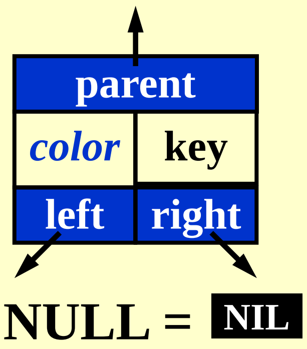
Red-Black Trees

and B+ Trees

Red-Black Trees



Target: **Balanced binary search tree**

【Definition】A red-black tree is a binary search tree that satisfies the following *red-black properties*:

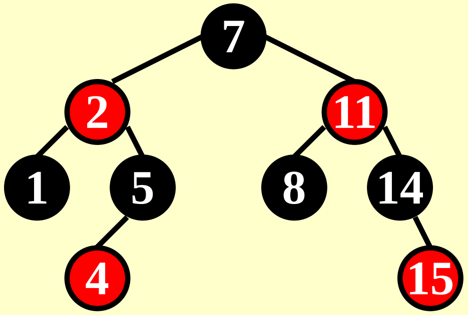
(1) Every node is either red or black.

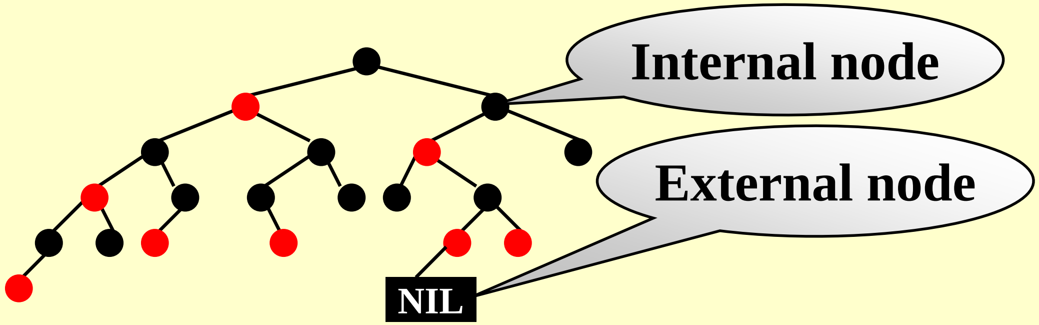
(2) The root is black.

(3) Every leaf (NIL) is black.

(4) If a node is red, then both its children are black.

(5) For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.





【Definition】 **The black-height of any node x, denoted by bh(x), is the number of black nodes on any simple path from x (x not incl ee) = bh(root).**

Number of internal nodes in the subtree rooted at *x*

【Lemma】 **A red as height at most 2ln(N +1).**

Proof: ① For any node x, sizeof(x) $\geq 2^{\text{bh}(x)} - 1$. Prove by induction.

If $h(x) = 0$, x is NULL ⟹ sizeof(x) $= 2^0 - 1 = 0$ ✓

Suppose it is true for all x with $h(x) \leq k$.

For x with $h(x) = k + 1$, bh($child$) = bh(x) or bh(x) – 1

Since $h(child) \leq k$, sizeof($child$) $\geq 2^{\text{bh}(child)} - 1 \geq 2^{\text{bh}(x) - 1} - 1$

Hence sizeof(x) = 1 + 2sizeof($child$) $\geq 2^{\text{bh}(x)} - 1$ ✓

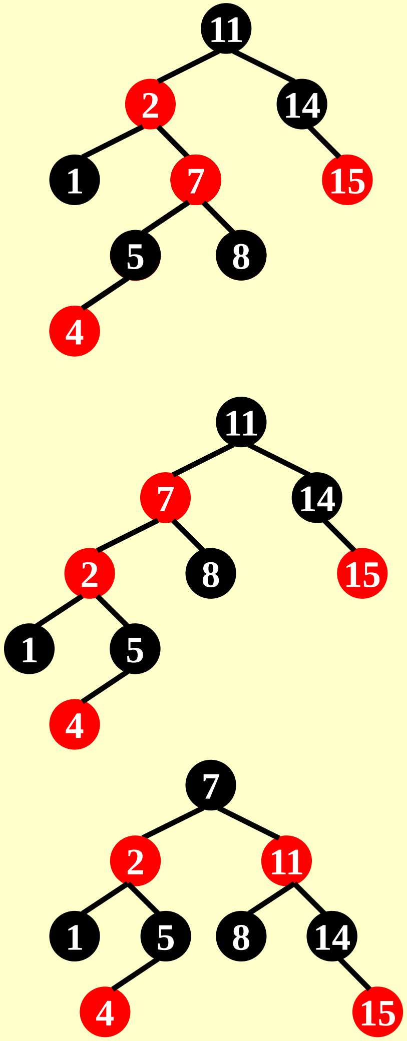
② bh($Tree$) $\geq h(Tree) / 2$?

Discussion 2: Please finish the proof.

☞ Insert — can be done *iteratively*

Sketch of the idea: Insert & color red

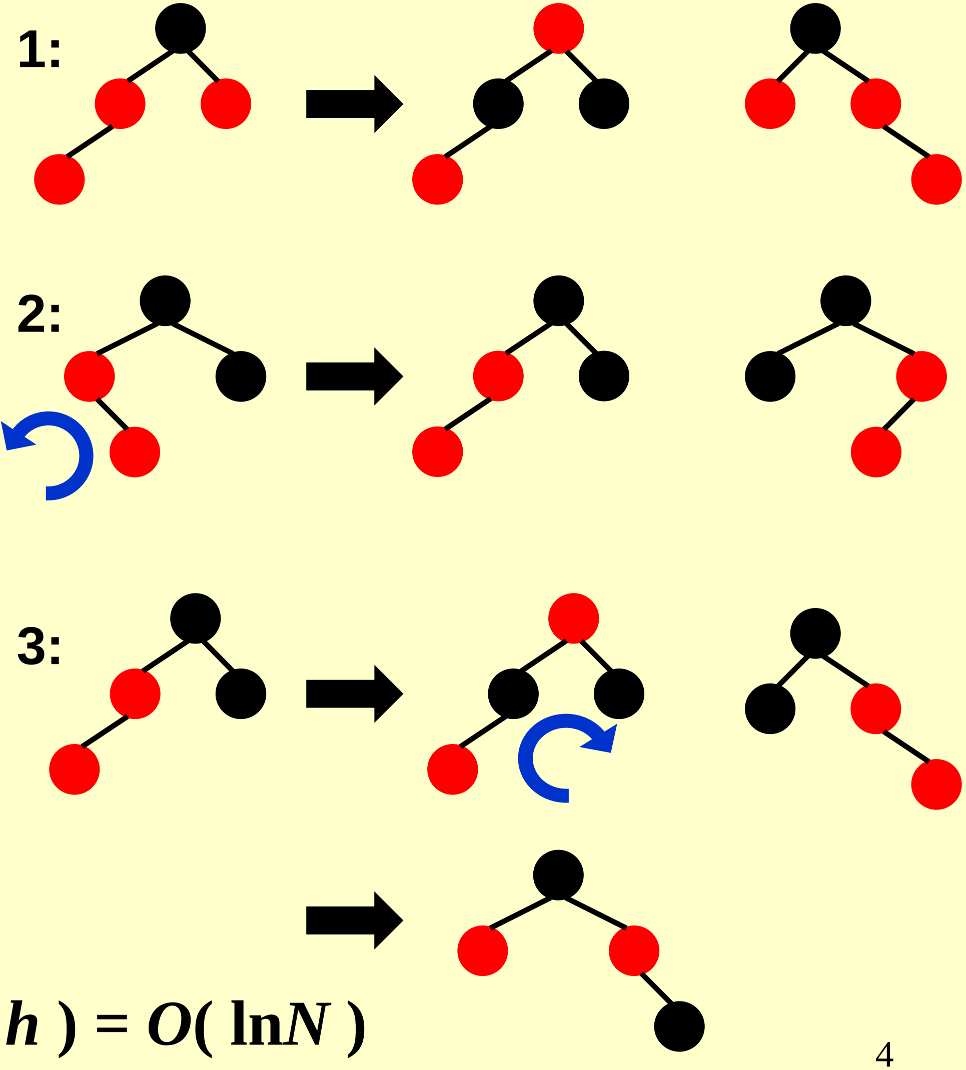
Symmetric



Case 1:

Case 2:

Case 3:



$$T = O(\,h\,) = O(\,\ln N\,)$$

☞ Delete

- ❖ **Delete a leaf node :** Reset its parent link to NIL.
- ❖ **Delete a degree 1 node :** Replace the node by its single child.
- ❖ **Delete a degree 2 node :**

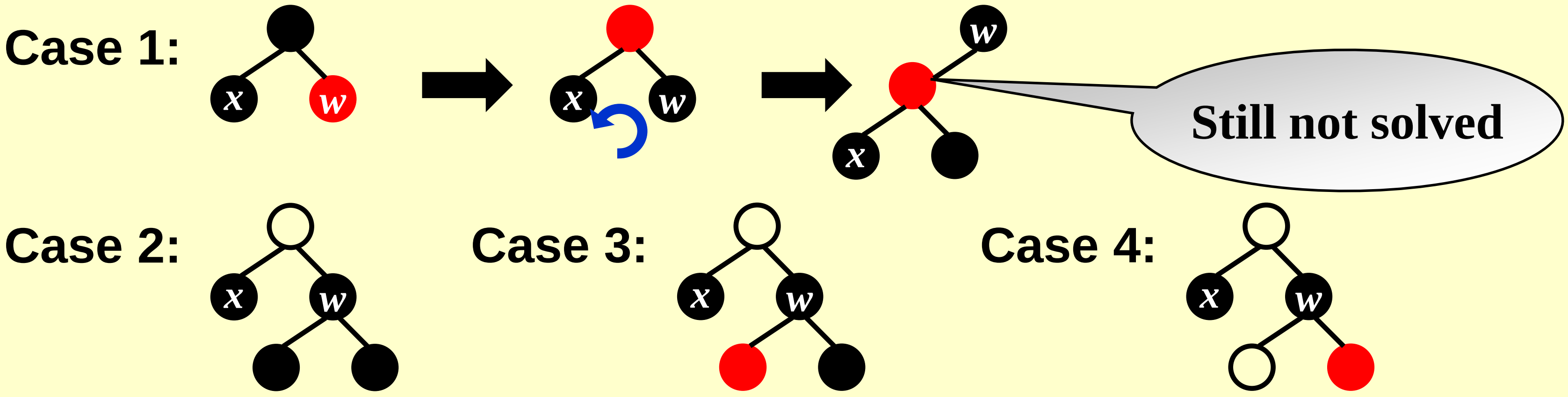
  ① Replace the node by the **largest** one in [illegible] subtree **or** the **smallest** one in its **right** subtree.

  ② Delete the replacing node from the subtree.

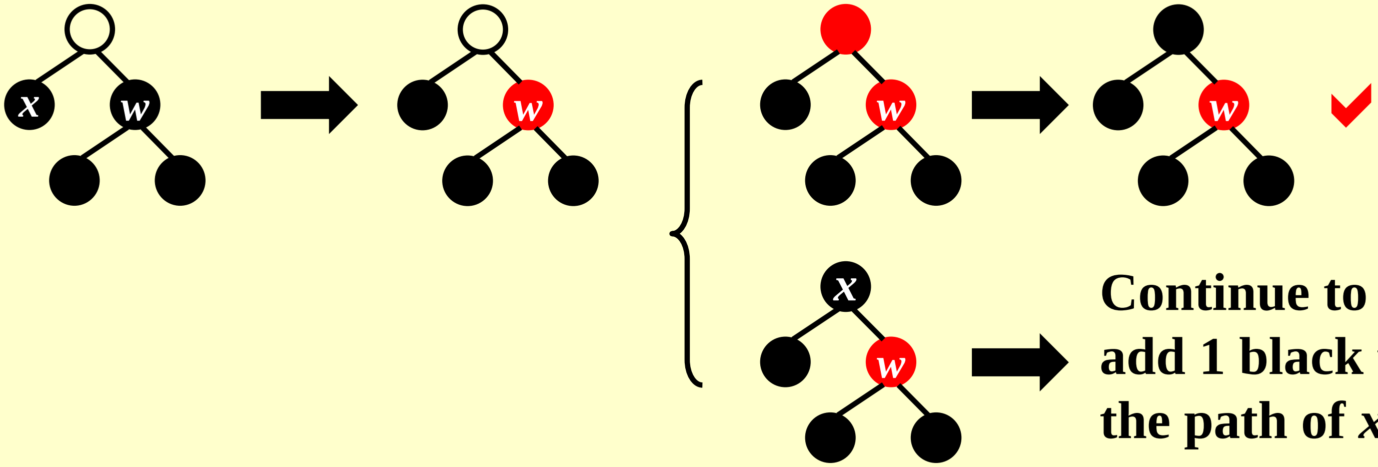
Adjust only if the node is black.

Keep the color

Must *add 1 black* to the path of the replacing node.

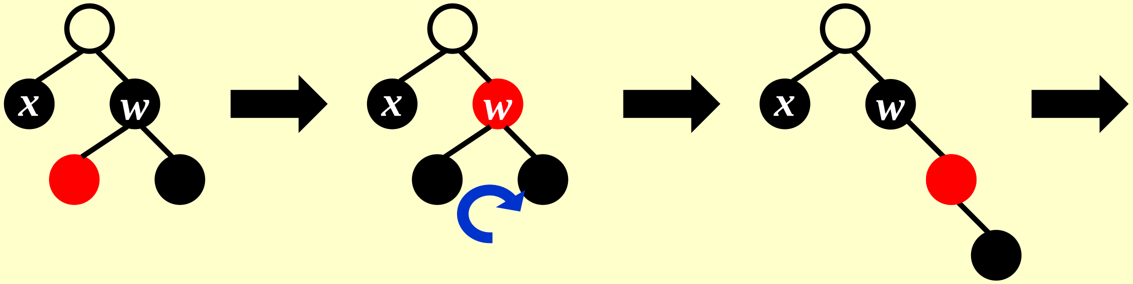


Case 2:



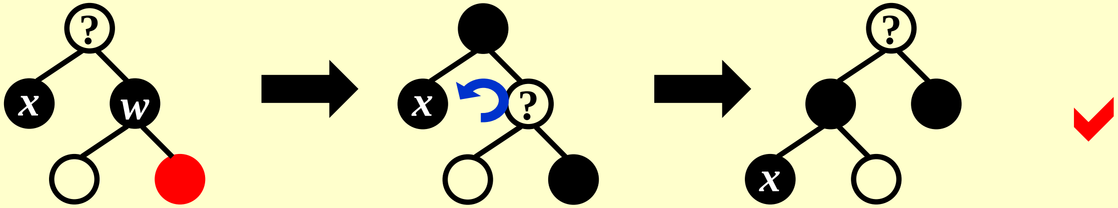
Continue to add 1 black to the path of *x*

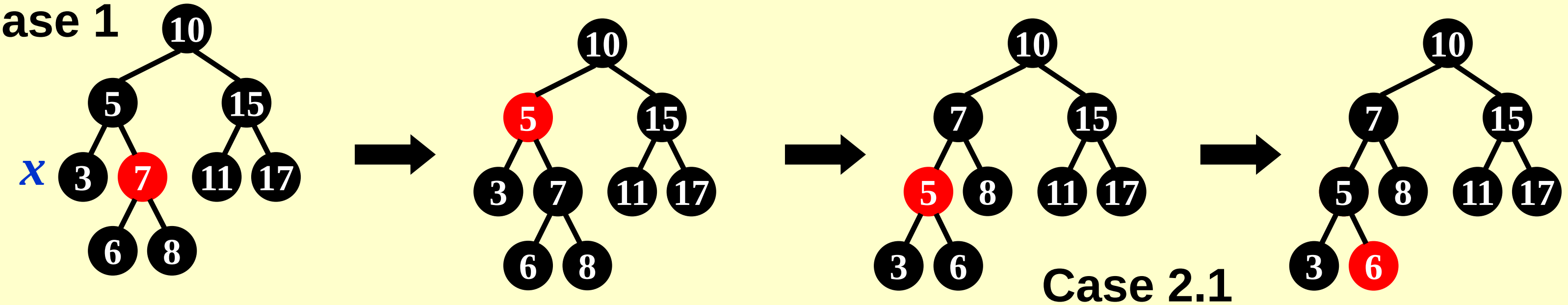
Case 3:



Case 4

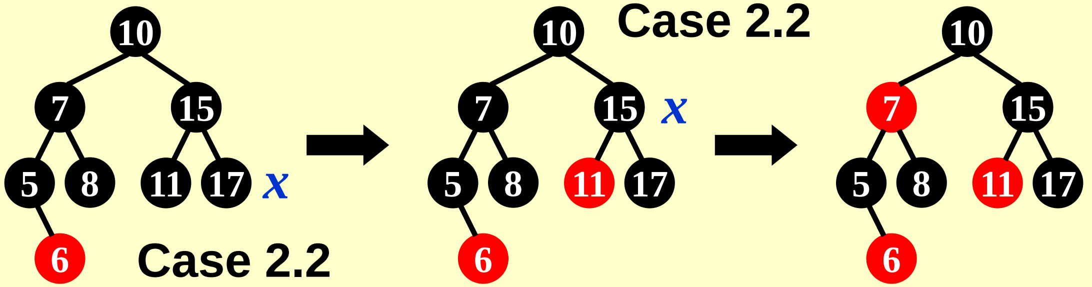
Case 4:





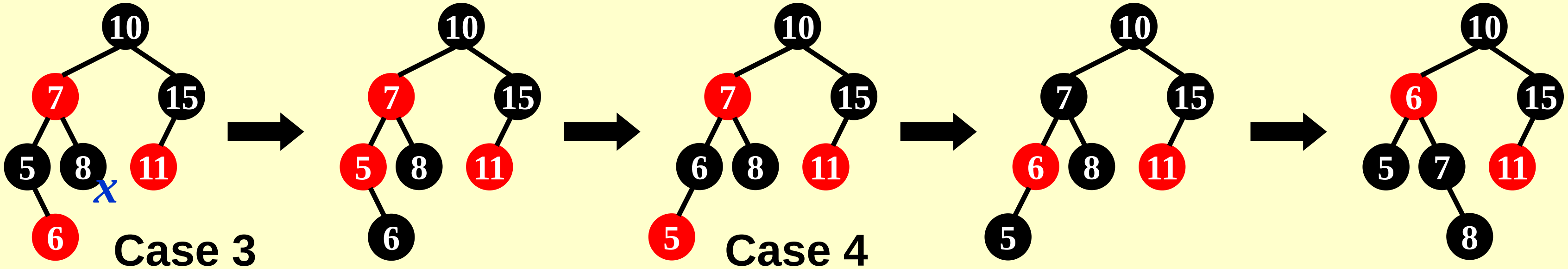
Case 1

Case 2.1



Case 2.2

Case 2.2



Case 3

Case 4

B+ Trees

【Definition】 A **B+ tree** of order **M** is a tree with the following structural properties:

(1) The root is either a leaf or has **between 2 and M children**.

(2) All nonleaf nodes (except the root) have **between ⌈M/2⌉ and M children**.

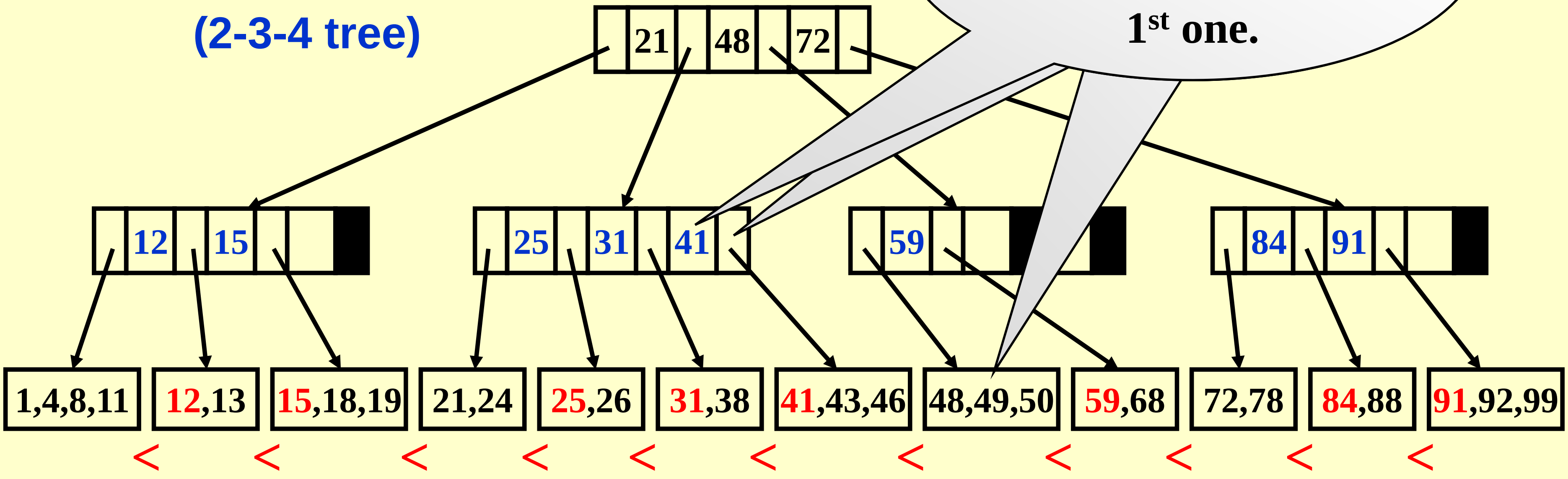
(3) All leaves are at the **same depth**.

Assume each nonroot leaf also has between ⌈M/2⌉

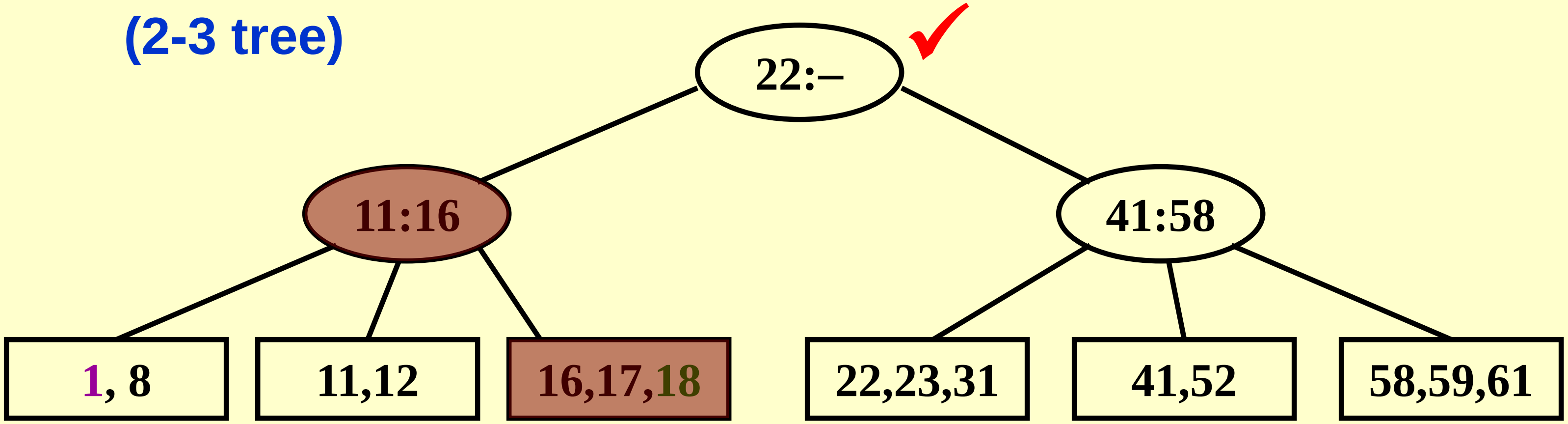
And M – 1 smallest key values in the subtrees except the 1st one.

A B+ tree of order 4

(2-3-4 tree)

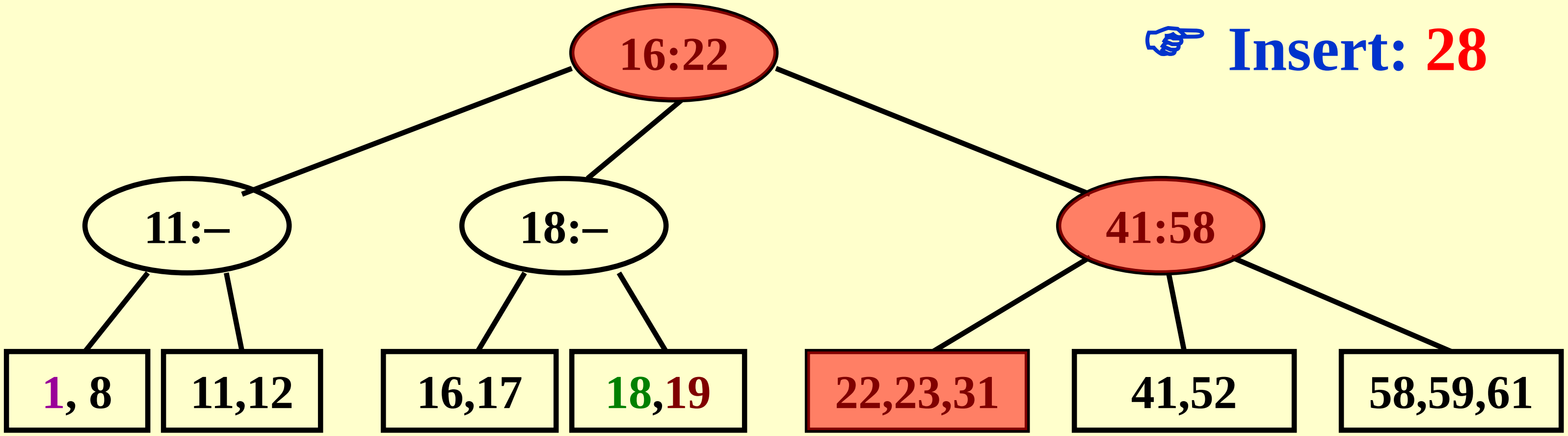


A B+ tree of order 3
(2-3 tree)

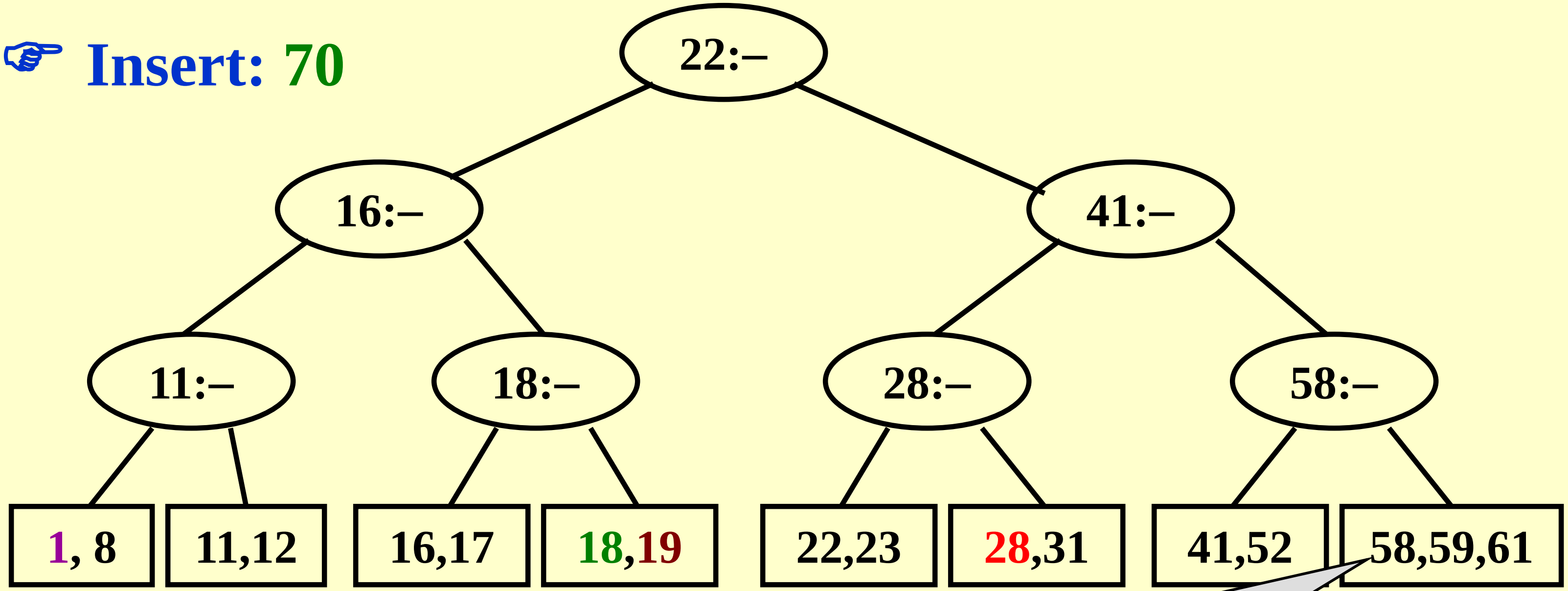


☞ Find: 52 ☞ Insert: 18 ☞ Insert: 1 ☞ Insert: 19

☞ Insert: 28



☞ Insert: 70



First find a sibling with 2 keys and adjust. Keep more nodes full.

☞ Deletion is similar to insertion except that the root is removed when it loses two children.

For a general B+ tree of order M

$T = O(M)$

```
Btree  Insert ( ElementType X,  Btree T )
{
    Search from root to leaf for X and find the proper leaf node;
    Insert X;
    while ( this node has M+1 keys ) {
        split it into 2 nodes with $\lceil (M+1)/2 \rceil$ and $\lfloor (M+1)/2 \rfloor$ keys, respectively;
        if (this node is the root)
            create a new root with two children;
        check its parent;
    }
}
```

$$T(M, N) = O(\ (M/\log M) \log N\ )$$

$$\text{Depth}(M, N) = \quad O(\lceil \log_{\lceil M/2 \rceil} N \rceil)$$

$$T_{Find}(M, N) = O(\ \log N\ )$$

Note: The best choice of M is 3 or 4.

Reference:

Introduction to Algorithms, 3rd Edition: Ch.13，p. 308-338；Ch.18，p. 484-504； ***Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009***