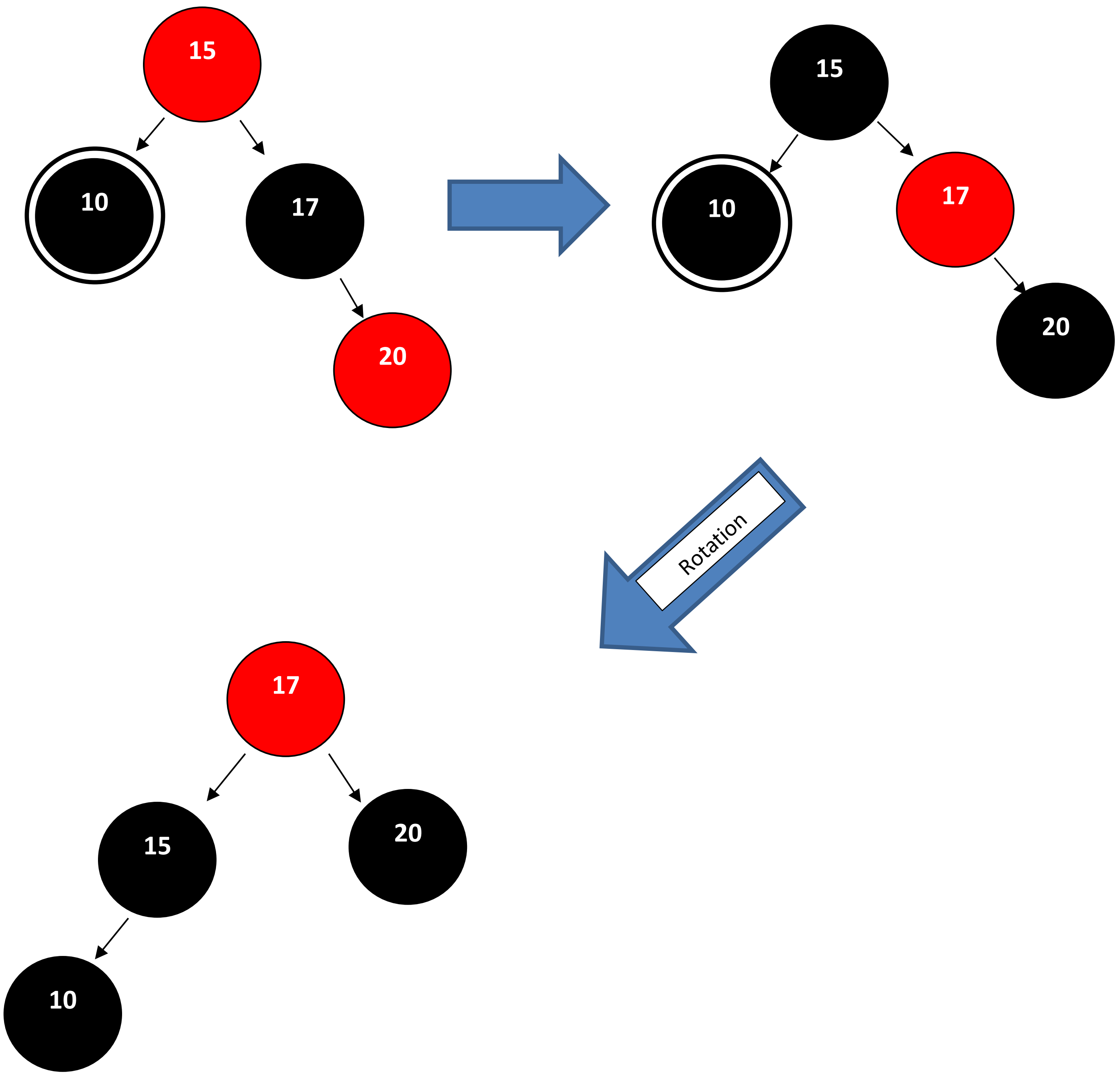




2- Delete:

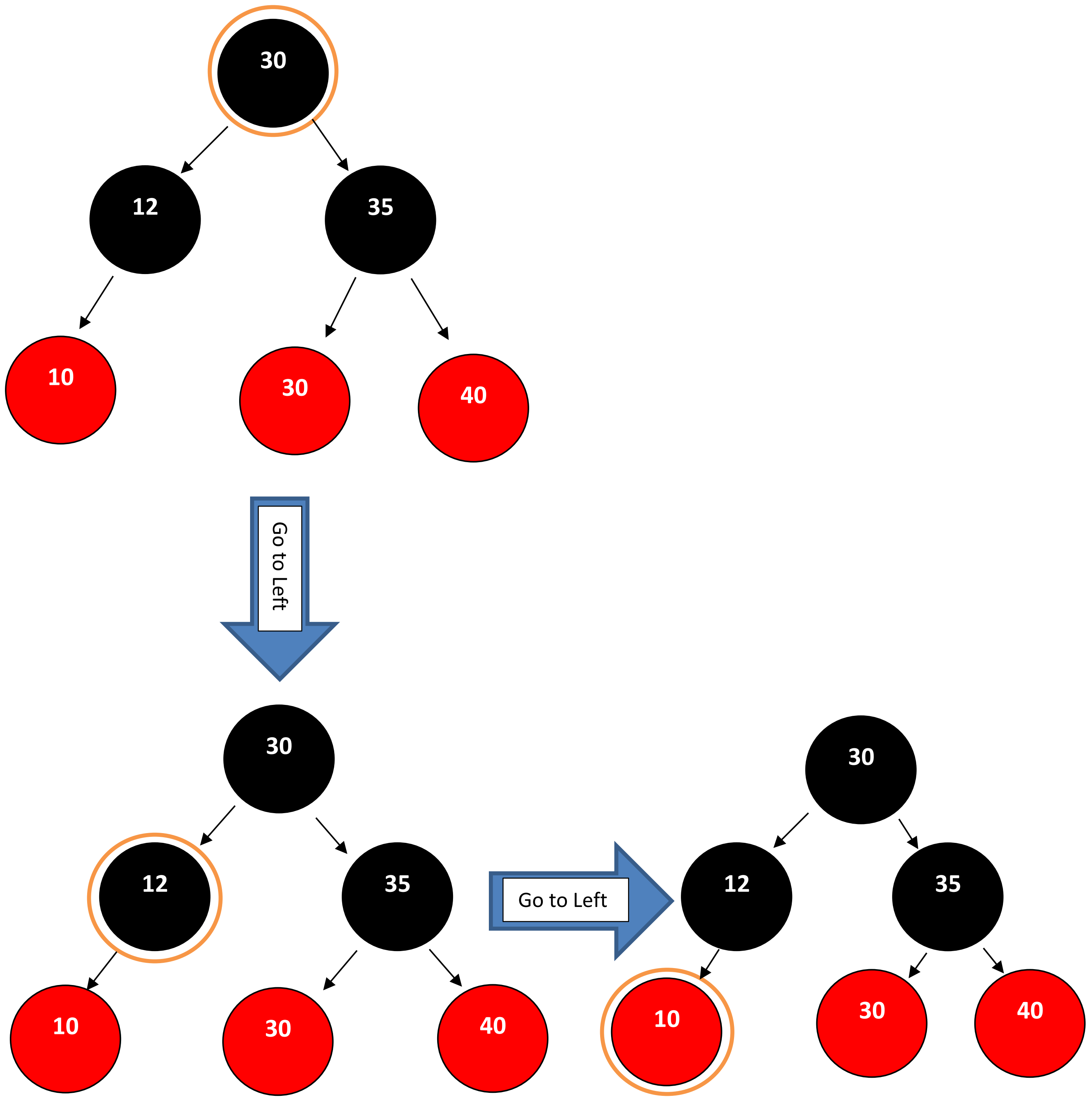


Note: in above example we need to delete node number 50 , so we delete the node 50 and re-color node 70 from red to black.



<u>Note:</u> in above example we need to delete double black node number 10 , so we need to re-color node 17 from black to red and re-color node 20 from red to black after that , make rotation to the node 17 , make it root and re-color it from black to red . and make node 15 ,20 parents and node 10 child to node 15.

3- **<u>Search (Find):</u>**



<u>Note:</u> in above example we need to search about number 10 node .

- **Time Complexities:**

The Table below shows the time complexity and the space complexity of different operations:

Operation	**Time Complexity**
Rotation	O (1)
Insertion	O (log n)
Find	O (log n)
Deletion	O (log n)
Space Complexity	O (n)

- **References:**

1- https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/

2- https://www.youtube.com/watch?v=qA02XWRTBdw

3- https://iq.opengenus.org/red-black-tree-search/

4- https://www.geeksforgeeks.org/red-black-tree-set-3-delete-2/

5- https://www.youtube.com/watch?v=w5cvkTXY0vQ&list=PLfKWEzH_uy-ViXSB8r4z3MwSMFph7X5iF&index=3