Class Overview

- Definition
- Tree height
- Rotation and color flip
- Insert
- Delete

Definition

A red-black tree is a special binary search tree in which

- every node is either red or black;
- the root is black;
- every leaf (null) is black;
- if a node is red, then both its children are black;
- every simple path from a node to a descendant leaf contains the same number of black nodes.

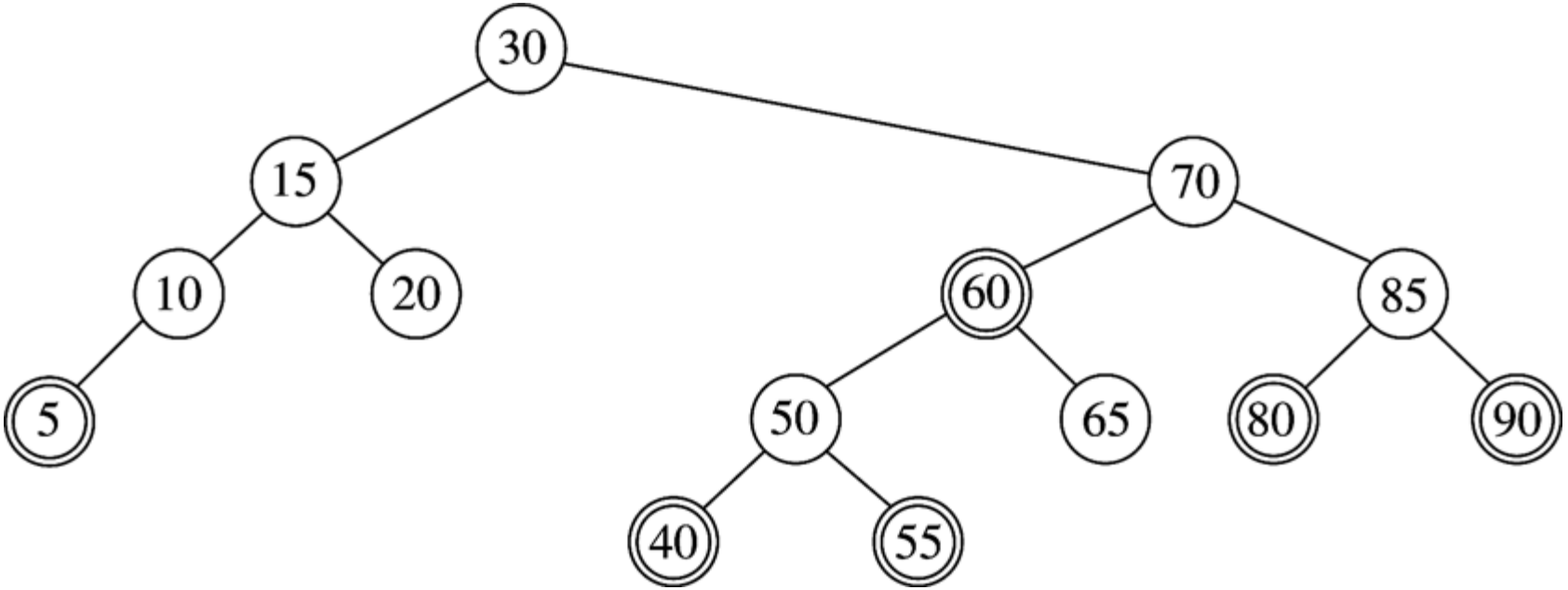
Height

- Experiments suggests that the average red-black tree is about as deep as an average AVL tree.
- The rotation happens less frequently.

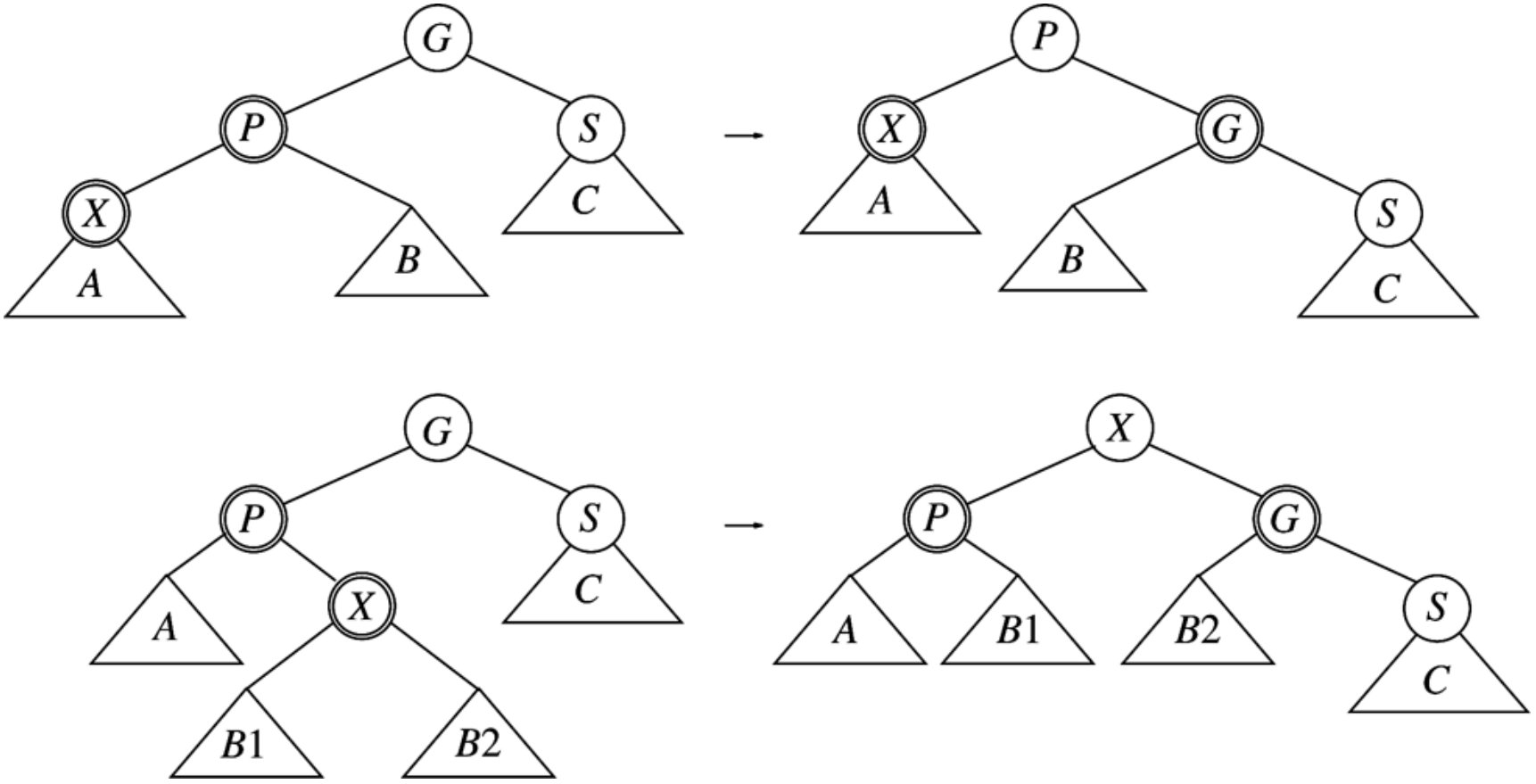
Cases

1. Add new leaf and add it red initially, if parent is black you are done……but
2. If parent is red rule 3 is broke so you must…
    1. If parents sibling is also red you can change grandparent to red and change parent and sibling to black. If root changes to red, take it back to black and you are done, no rules broken.
    2. If parent does not have a red sibling then we change the color of the grandparent to red and the parent to black which breaks rule #4. Correct this by rotating about the grandparent so that the parent moves into the position where the grandparent was, thus restoring rule #4 and bringing peace to the World.
    3. Of course that only works if you have a left left, or right right type problem, if you have a right-left or a left-right you have to do a double rotation like in the AVL trees.

An Example

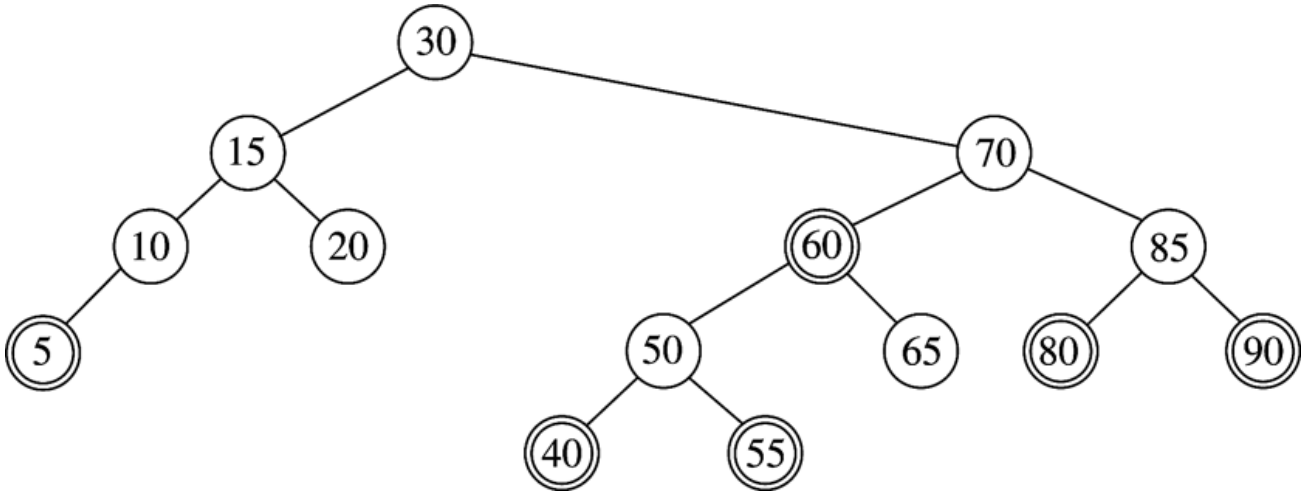


Rotation if S is Black

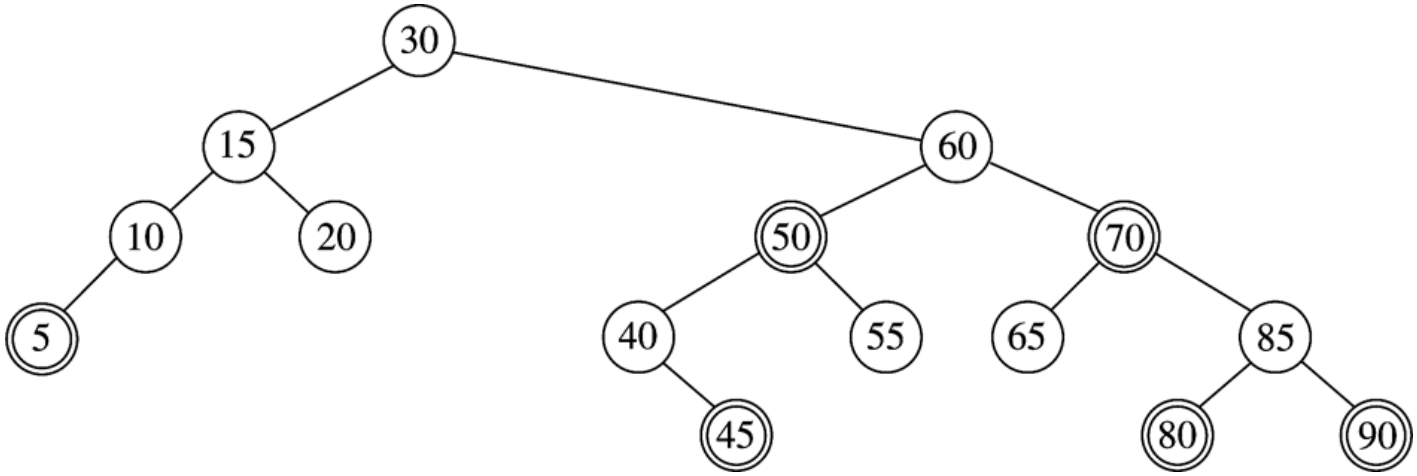


Color Flip if S is red

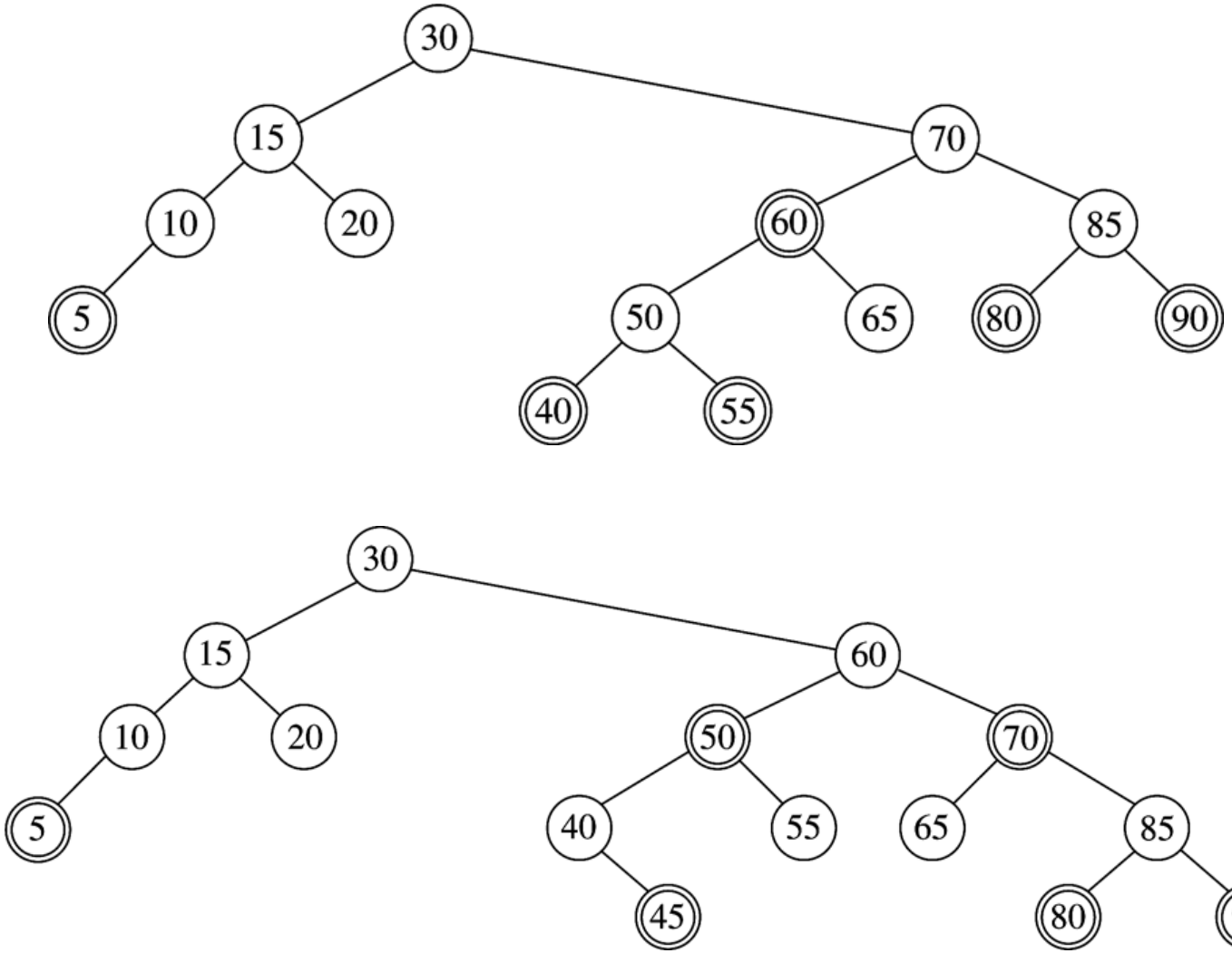
Example: Insert 45



After insertion of 45 into last slide



Both together



Insert

```
    /**
     * Insert into the tree.
     * @param item the item to insert.
     */
    public void insert( AnyType item )
    {
        current = parent = grand = header;
        nullNode.element = item;

        while( compare( item, current ) != 0 )
        {
            great = grand; grand = parent; parent = current;
            current = compare( item, current ) < 0 ? current.left :
current.right;

                // Check if two red children; fix if so
            if( current.left.color == RED && current.right.color == RED )
                handleReorient( item );
        }

            // Insertion fails if already present
        if( current != nullNode )
            return;
        current = new RedBlackNode<AnyType>( item, nullNode, nullNode );

            // Attach to parent
        if( compare( item, parent ) < 0 )
            parent.left = current;
        else
            parent.right = current;
        handleReorient( item );
    }
```

Insert

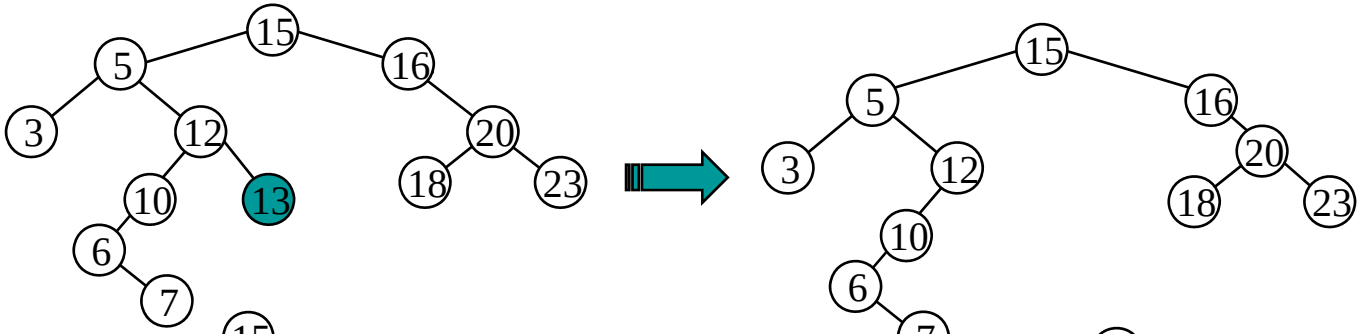
```
/**
 * Internal routine that is called during an insertion
 * if a node has two red children. Performs flip and rotations.
 * @param item the item being inserted.
 */
private void handleReorient( AnyType item )
{
        // Do the color flip
    current.color = RED;
    current.left.color = BLACK;
    current.right.color = BLACK;

    if( parent.color == RED )   // Have to rotate
    {
        grand.color = RED;
        if( ( compare( item, grand ) < 0 ) !=
            ( compare( item, parent ) < 0 ) )
            parent = rotate( item, grand );  // Start dbl rotate
        current = rotate( item, great );
        current.color = BLACK;
    }
    header.right.color = BLACK; // Make root black
}
```

Insert

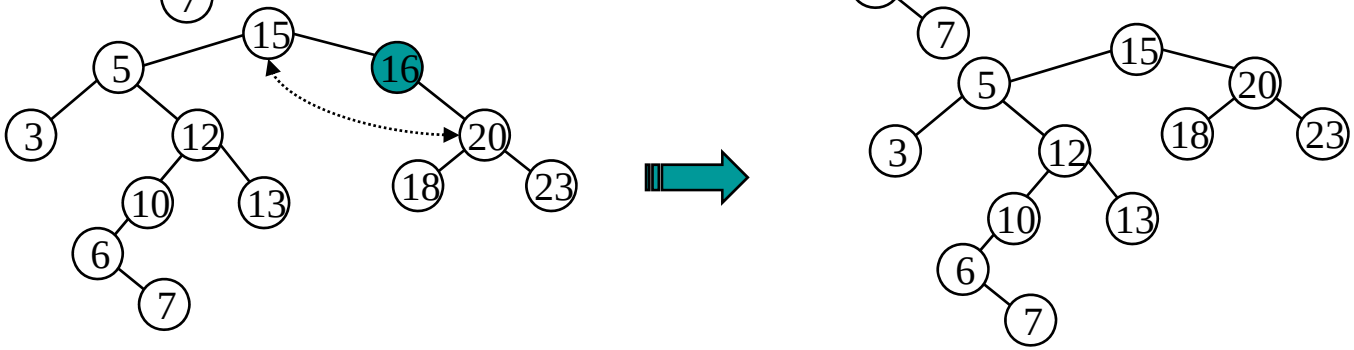
```
    /**
     * Internal routine that performs a single or double rotation.
     * Because the result is attached to the parent, there are four cases.
     * Called by handleReorient.
     * @param item the item in handleReorient.
     * @param parent the parent of the root of the rotated subtree.
     * @return the root of the rotated subtree.
     */
    private RedBlackNode<AnyType> rotate( AnyType item, RedBlackNode<AnyType>
parent )
    {
        if( compare( item, parent ) < 0 )
            return parent.left = compare( item, parent.left ) < 0 ?
                rotateWithLeftChild( parent.left )  :  // LL
                rotateWithRightChild( parent.left ) ;  // LR
        else
            return parent.right = compare( item, parent.right ) < 0 ?
                rotateWithLeftChild( parent.right ) :  // RL
                rotateWithRightChild( parent.right );  // RR
    }
```

Delete

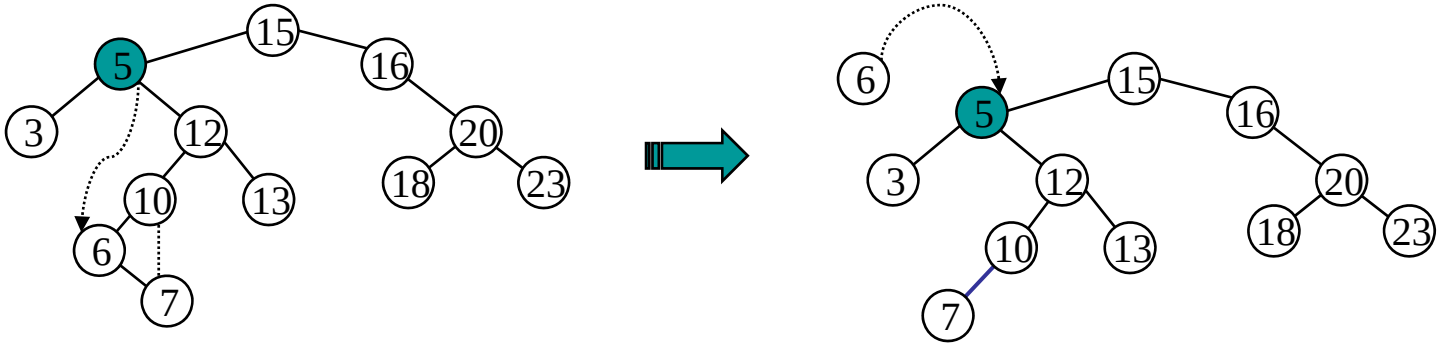
Case 1:



Case 2:



Case 3:



Delete

- In terms of deletion, everything boils down to delete a leaf.
- Deletion of a red leaf is trivial.
- How to delete a black leaf: Change the color of the leaf to be deleted to red by a top-down pass.

Three Cases When X Has Two Black Children