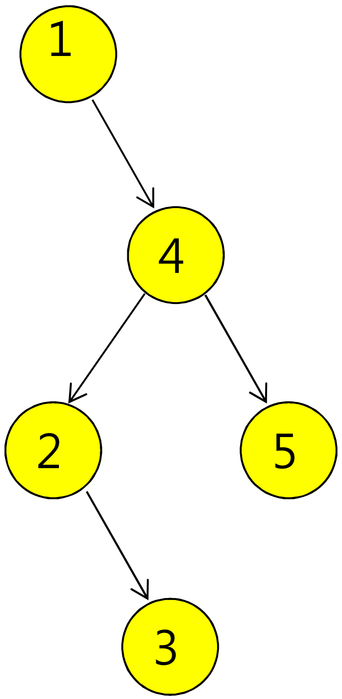
BST trees



insert(1)
insert(4)
insert(2)
insert(5)
insert(3)

- How many (different ways) BST trees for the insertion of five keys in {1, 2, 3, 4, 5} ?
- What is the shortest height a BST can have for n key insertions ?
- What is the longest height a BST can have for n key insertions ?

$$\boxed{} \leq h \leq \boxed{}$$

Time complexity

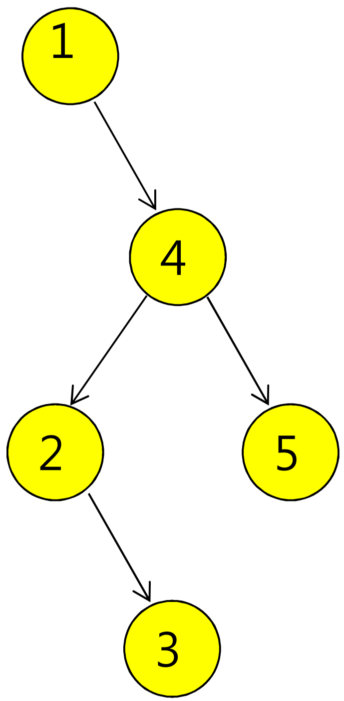
The height of a BST depends on the order in which the data is inserted into it.

ex. 1 3 2 4 5 7 6 vs. 4 2 3 6 7 1 5

For a randomly generated BST by n insertions, the time to process the following operations:

operation	Avg. case	Worst case	Sorted array	Sorted list
find				
insert				
delete				
traverse				

Height Balanced Tree



Let

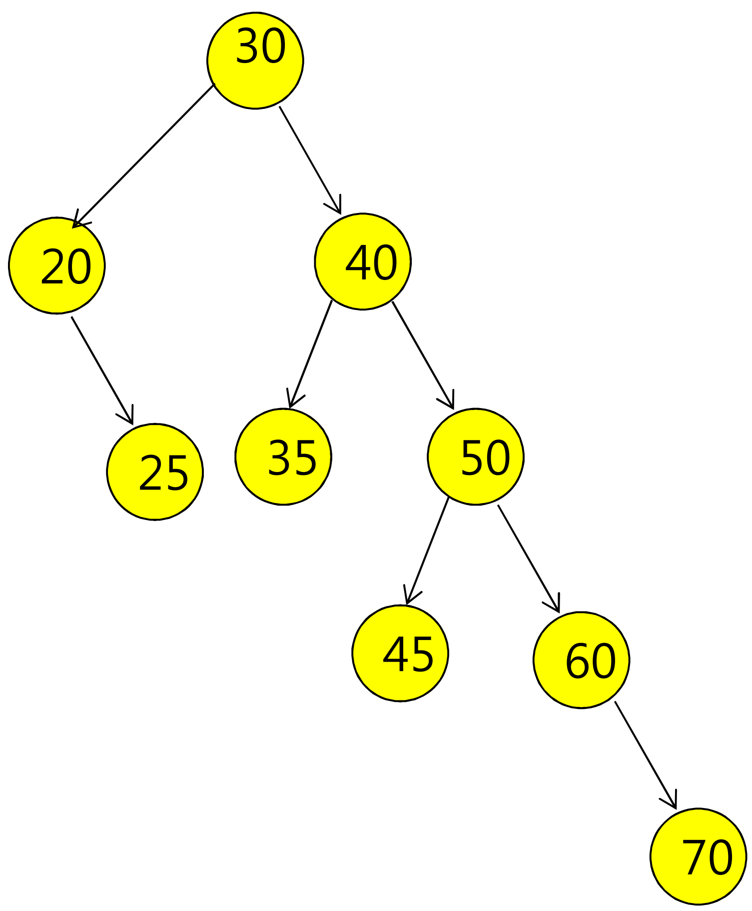
$bf = height(T_L) - height(T_R)$

A BST T is called height balanced if:

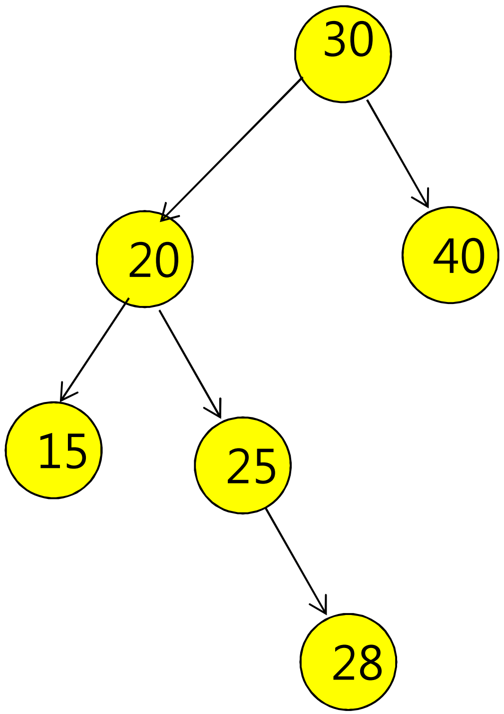
$|\mathbf{bf}| \leq \mathbf{1}$

- T = { } OR
- $T = \{r, T_L, T_R\}$, ~~bf = 1 or -1~~, and T_L and T_R are height balanced.

Operations to be balanced BST tree



Another rotation



Rotation summary:

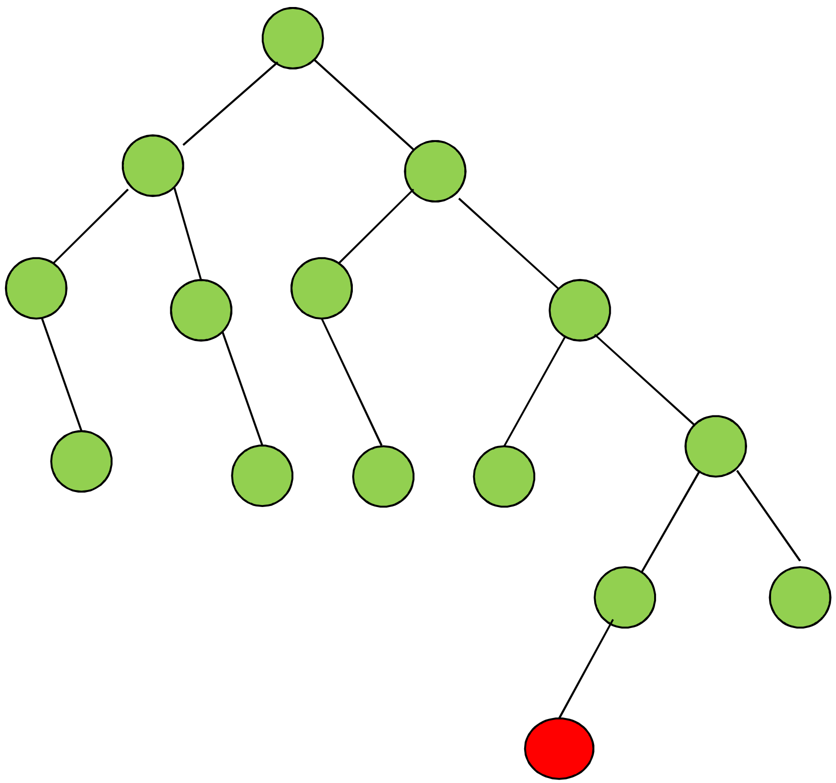
- 4 kinds: left, right, left-right, right-left
- O(1) time and local operations
- Still BST maintained

We apply a rotation operation whenever an insertion/removal causes an imbalanced trees.

The issues for implementation:

1. Detecting imbalance (?)
2. Selecting rotation type (?)
3. Rotating – discussed

AVL trees:

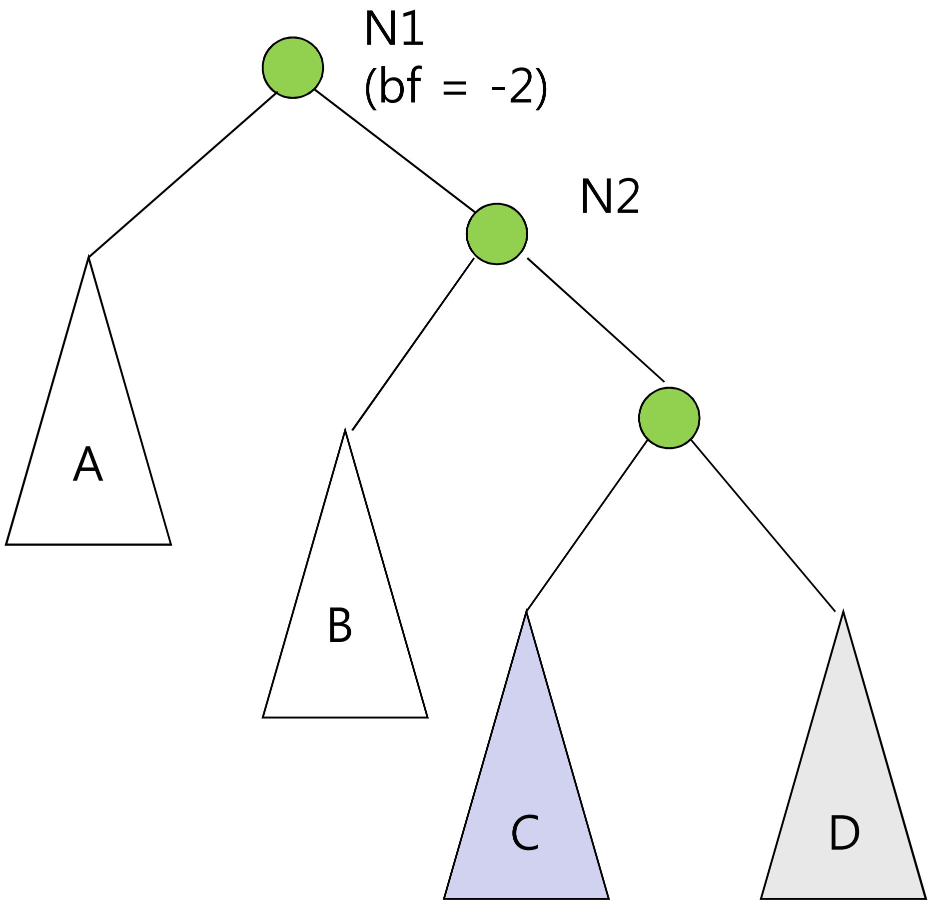


Insert:

- Insert at proper place
- Update height
- Check for imbalance
- Rotate if necessary

```
class treeNode {
   T key;
   int height;
   treeNode * left;
   treeNode * right;
};
```

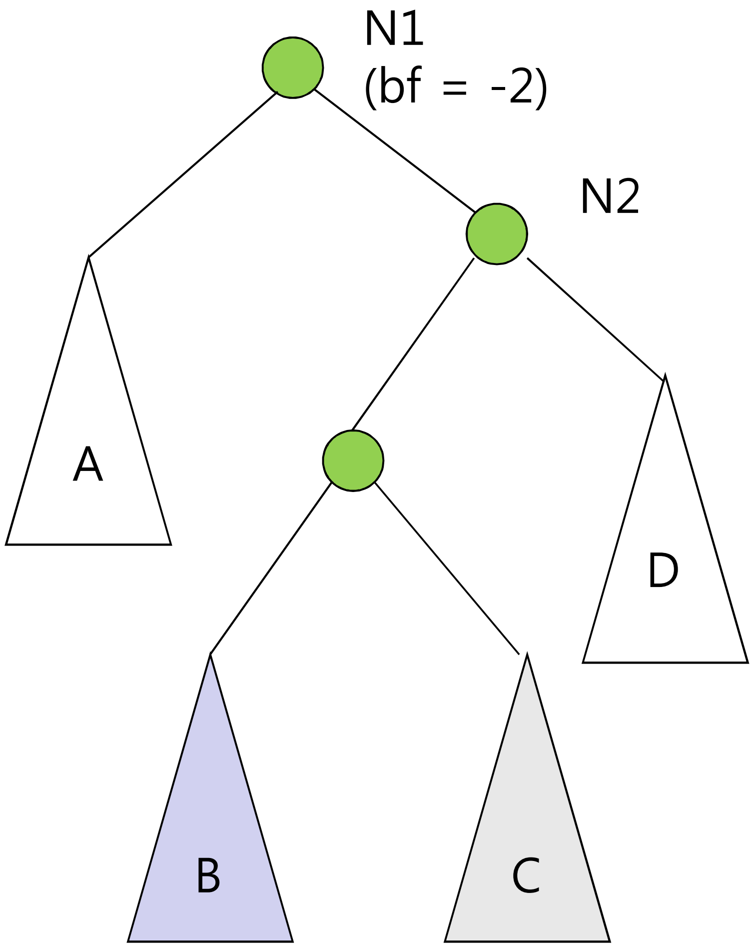
Identifying single rotation



If an insertion was made in subtrees C or D and if an imbalance occurs at N1, then a ______________ rotation about N1 rebalances the tree.

The imbalance at N1 is established by noting that bf of N2 is _____

Identifying double rotation



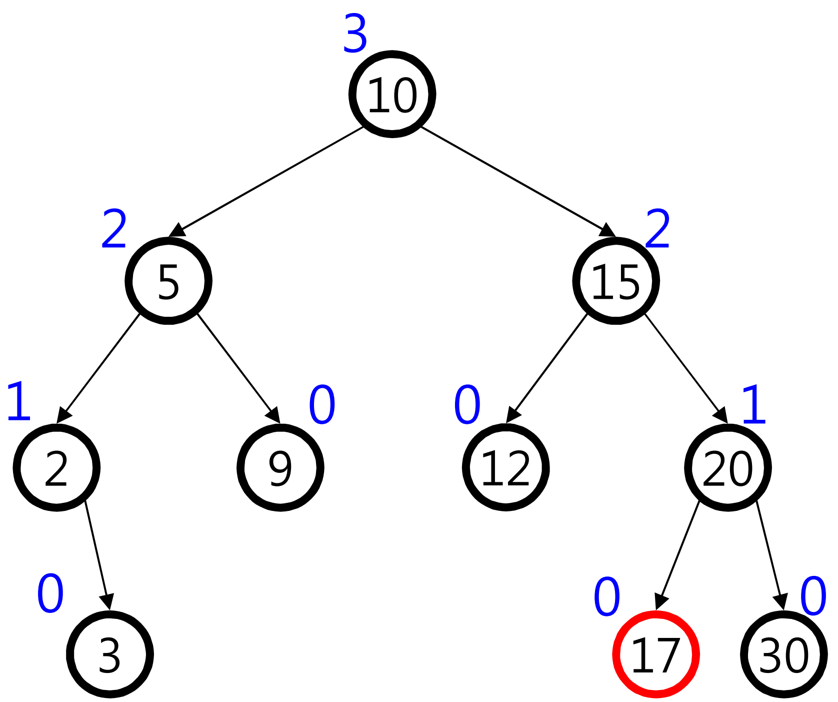
If an insertion was made in subtrees B or C and if an imbalance occurs at N1, then a ____________________ rotation about N1 rebalances the tree.

The imbalance at N1 is established by noting that bf of N2 is ______

```
template <class T>
void AVLTree<T>::insert(const T & x, treeNode<T> * & t ) {
    if ( t == NULL ) t = new treeNode<T>( x, 0, NULL, NULL ); return;
    else if ( x < t->key ) {
        insert( x, t->left );
        int bf = height(t->left)-height(t->right);
        int left_bf = height(t->left->left)-height(t->left->right);
        if ( bf == 2 )
            if ( left_bf == 1 )
                  rotate____________( t );
            else
                  rotate____________( t );
    }
    else if ( x > t->key ) {
        insert( x, t->right );
        int bf = height(t->left)-height(t->right);
        int right_bf = height(t->right->left)-height(t->right->right);
        if ( bf == -2 )
            if ( right_bf == -1 )
                  rotate____________( t );
            else
                  rotate____________( t );
    }
    t->height=max( height(t->left), height(t->right) ) + 1; return;
```

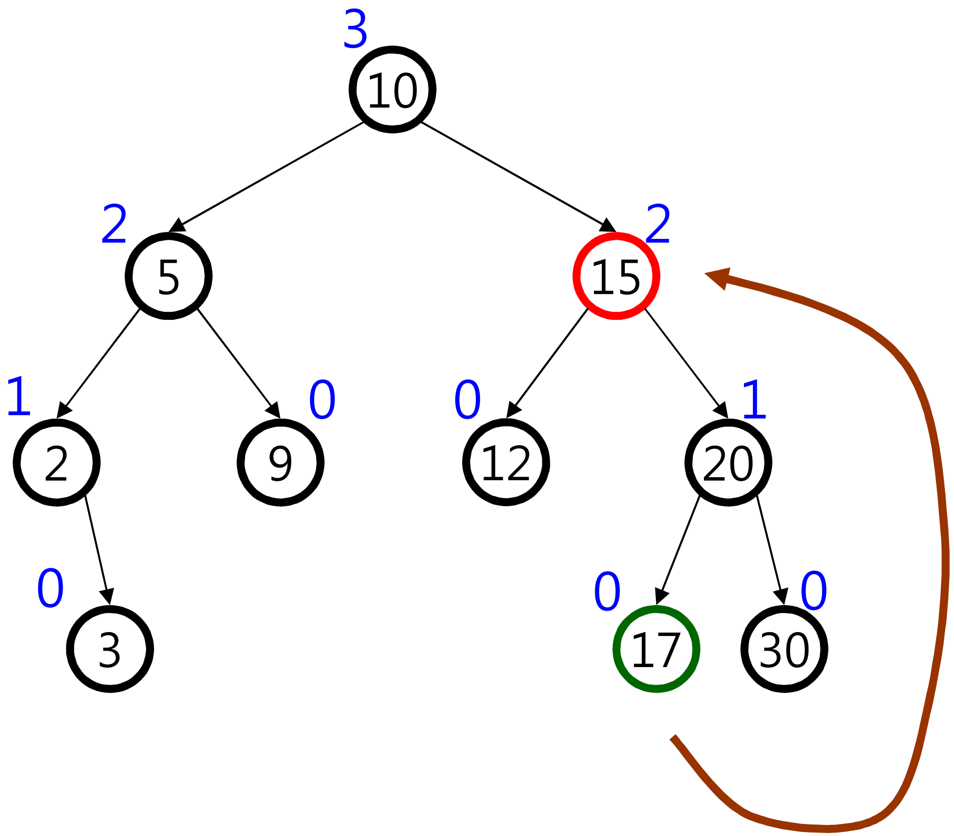
Deletion (no rotation)

Delete(17)



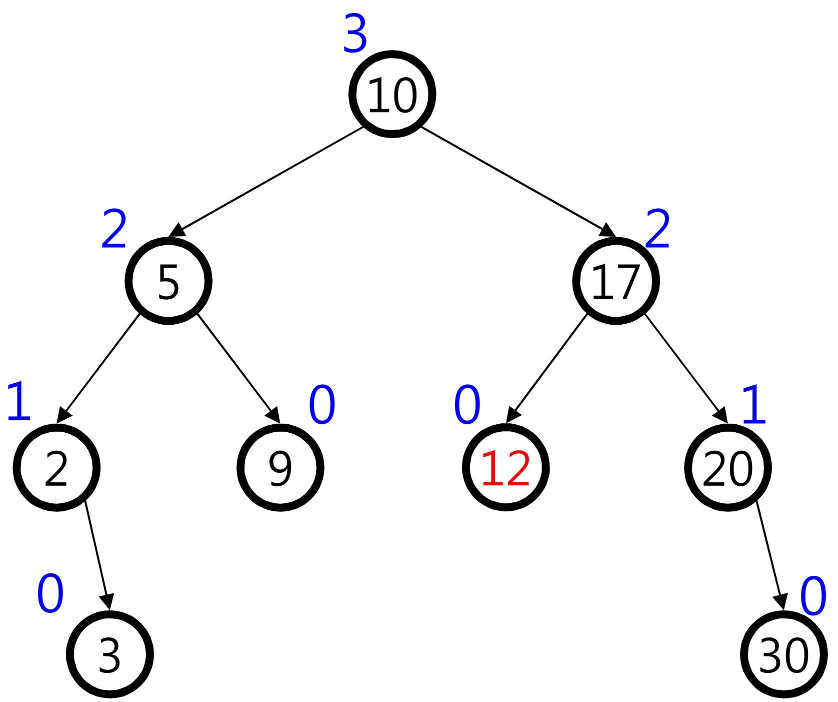
Deletion (no rotation)

Delete(15)

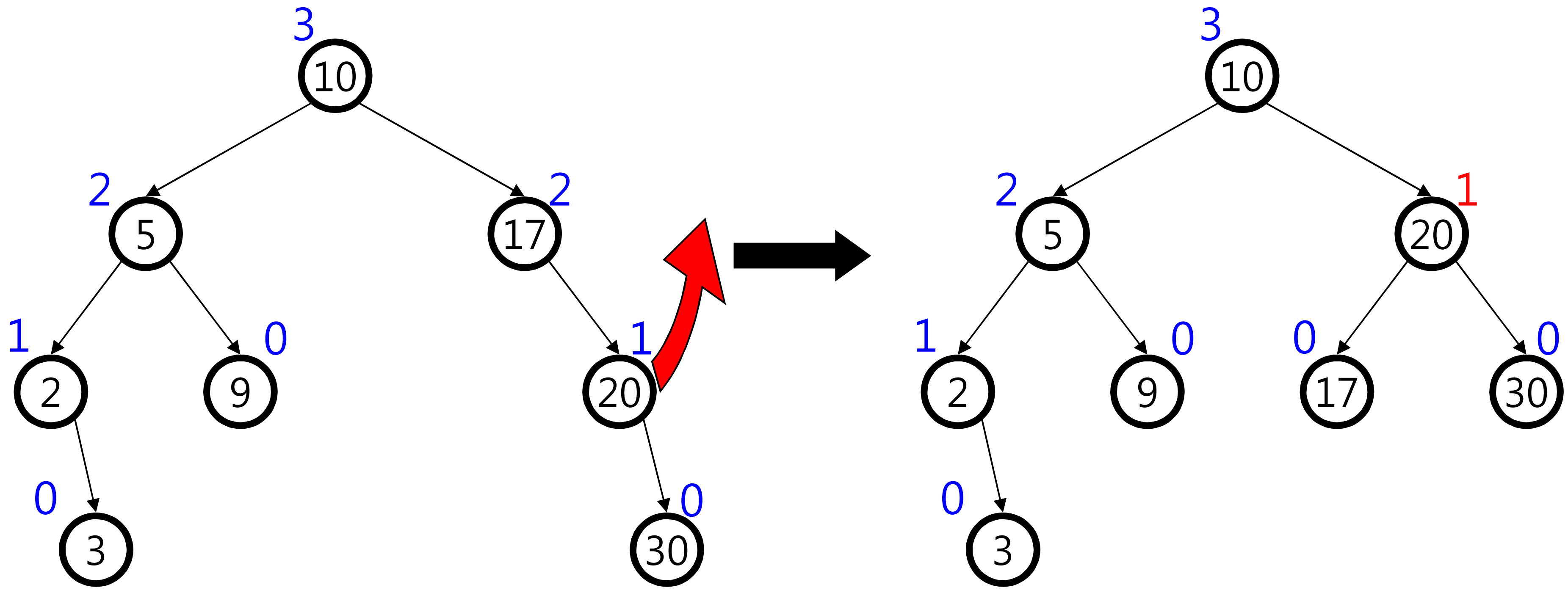


Deletion (Case #1)

Delete(12)

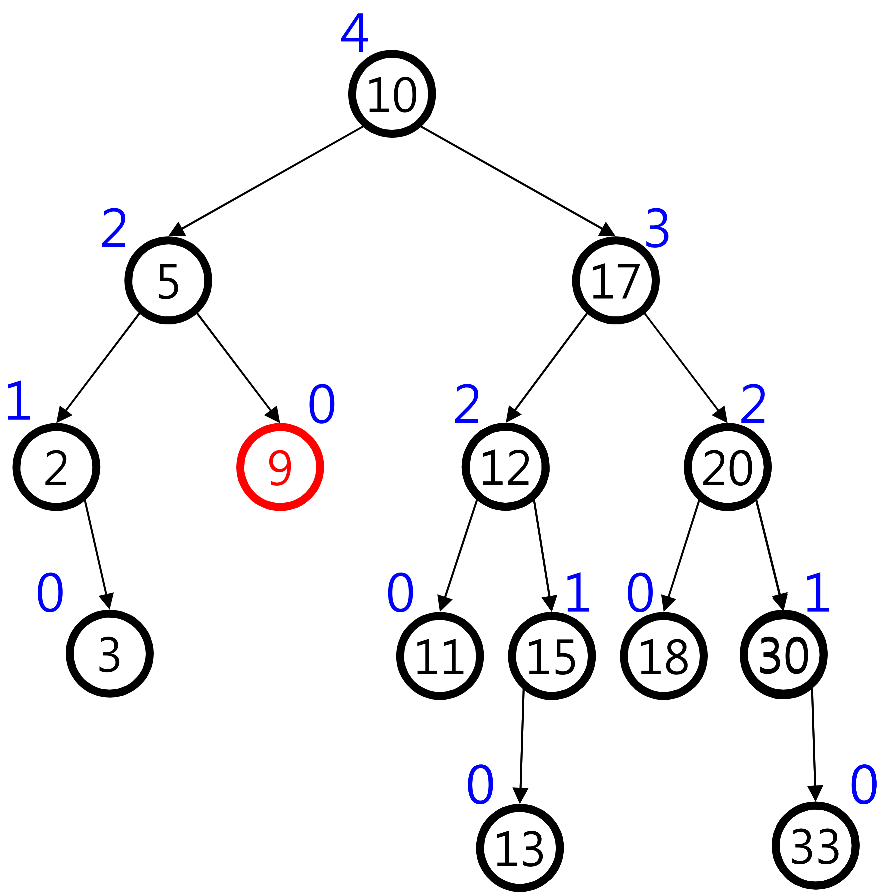


Single Rotation

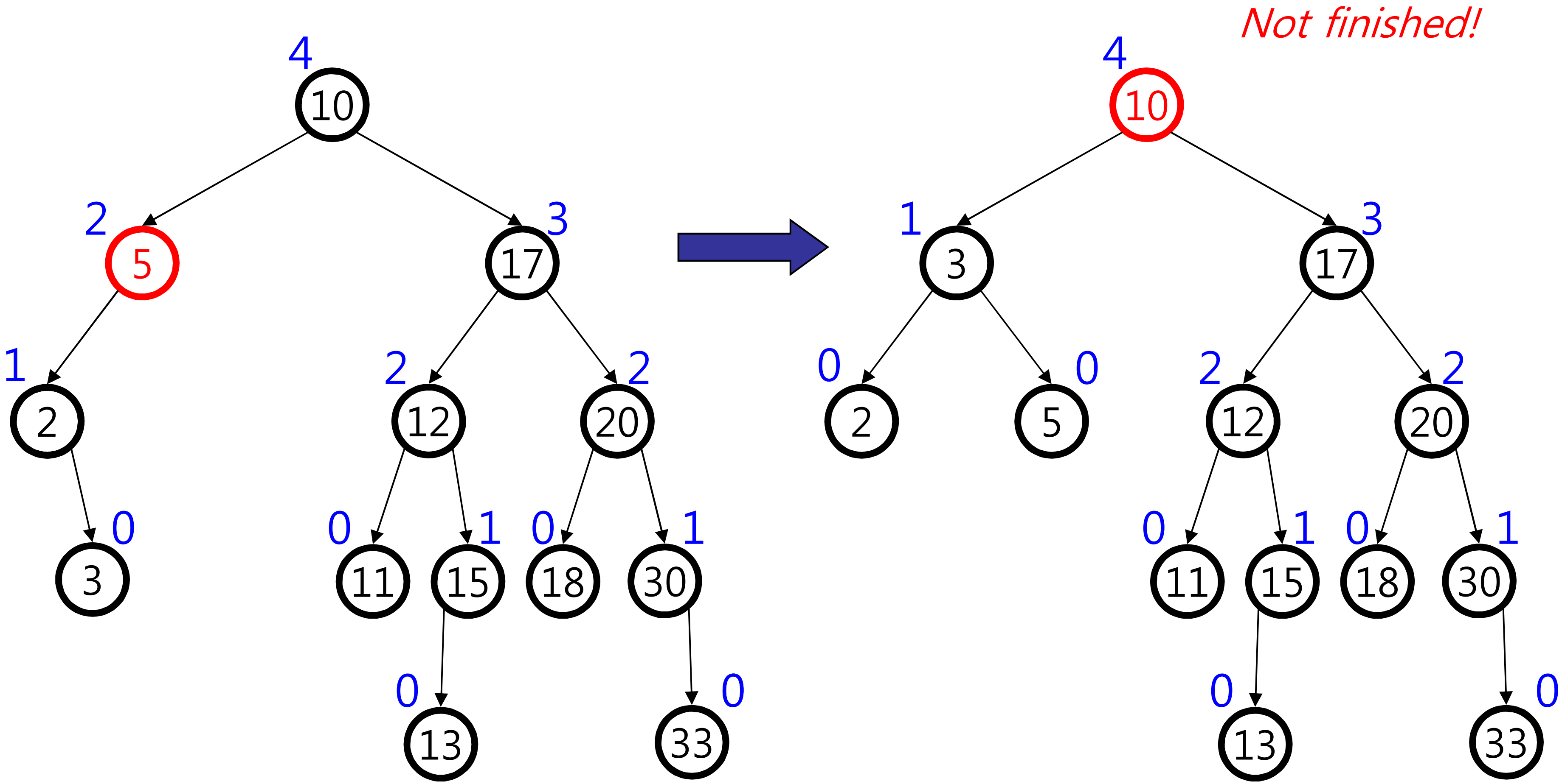


Deletion (Case #2)

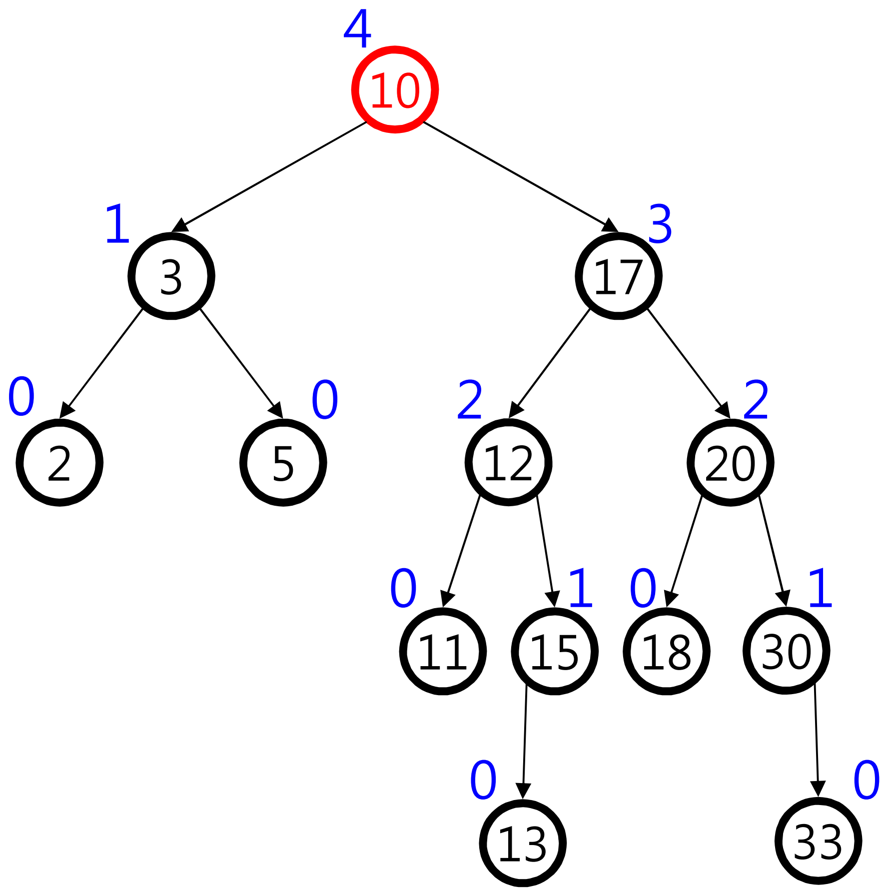
Delete(9)



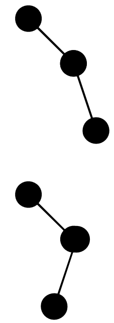
Rotation on Deletion



Deletion with Propagation

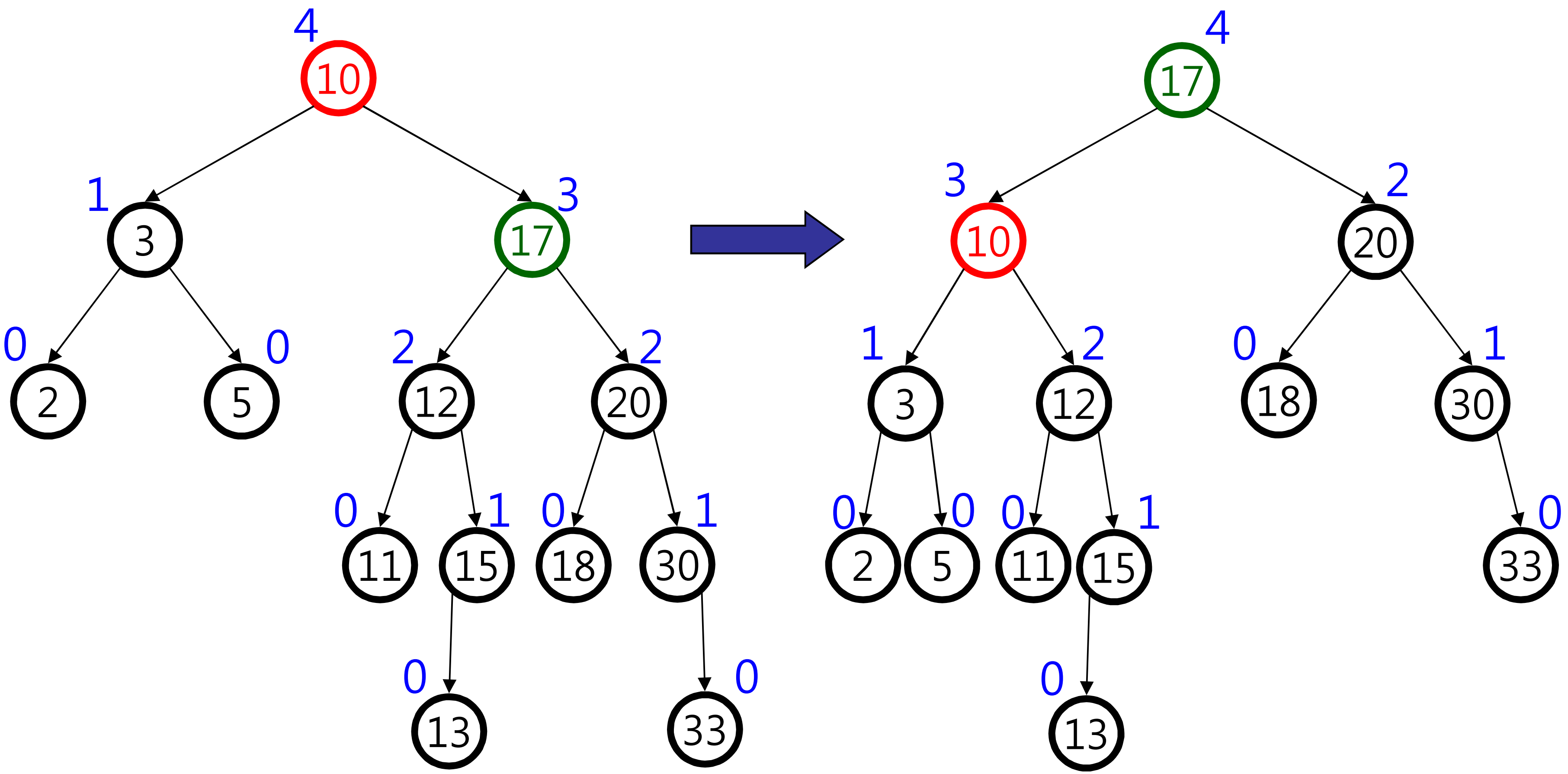


What different about this case?

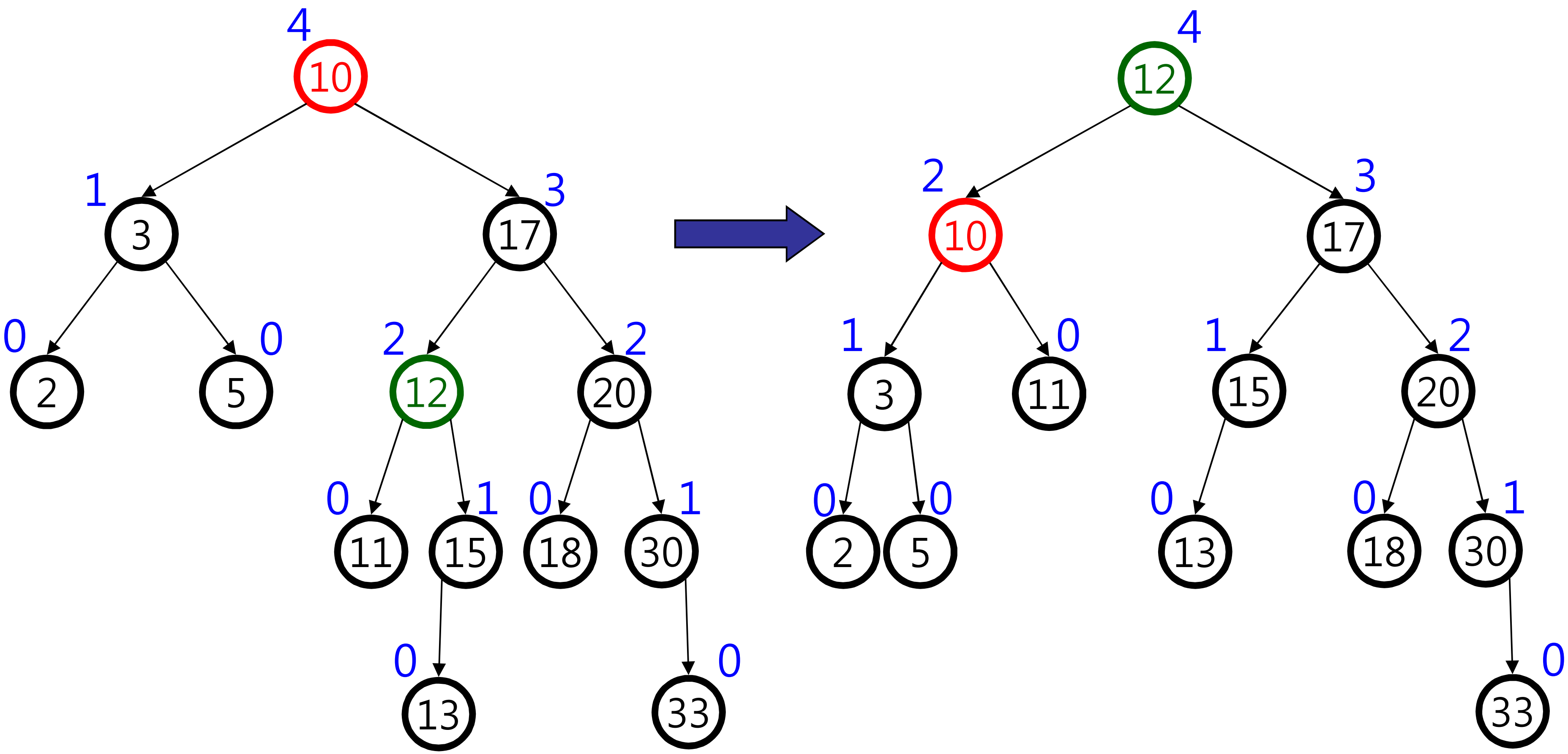


We get to choose whether to single or double rotate!

Propagated Single Rotation



Propagated Double Rotation



AVL tree analysis:

An AVL tree with n nodes has height O(log n).

Proof.
Denote the height as h. We want to show that an AVL tree with height h must have $\Omega(2^{h/2})$ nodes.

Once this is done, it follows that there is a constant $c > 0$ such that:

$$
\begin{aligned}
& n \geq c \cdot 2^{h/2} \\
\rightarrow \quad & 2^{h/2} \leq n / c \\
\rightarrow \quad & h/2 \leq \log_2(n / c) \\
\rightarrow \quad & h = O(\log n)
\end{aligned}
$$

An AVL tree with n nodes has height O(log n).

Define N(h): the least number of nodes in an AVL tree of height h.

Then,

AVL tree vs. …..

Red-Black trees -- max height: _________

constant # of rotations for insert, remove, find.

AVL trees – max height ___________

O(log n) rotations upon insert, remove.

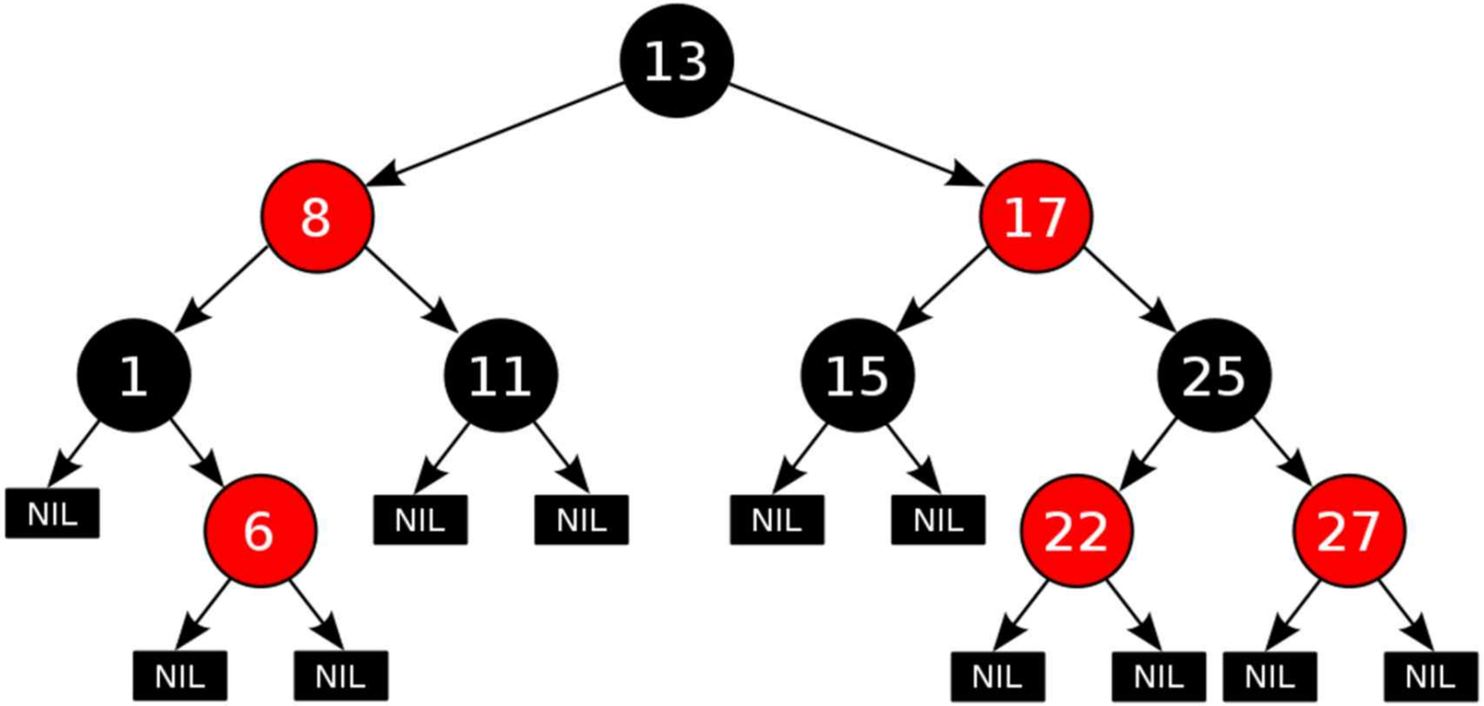
Positive:

- ✓ Insert, Remove, and Find are always O(log n)
- ✓ An improvement over: BST, Link-lists, Arrays
- ✓ Range finding & nearest neighbor (___________)

Negative:

- ✓ Possible to search for single keys faster. (_________)
- ✓ If data is so big that it doesn't fit in main memory it must be stored on disk and we require a different structure. (_________)

Red-Black tree



A red-black tree is a binary search tree which has the following *properties*:

1. Every node is either red or black.
2. Every leaf (NULL) is black.
3. If a node is red, then both its children are black.
4. Every simple path from a node to a descendant leaf contains the sam e number of black nodes.