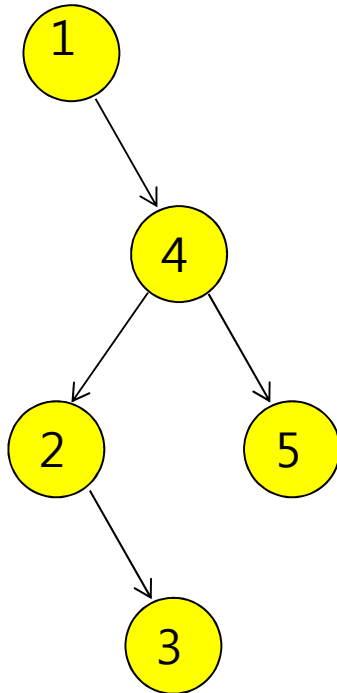




BST trees

BST의 문제 너무 많은 종류
의 트리가 만들어질 수 있다.



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 ? 

log n의 내림

$$\boxed{} \leq h \leq \boxed{n-1}$$

log n으로 트리를 만들려고 한다

Time complexity

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

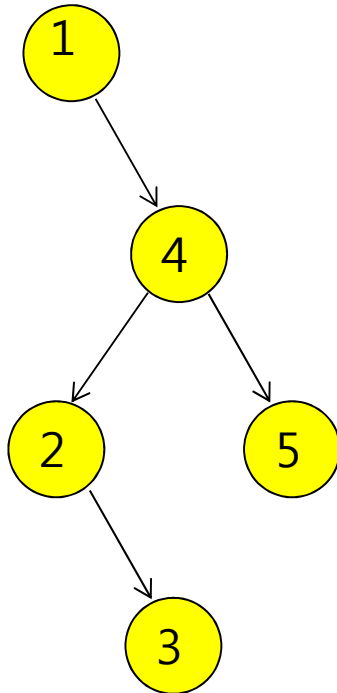
ex. 1 3 2 4 5 7 6 vs. 4 2 3 6 7 1 5

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

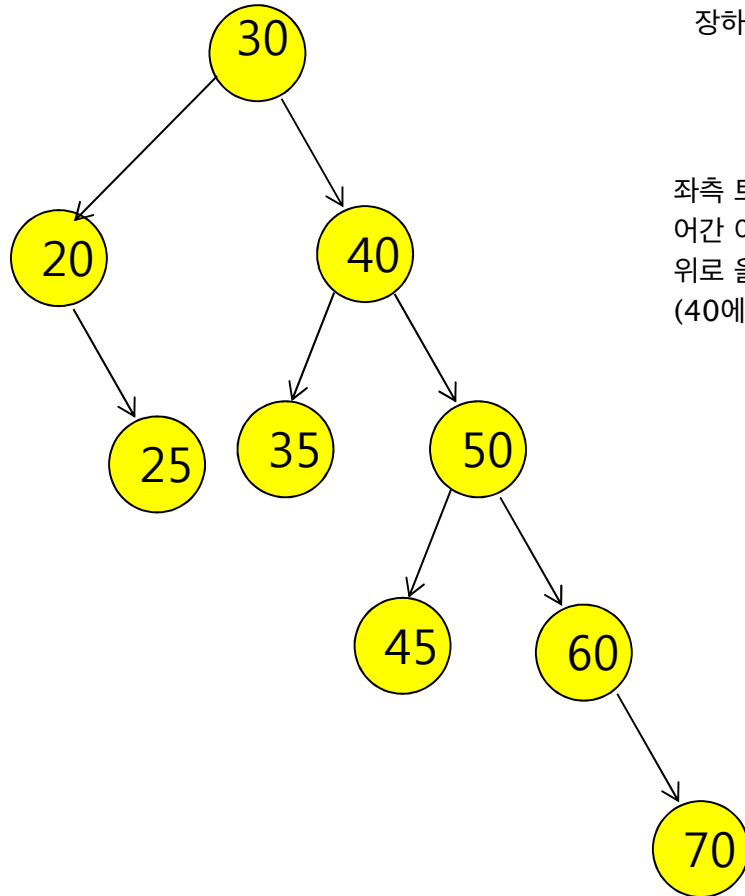
$$bf = \text{height}(T_L) - \text{height}(T_R)$$

A BST T is called height balanced if:

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

Operations to be balanced BST tree

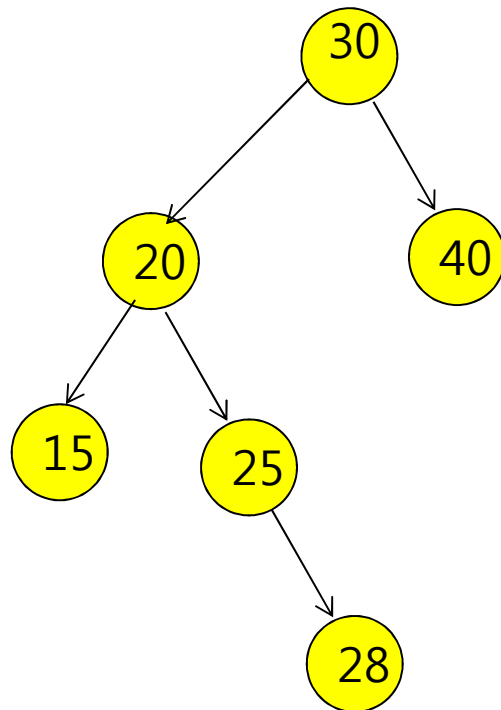
insertion 되면 1만큼 높이가 ↑ 증가하거나 감소한다. 만약 +2나 -2등 장하는 경우에만 트리를 변형시키면 된다.



좌측 트리는 70 없었을 때 avl tree 만족한다. 70들 어간 이후의 상황을 생각해보자 insertion한 70에서 위로 올라오면 40에서 처음으로 문제가 발생한다. (40에서 bf가 -2가 되므로)

문제되는 node에서 rotate 한다.

Another rotation



Rotation summary:

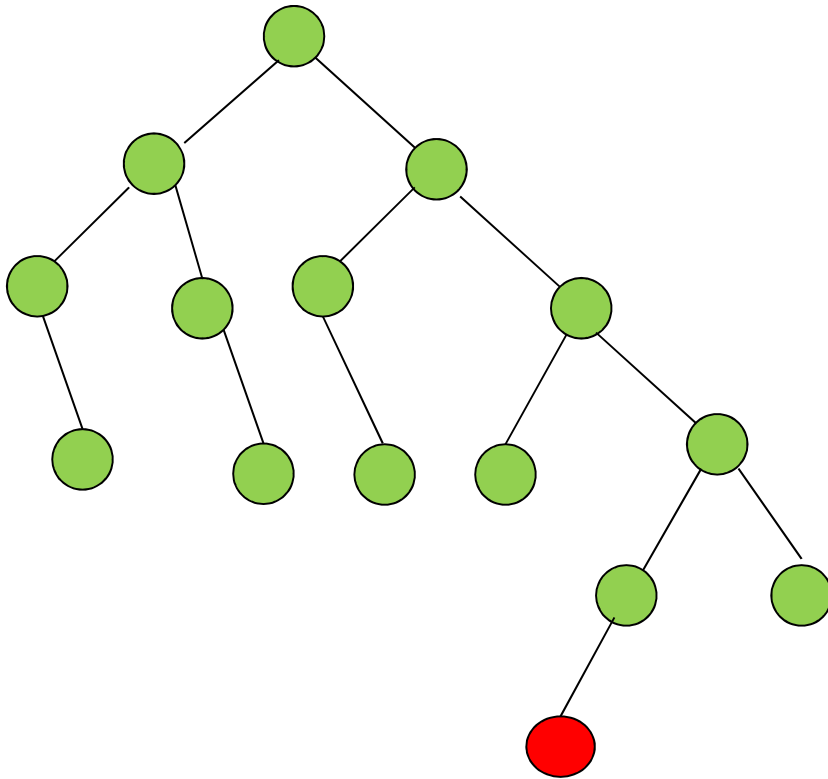
- 4 kinds: left, right, left-right, right-left
- $O(1)$ time and local operations
- Still BST maintained

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:

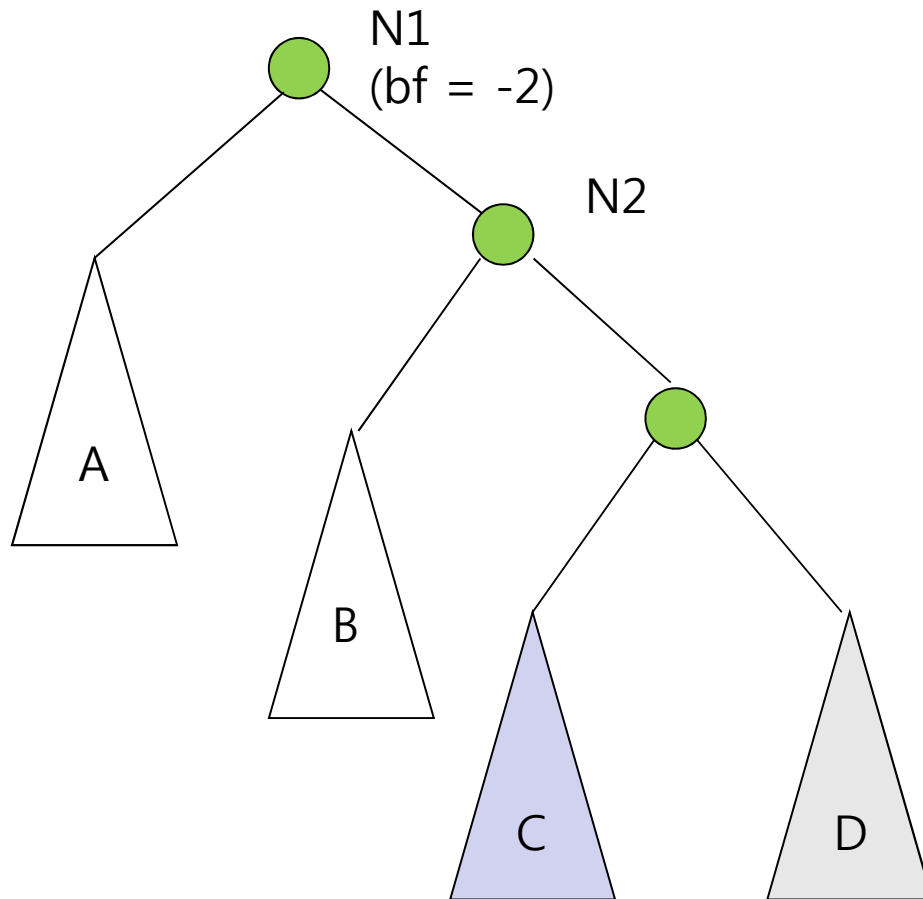


Insert:

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

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

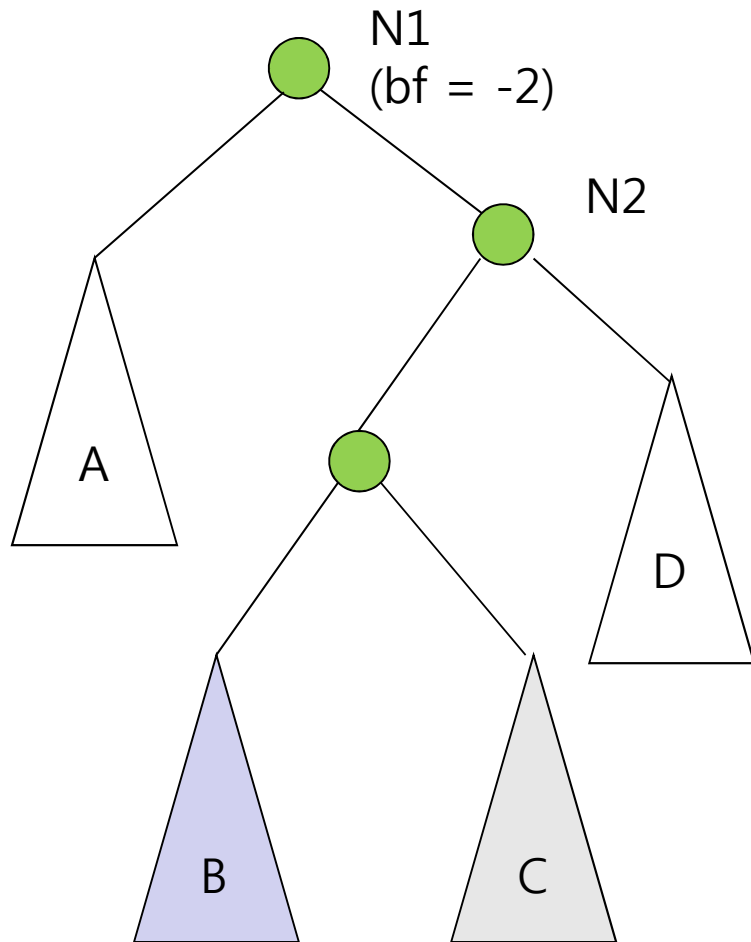
Identifying single rotation



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 _____

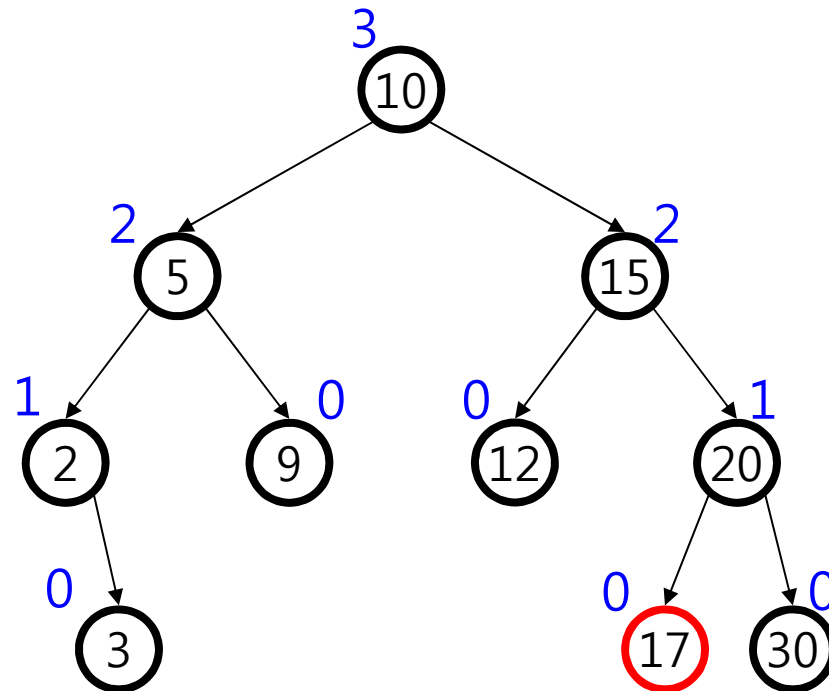
```

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;
}

```

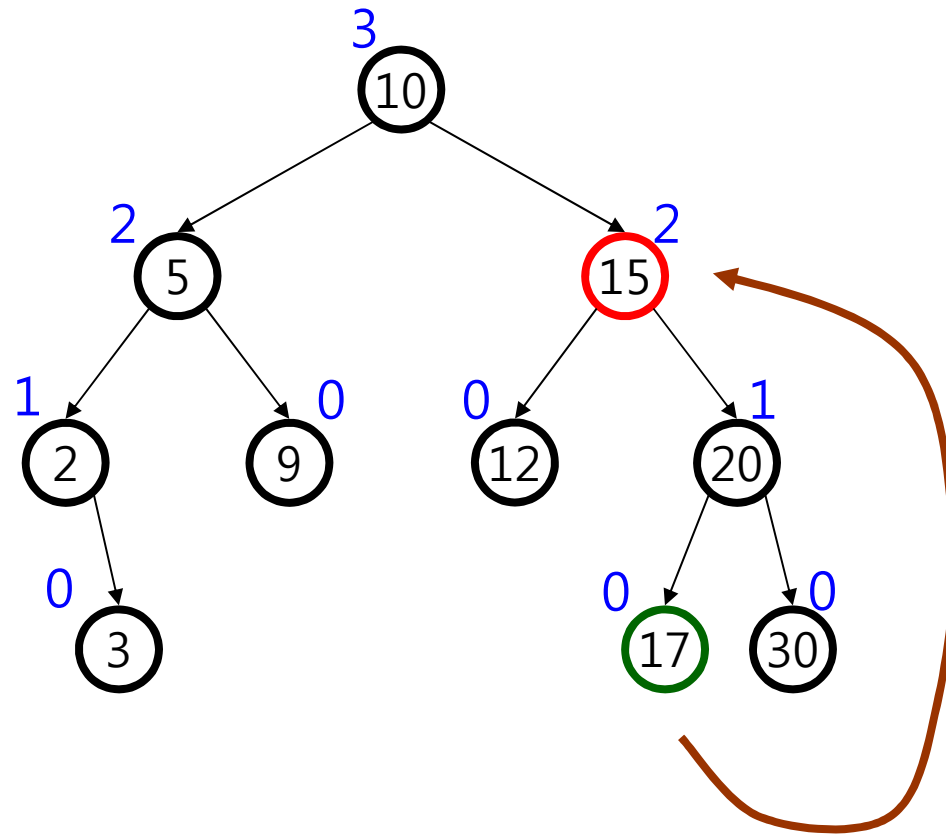
Deletion (no rotation)

Delete(17)



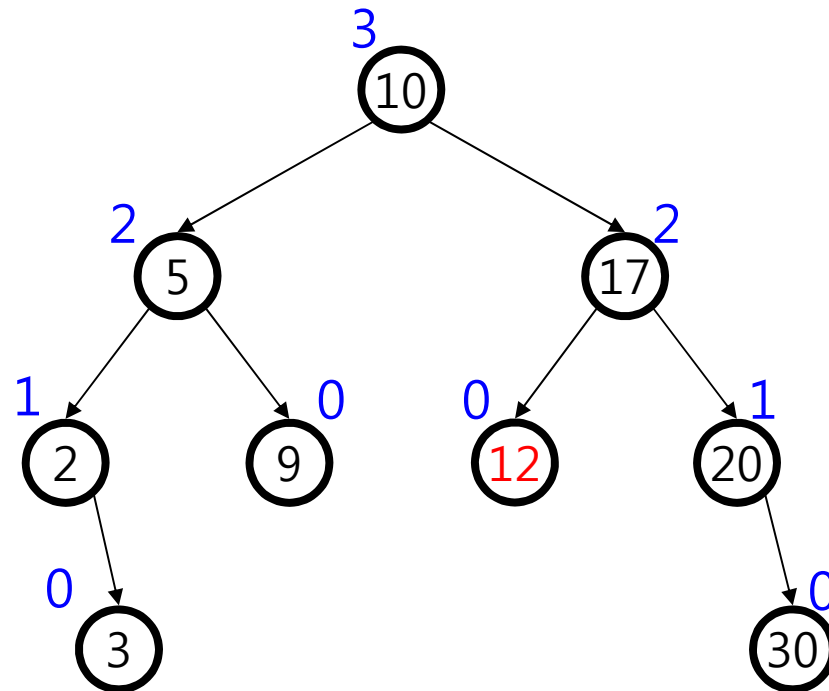
Deletion (no rotation)

Delete(15)

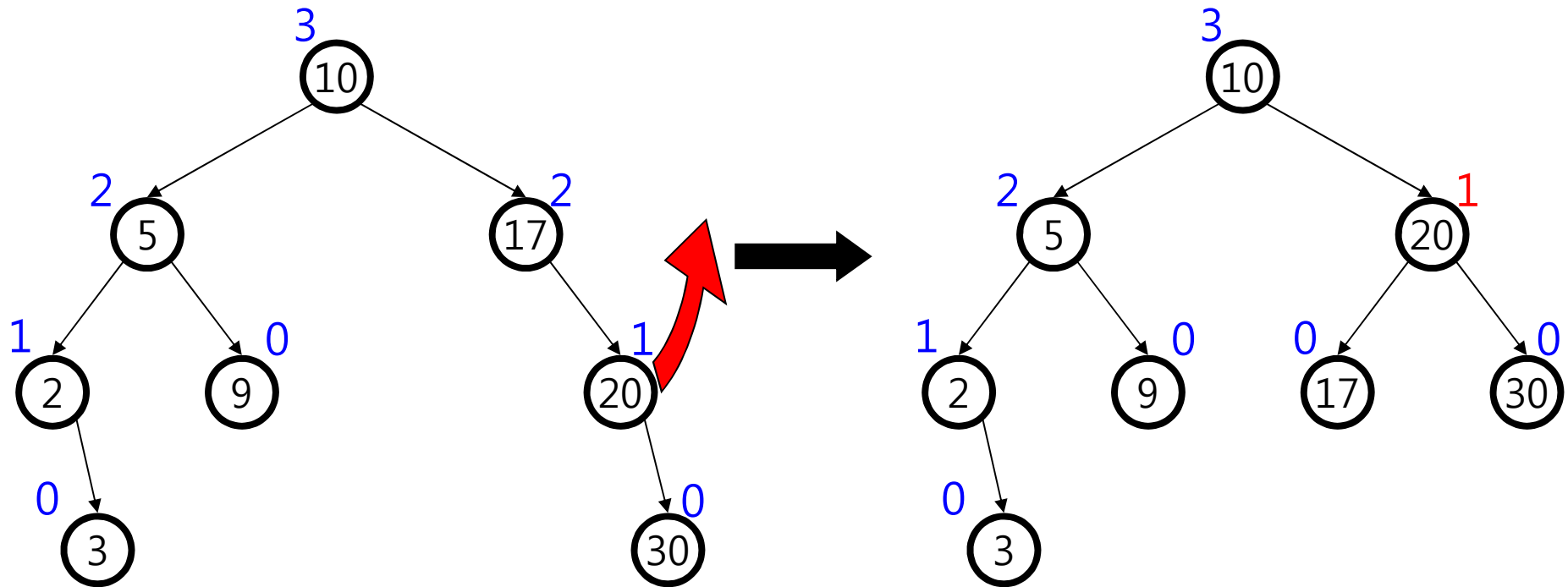


Deletion (Case #1)

Delete(12)

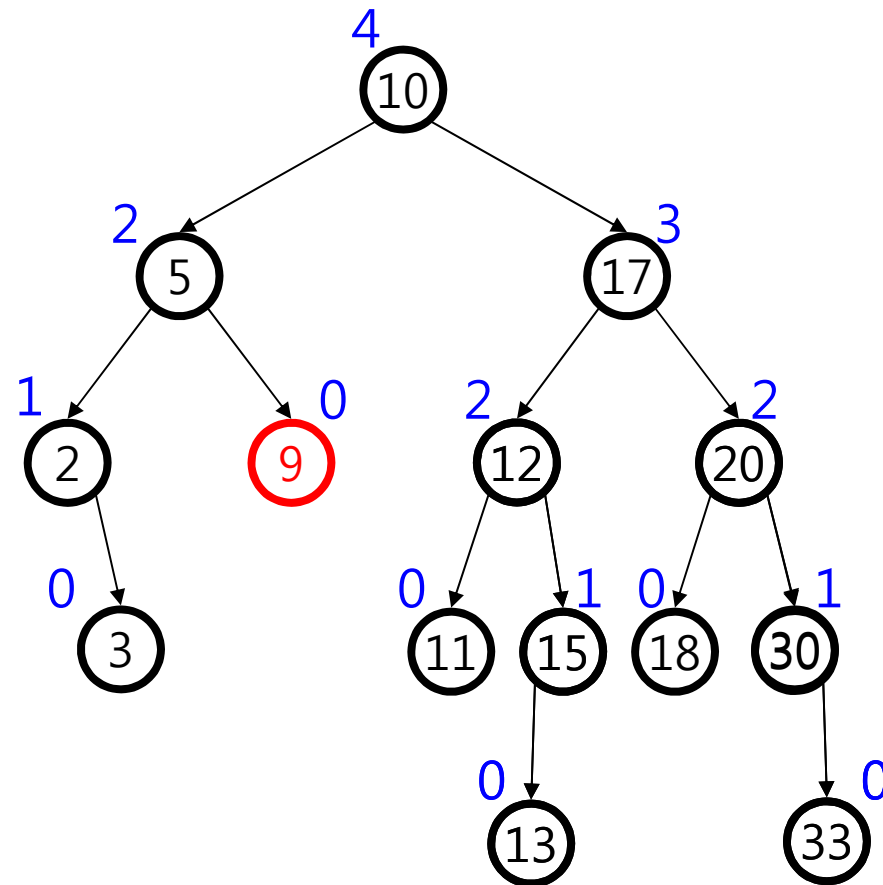


Single Rotation

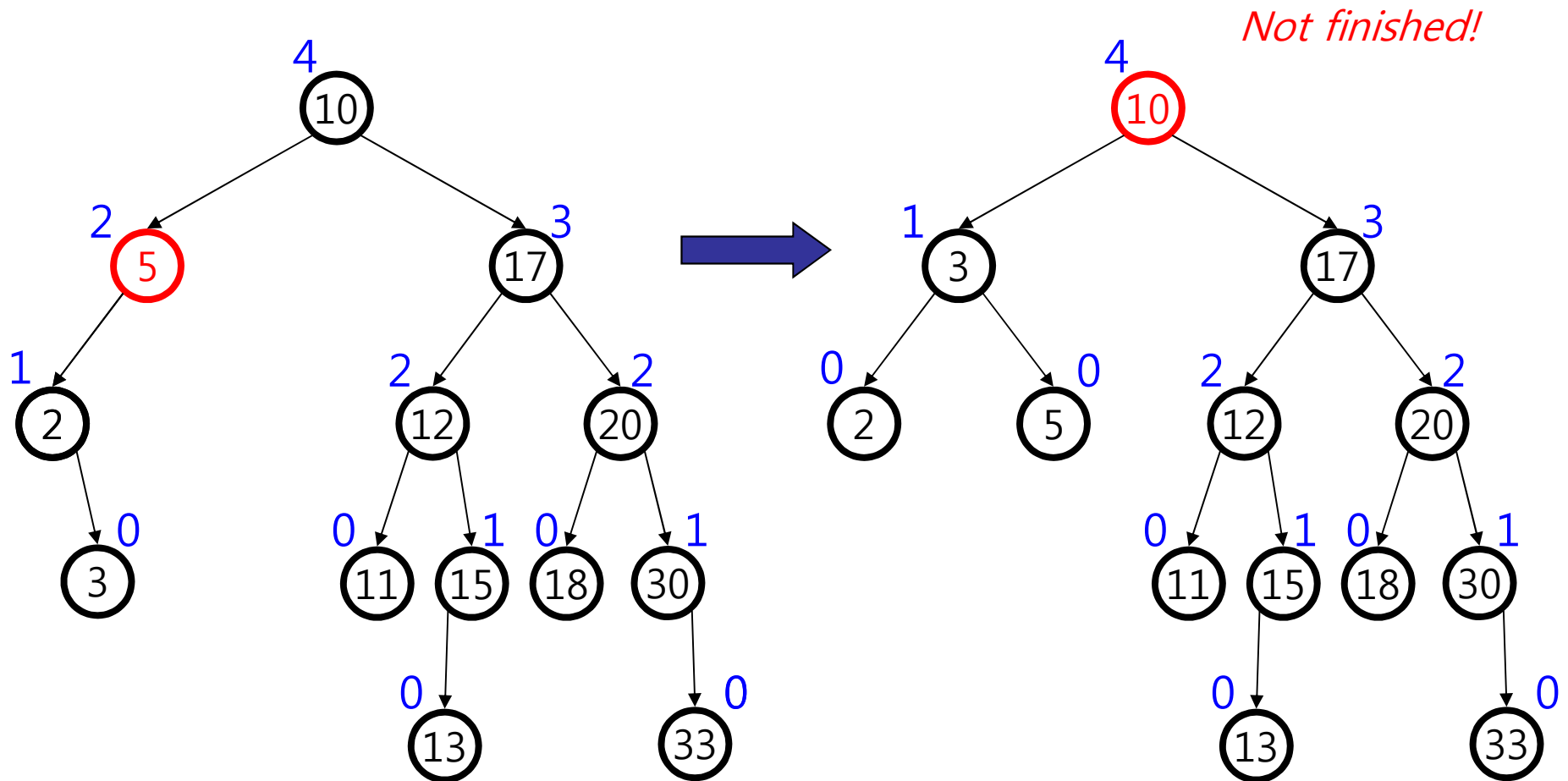


Deletion (Case #2)

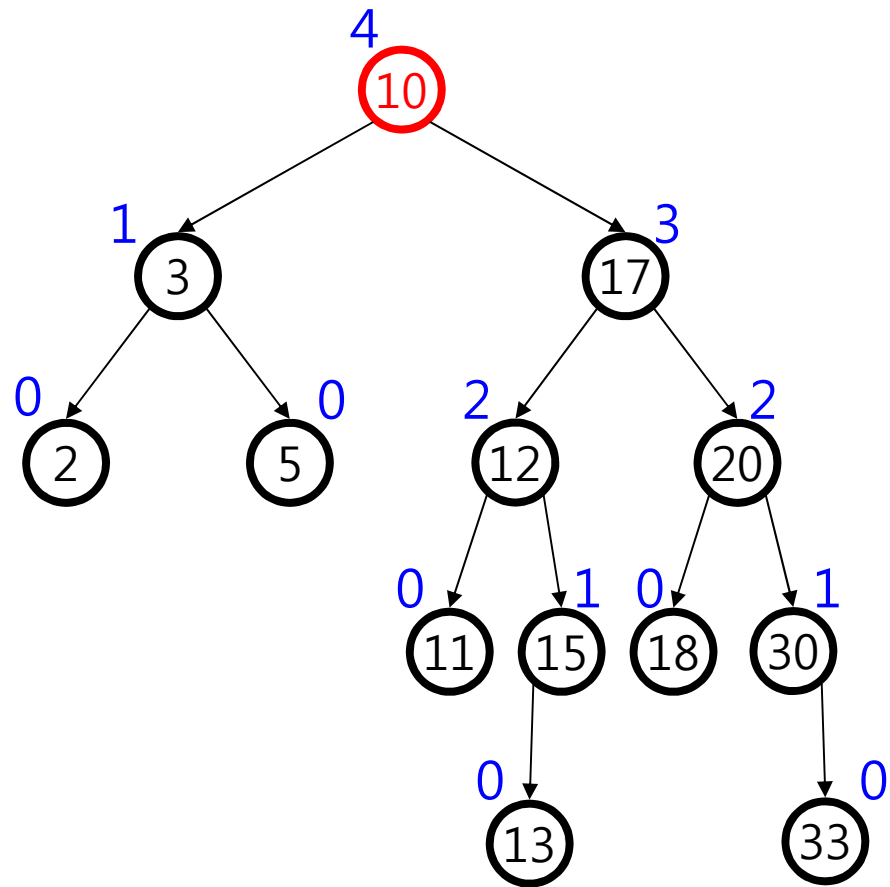
Delete(9)



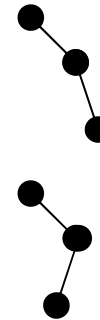
Rotation on Deletion



Deletion with Propagation

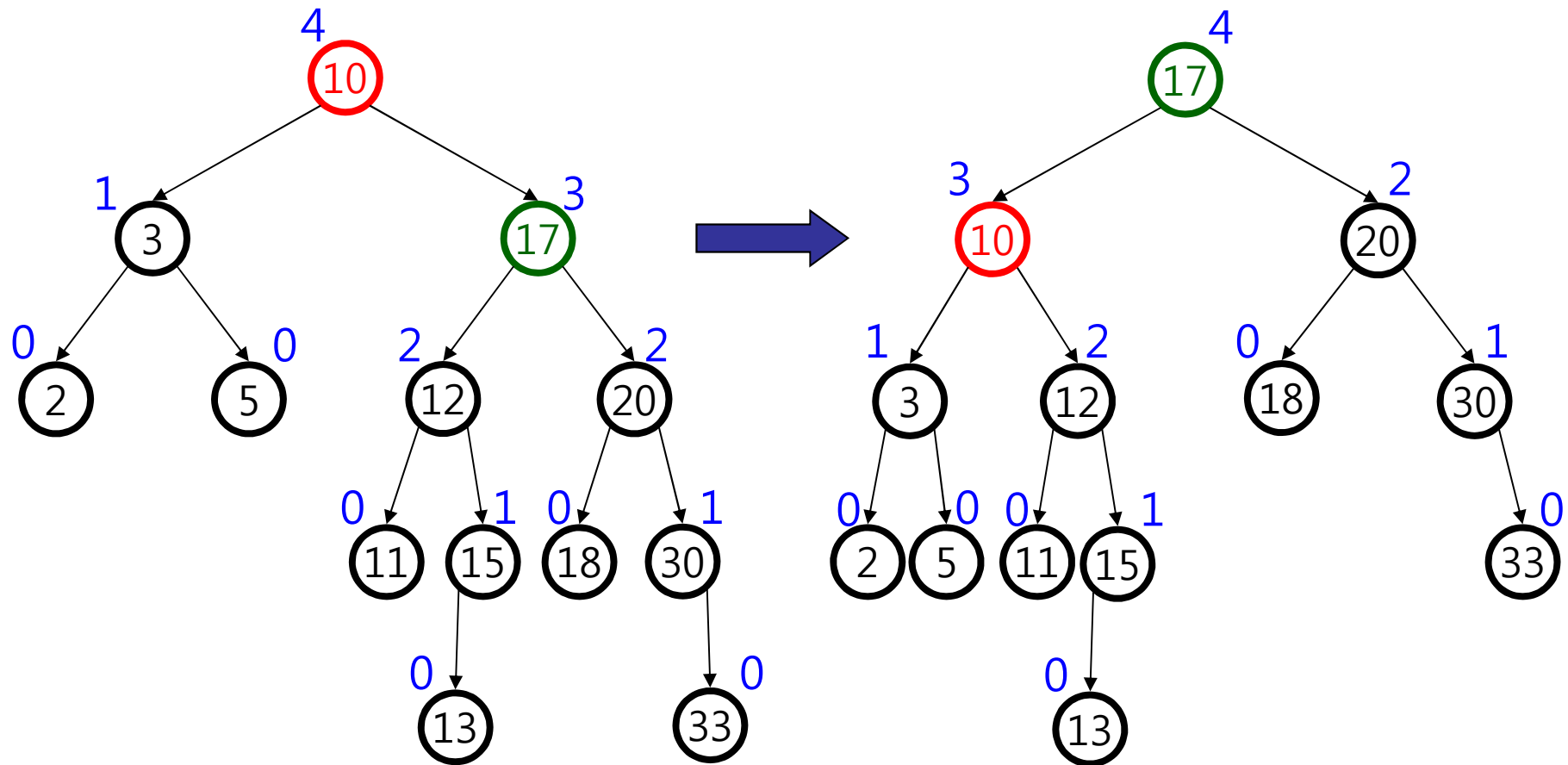


What different about this case?

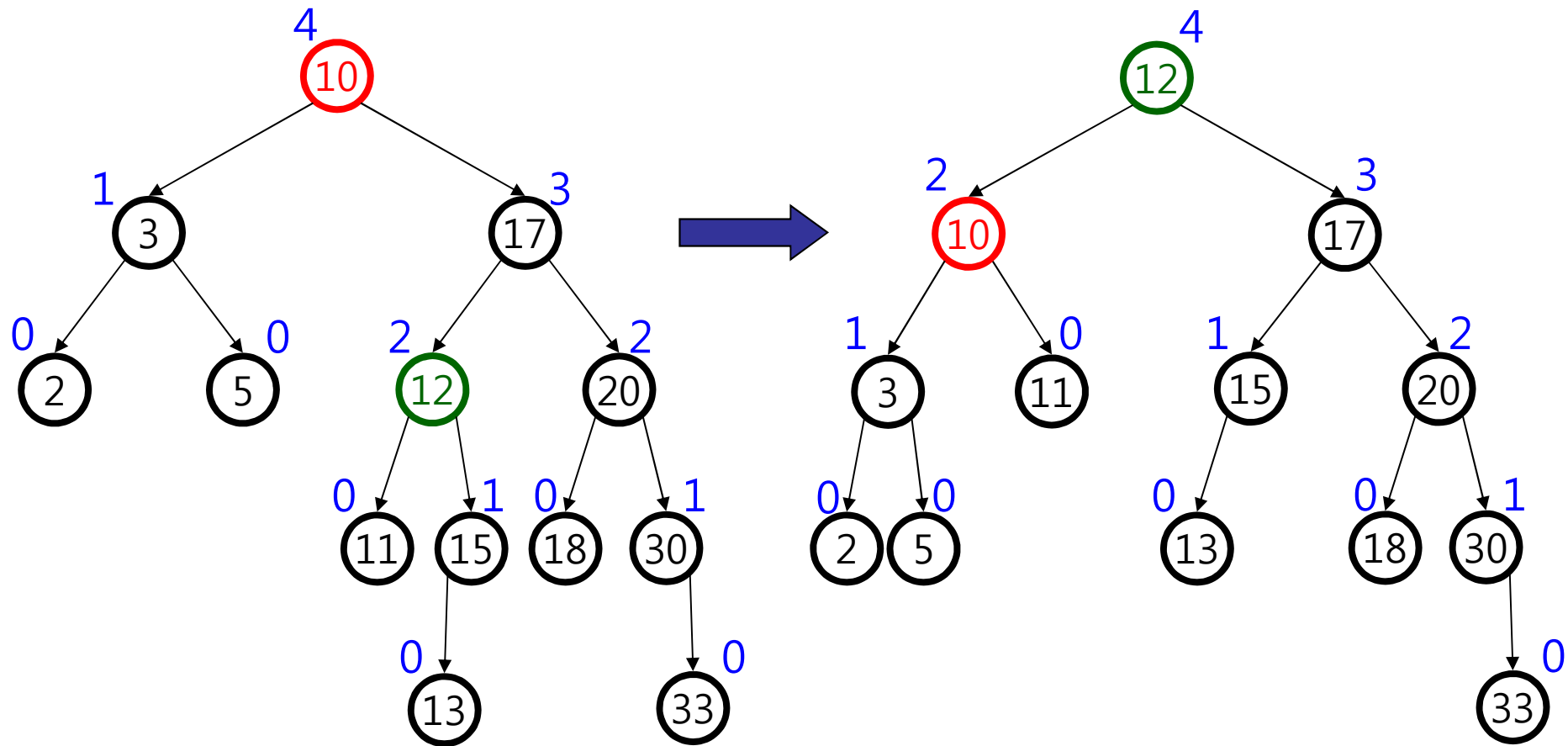


We get to choose whether to single or double rotate!

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:

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

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.

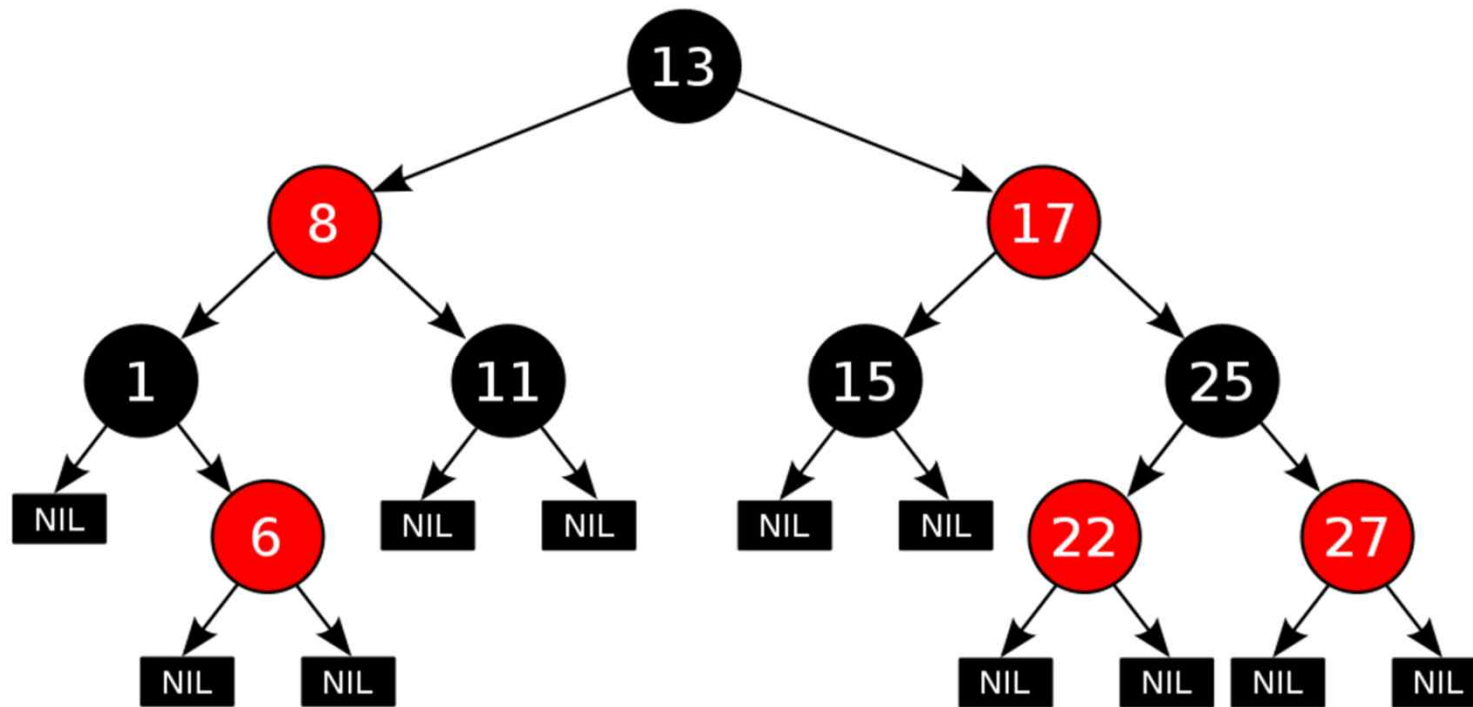
Positive:

- ✓ 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 same number of black nodes.