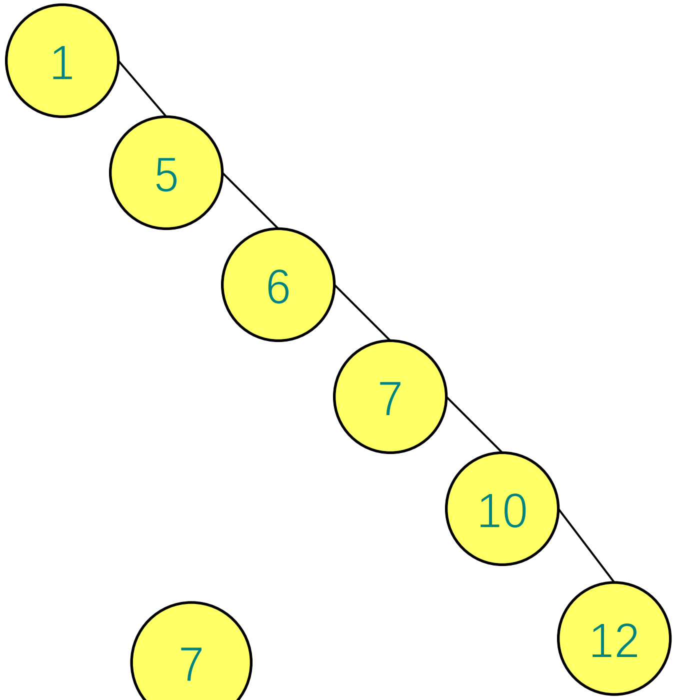
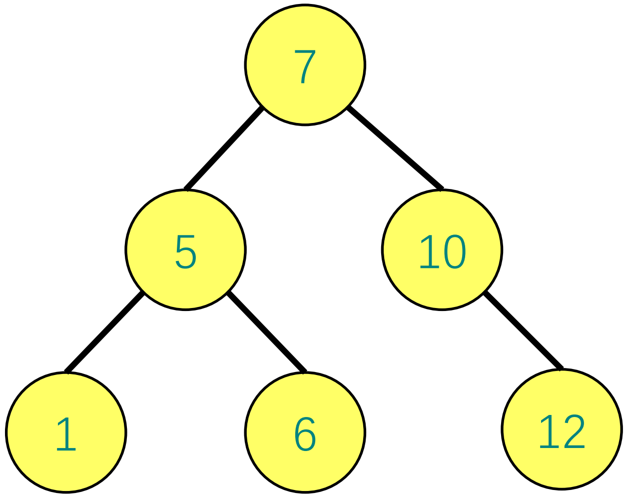
Balanced BSTs: AVL Trees

Overview

- Review: Binary Search Trees
- Importance of being balanced
- Balanced BSTs
    - AVL trees
        - definition
        - rotations, insertion





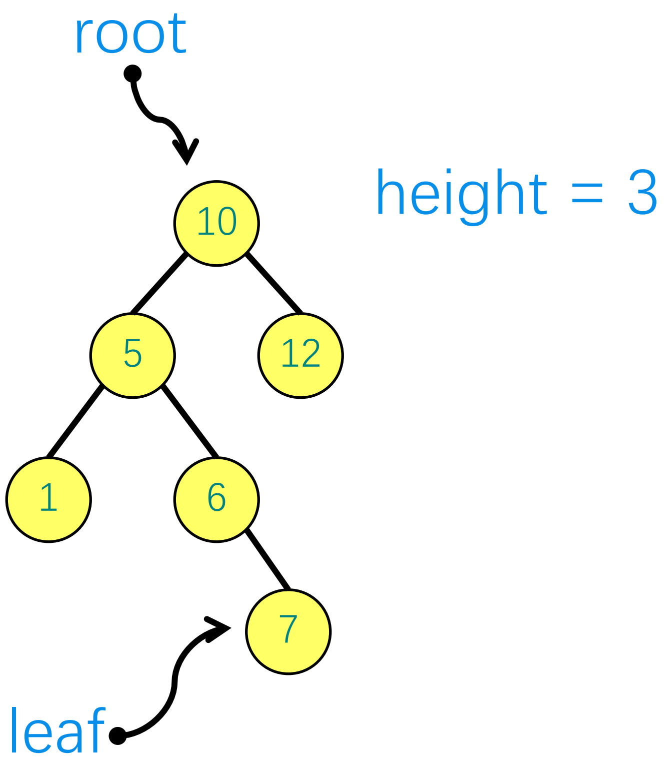
Review: Binary Search Trees

BST

- Each node x has:
  - key[x]
  - Pointers: left[x], right[x], p[x]
- Property: for any node x:
  - For all nodes y in the left subtree of x:

    key[y] ⩽ key[x]
  - For all nodes y in the right subtree of x:

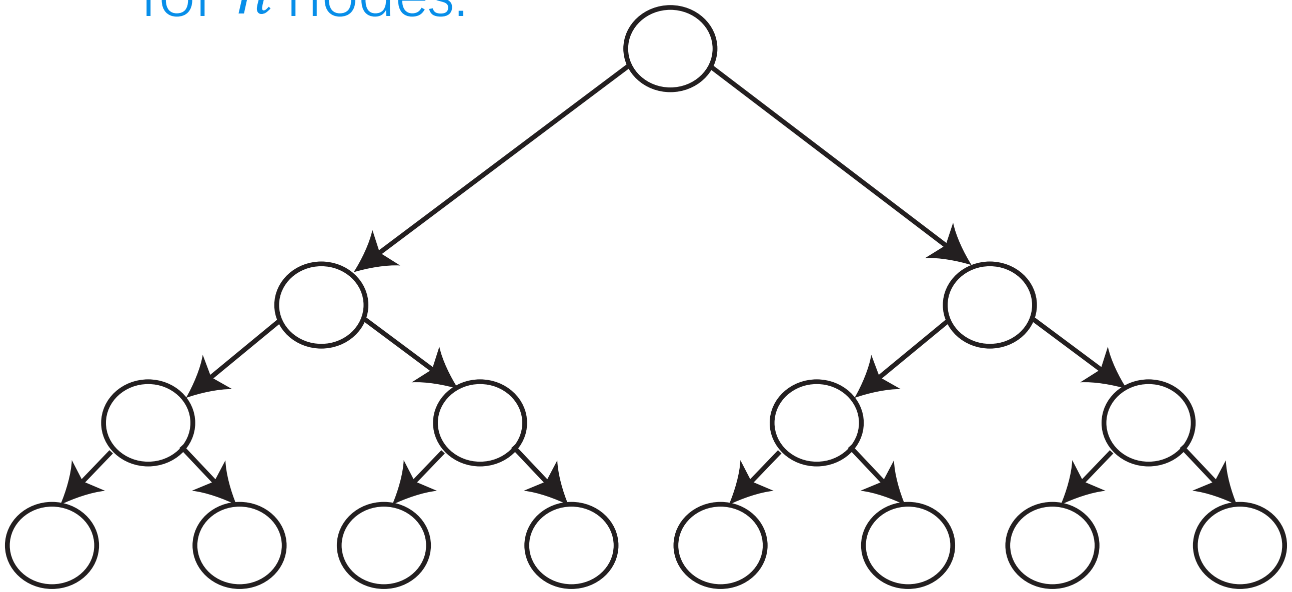
    key[y] > key[x]



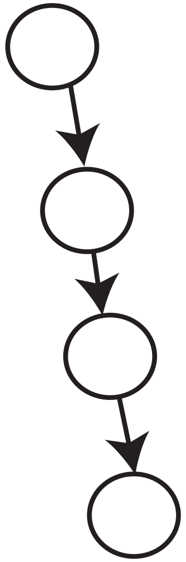
The importance of being balanced

The importance of being balanced

for n nodes:



vs.



Perfectly Balanced

Path

$h \;= \Theta(\log n)$

$h \;= \Theta(n)$

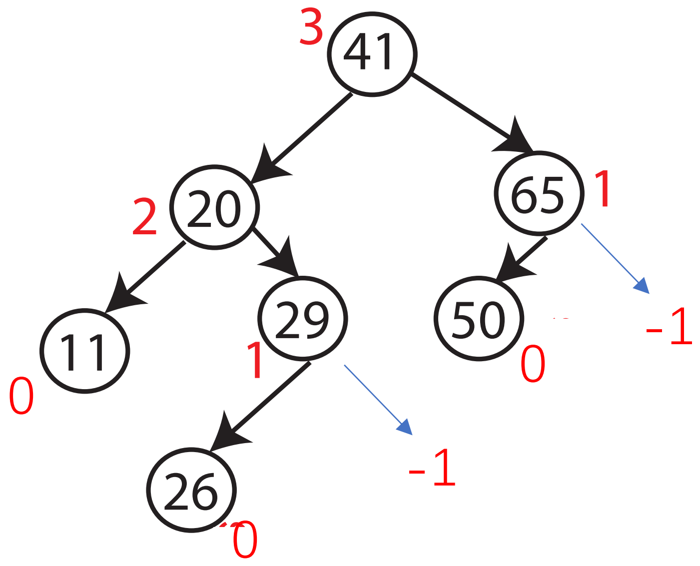
Balanced BSTs

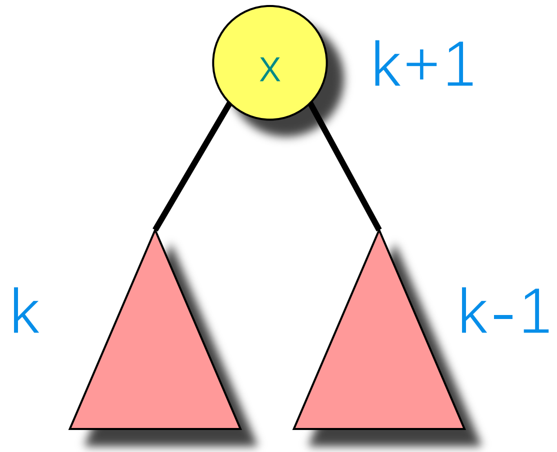
Balanced BST strategy

- Augment every node with some property
- Define a local invariant on property
- Show (prove) that invariant guarantees $\Theta(\log n)$ height
- Design algorithms to maintain property and the invariant

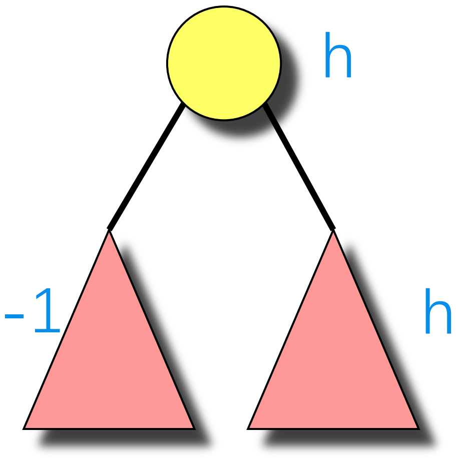
AVL Trees: Definition

- **Property**: for every node, store its height ("augmentation")
  - Leaves have height 0
  - NIL(empty tree) has "height" -1
  - Node have height max(children's height) + 1
- **Invariant**: for every node x, the heights of its left child and right child differ by at most **1**





AVL trees have height Θ(log n)



- Let $\underline{n}_h$ be the minimum number of nodes of an AVL tree of height h, i.e., $n_h \geq \underline{n}_h$
- We have

$n_h \geq \underline{n}_h \geq 1 + \underline{n}_{h-1} + \underline{n}_{h-2}$ % there is a $h-2$ subtree

$> 2\underline{n}_{h-2}$ % since $\underline{n}_{h-1} > \underline{n}_{h-2}$

$> 2 \times 2\ \underline{n}_{h-4}$ % since $\underline{n}_{h-2} > 2\ \underline{n}_{h-4}$

$> 2^{\frac{h}{2}}$

$\Rightarrow 2\log n_h > h$

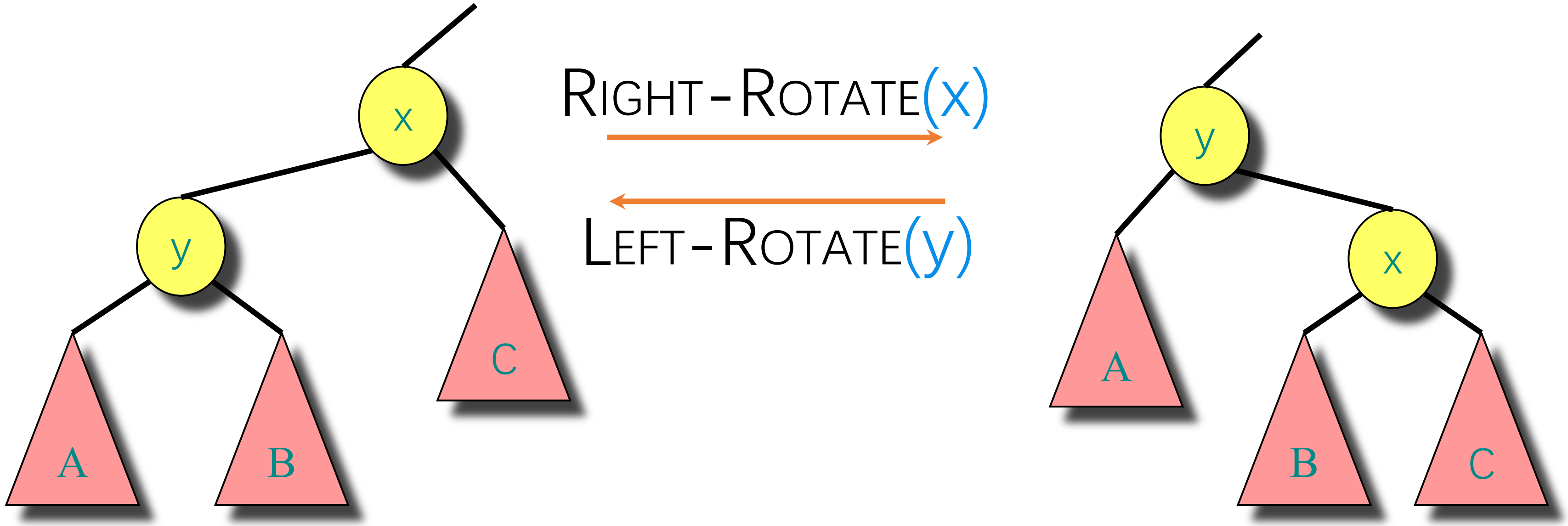
$\Rightarrow h = O(\log n_h)$

Why $h = \Omega(\log n_h)$?

$n_h \leq 2^0 + 2^1 + \cdots + 2^h = 2^{h+1} - 1 < 2^{h+1}$

$\Rightarrow \log n_h < h + 1 \Rightarrow h > \log n_h - 1$

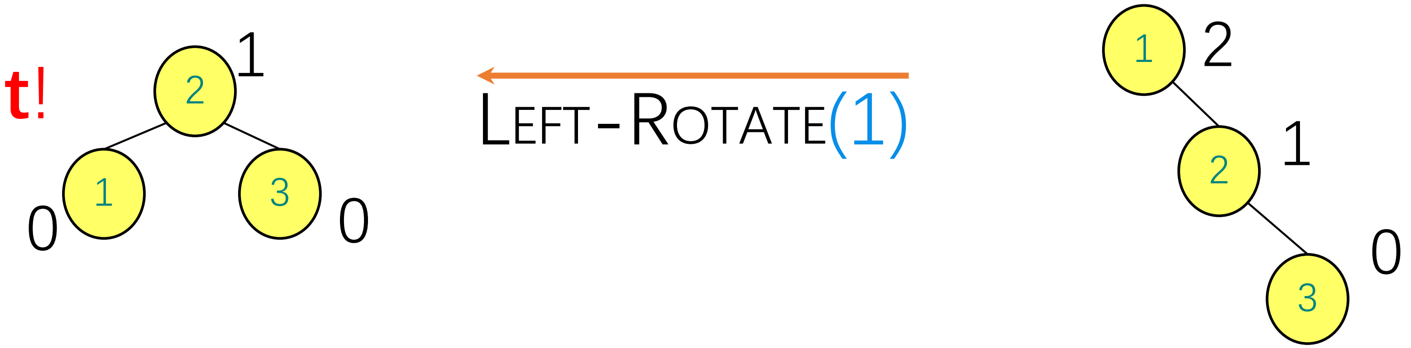
Tree acrobatics! - Rotations



Rotations **maintain the in-order ordering of keys**:

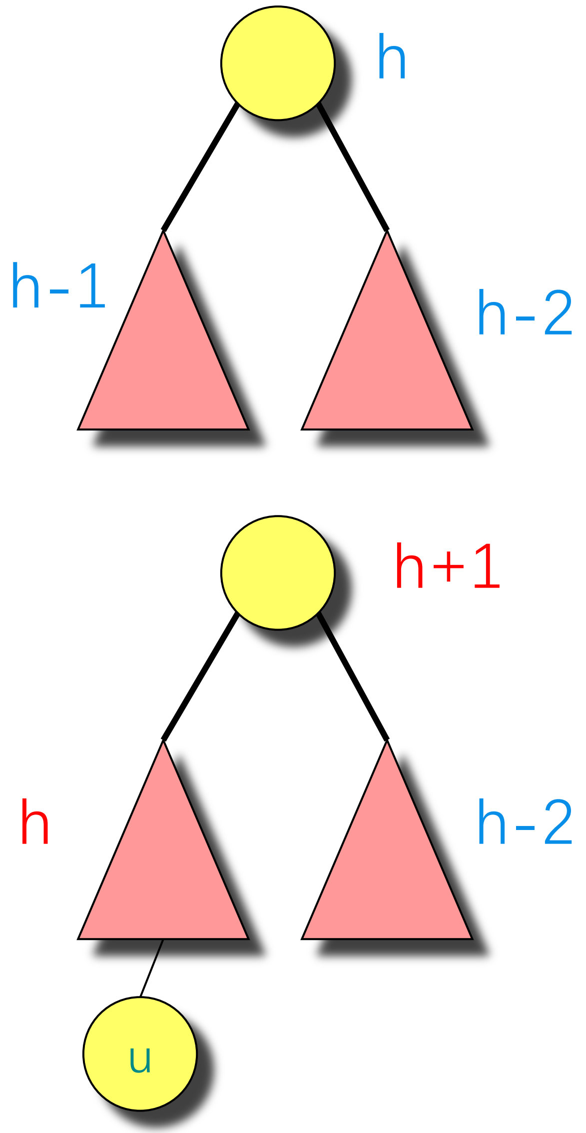
$$\forall a \in A, b \in B, c \in C \rightarrow a \leq y < b \leq x < c$$

Rotations **can reduce the height**!



Insertions/Deletions

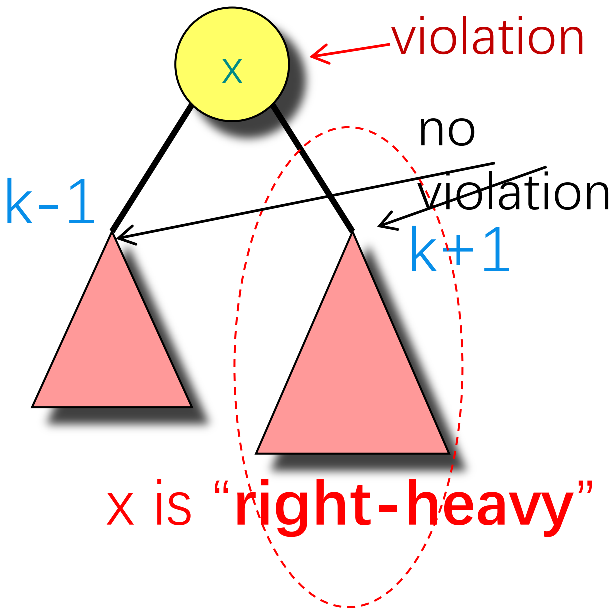
- Insert new node u as in the simple BST
  - Can create imbalance
- Work your way up the tree, restoring the balance
- Similar issue/solution when deleting a node

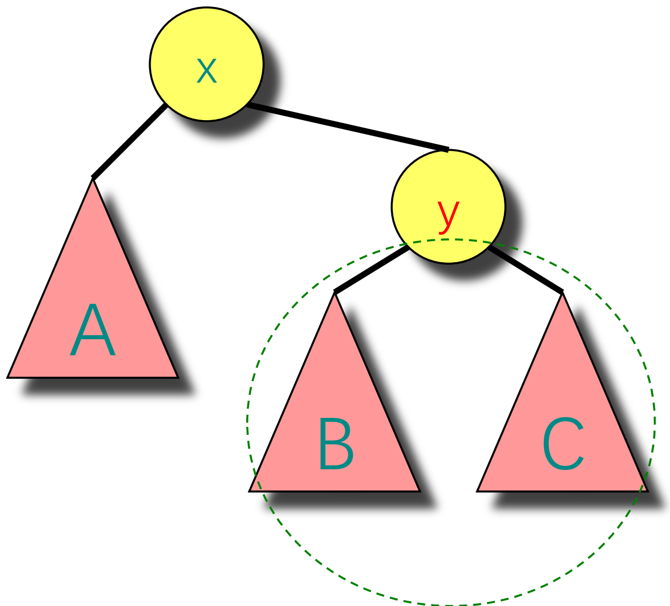


Balancing

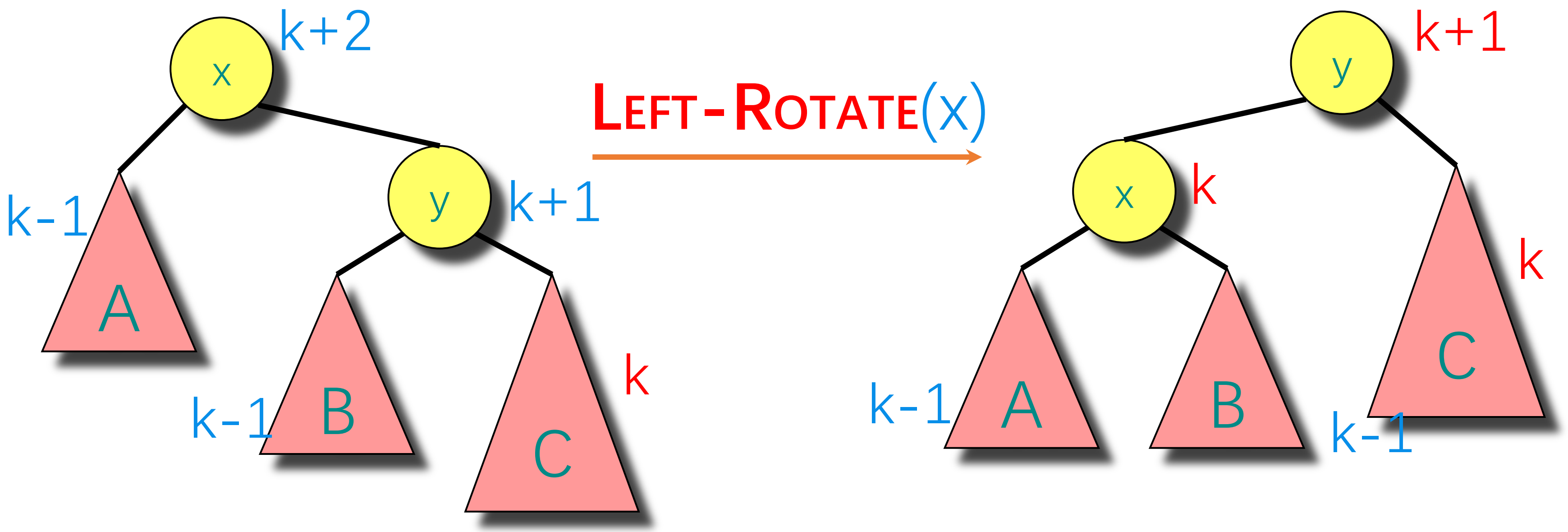
- Let **x** be the lowest "violating" node
  → we will try to correct that and move up the tree
- Assume that x is "right-heavy"
  → we analyze more the right subtree of x
  - **y** is the right child of x
- Scenarios:
  - Case 1: y is right-heavy
  - Case 2: y is balanced
  - Case 3: y is left-heavy

The right child of x has +2 height than the left child of x

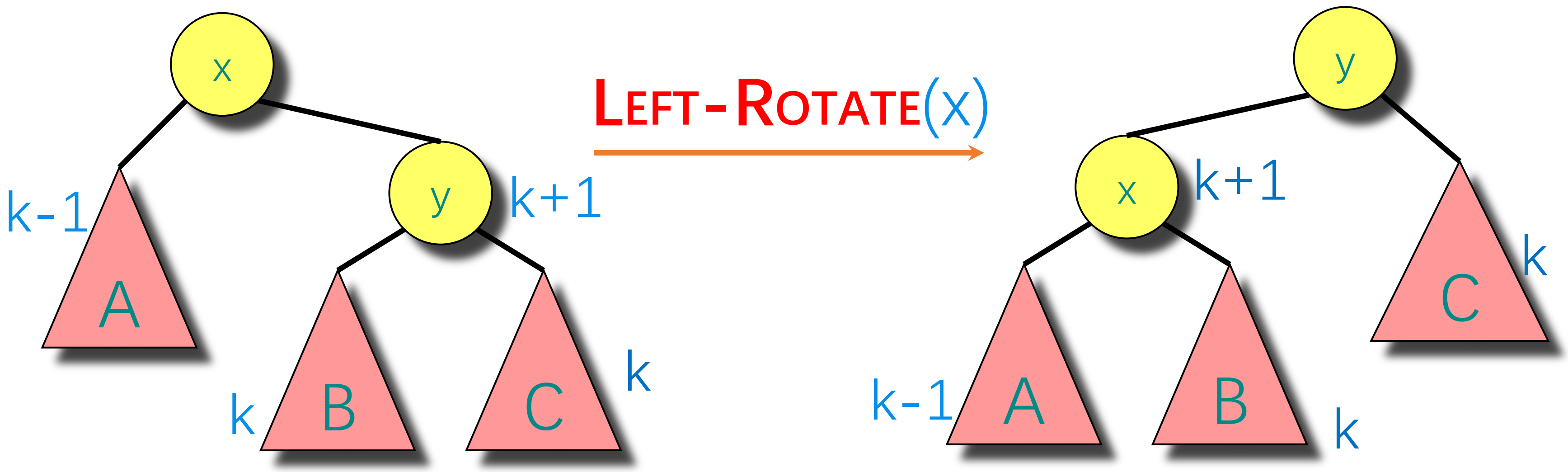




Case 1: y is right-heavy

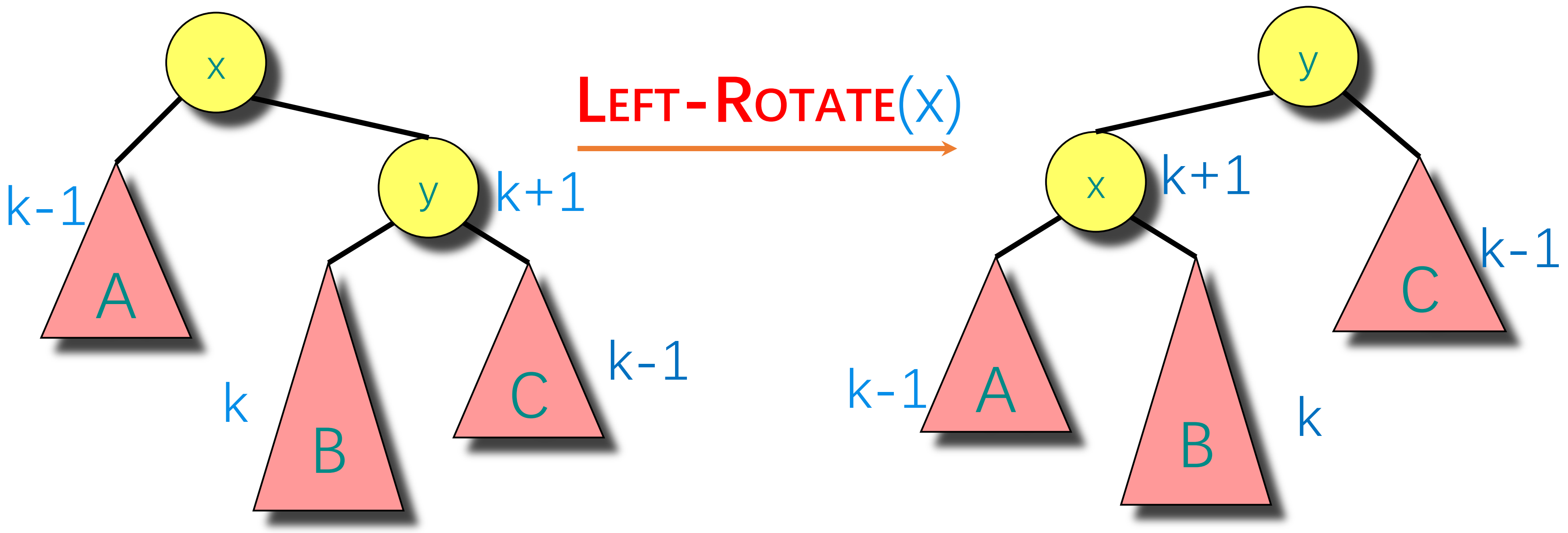


Case 2: y is balanced



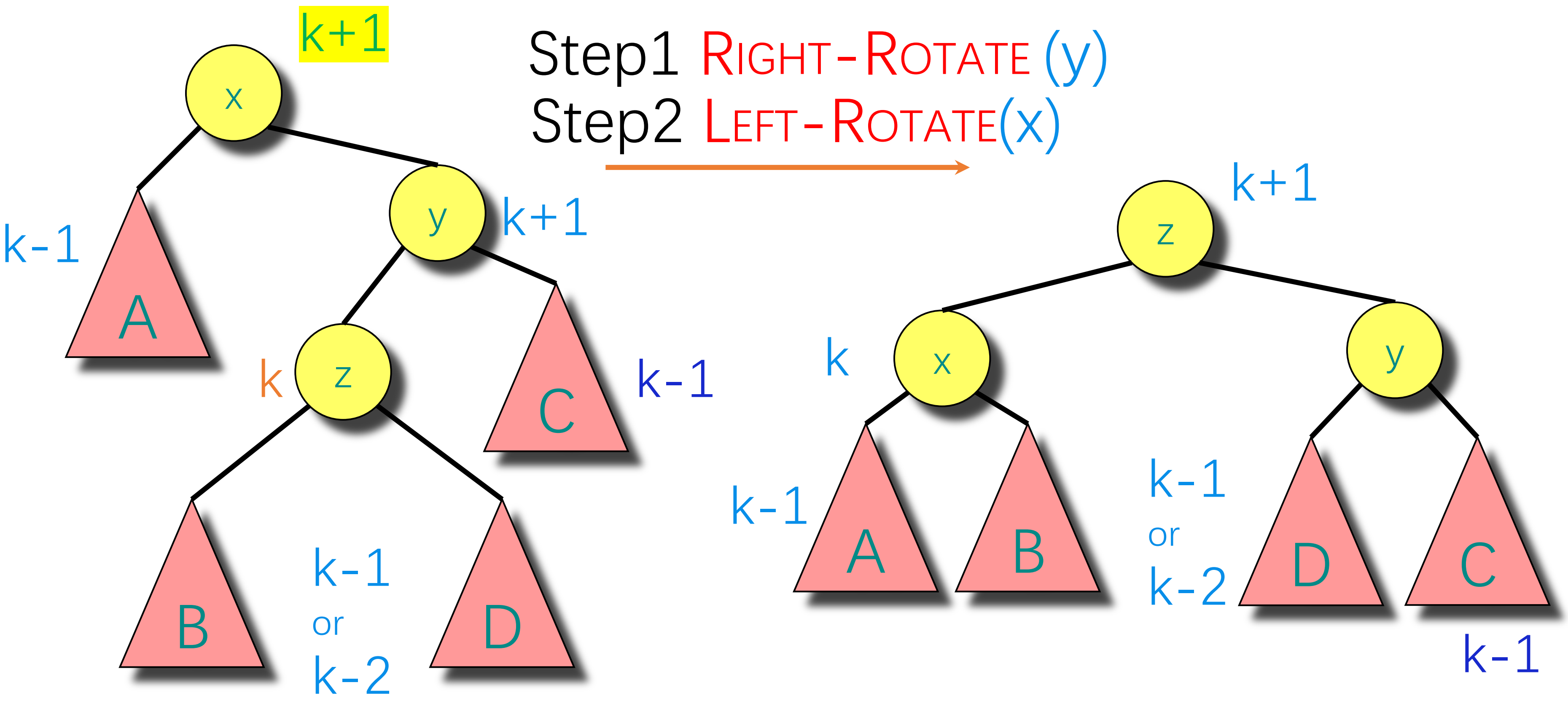
Same as Case 1

Case 3: y is left-heavy

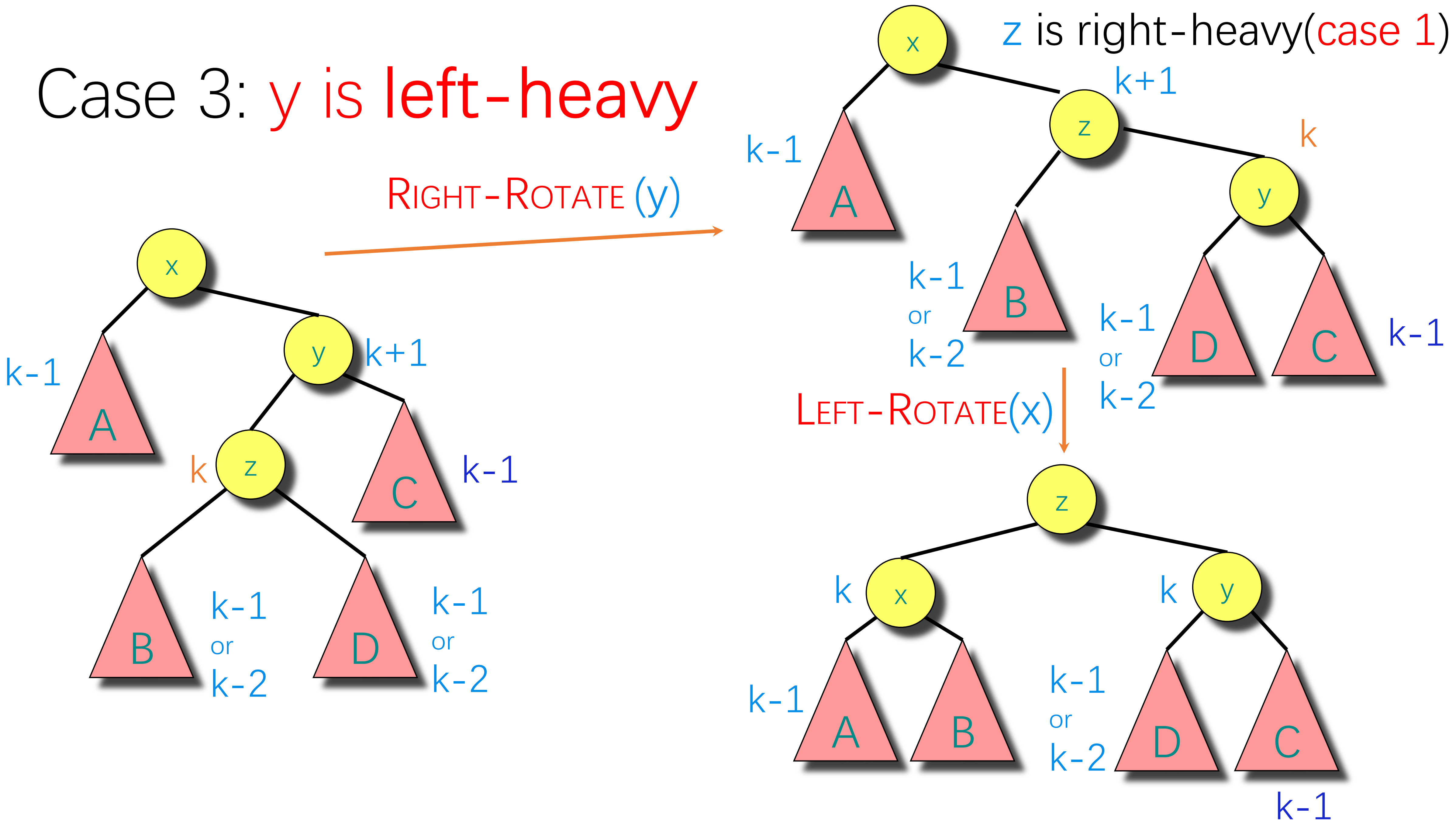


Need to do more …

Case 3: y is left-heavy



Case 3: y is left-heavy



Conclusions

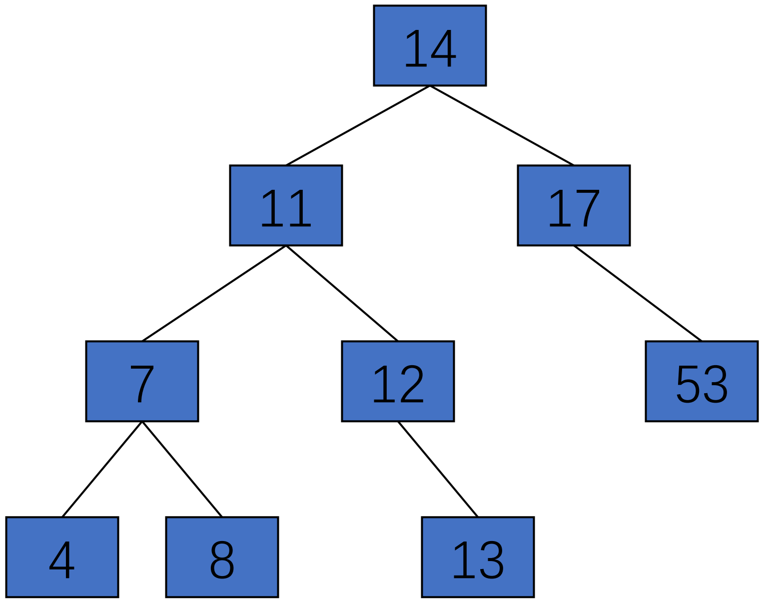
- In balanced BSTs all operations take $O(\log n)$ time.
- Can maintain balanced BSTs using $O(\log n)$ time per deletion.
- Insertion needs to restore imbalance at most once. But for deletion, imbalance may propagate upward so that many rotations may be needed.

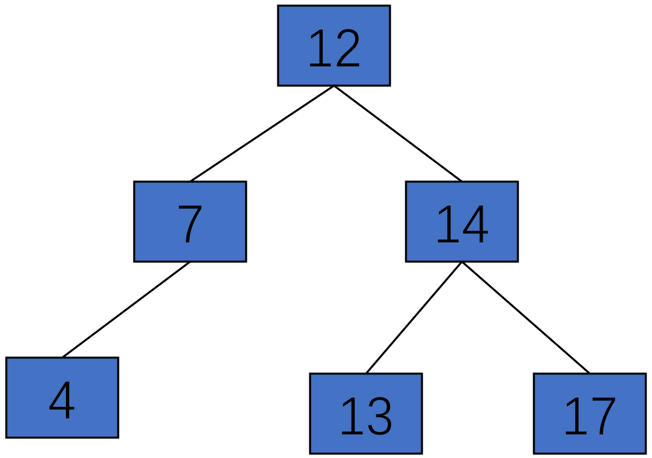
Example 1

- Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree.
- Then delete 53, 11, 8

Example 1

- Insert 14, 17, 11, 7, 53, 4, 13, 12, 8 into an empty AVL tree.
- Then delete 53, 11, 8

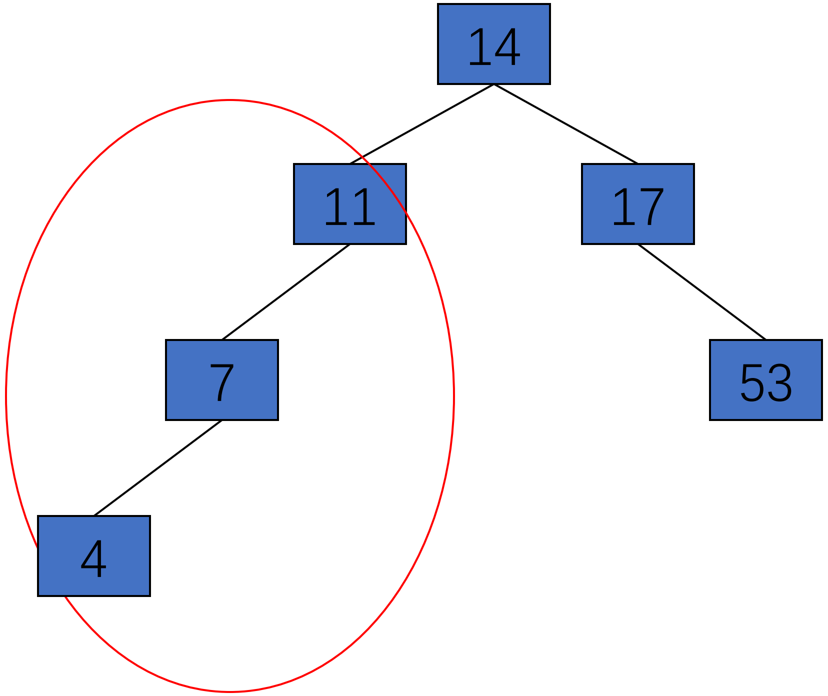




AVL Tree Exercise:

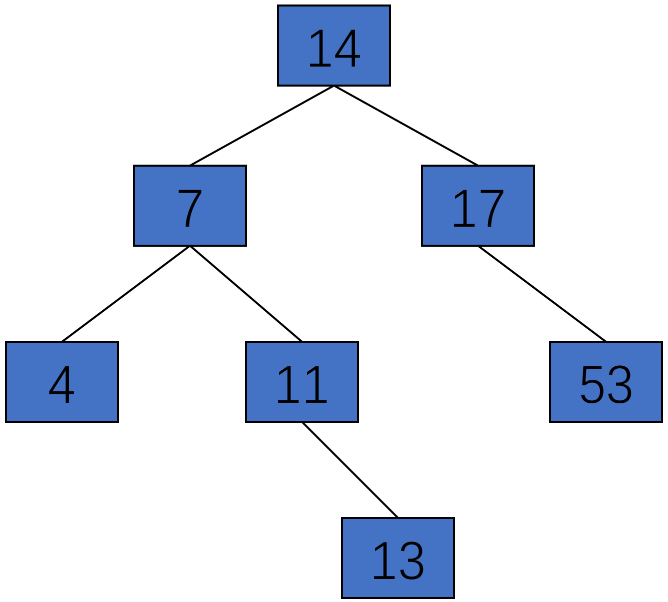
- Insert 14, 17, 11, 7, 53, 4, 13，12，8 into an empty AVL tree

- Now insert 4



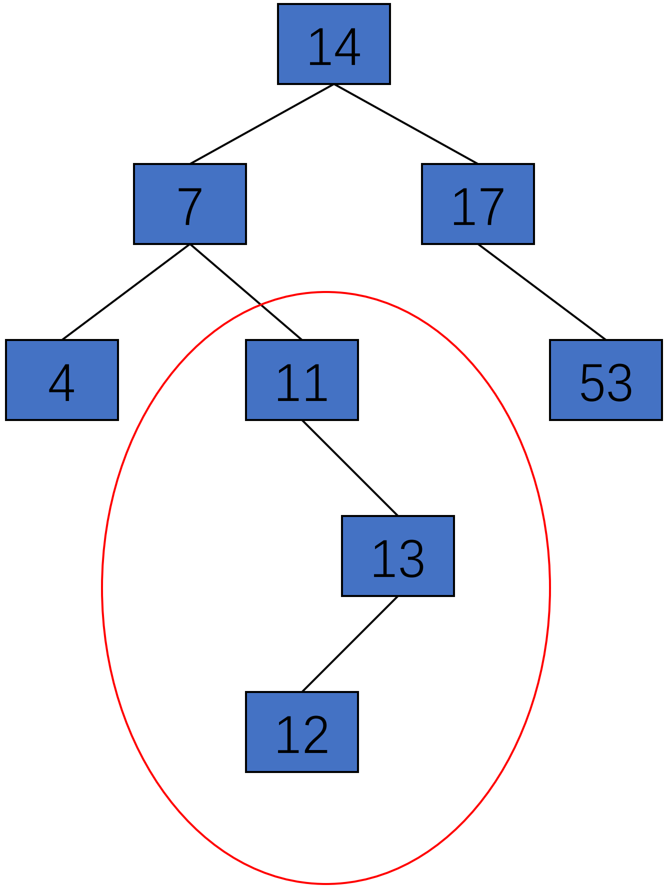
AVL Tree Exercise:

- Now the AVL tree is balanced.



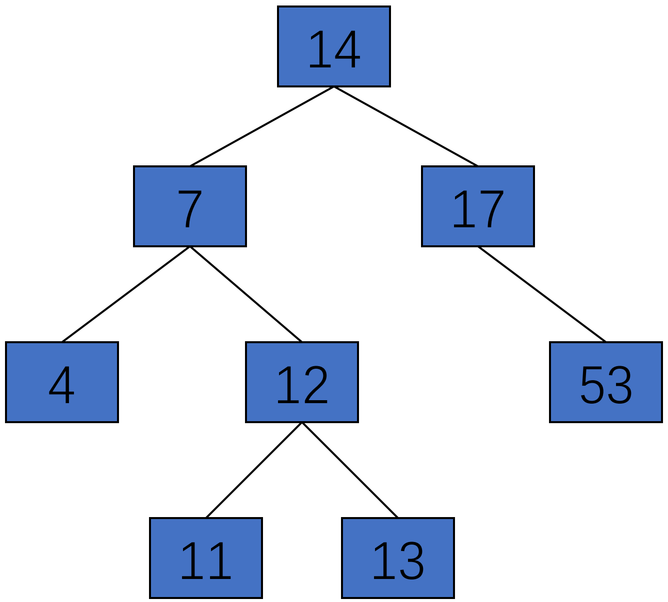
AVL Tree Exercise:

- **Now insert 12**



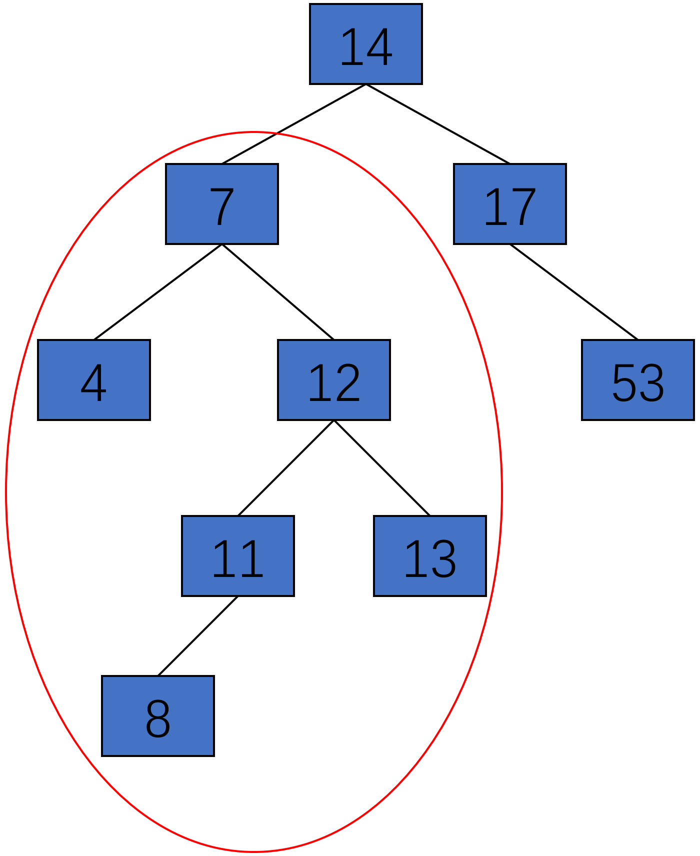
AVL Tree Exercise:

- Now the AVL tree is balanced.



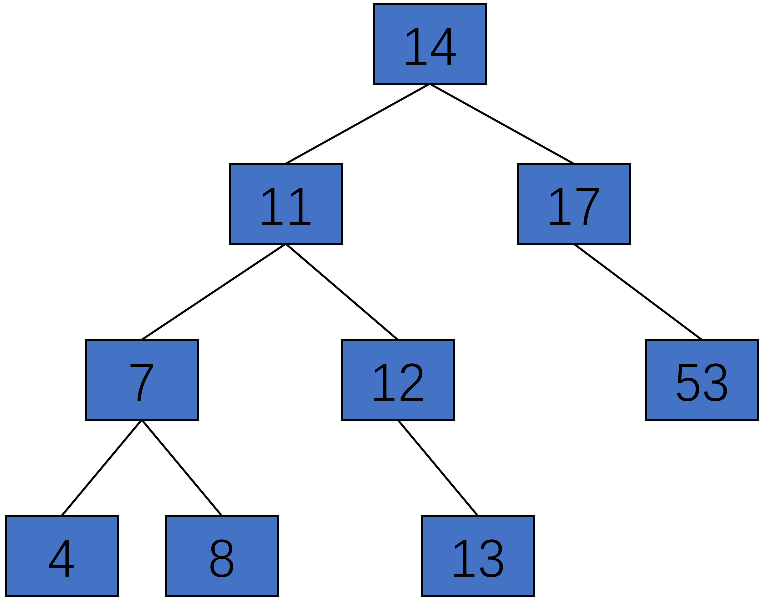
AVL Tree Exercise:

- Now insert 8



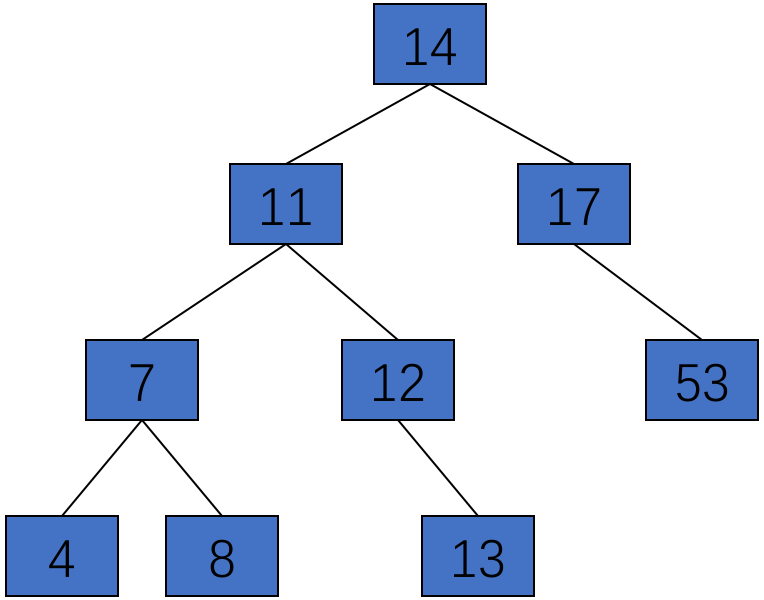
AVL Tree Exercise:

- Now the AVL tree is balanced.



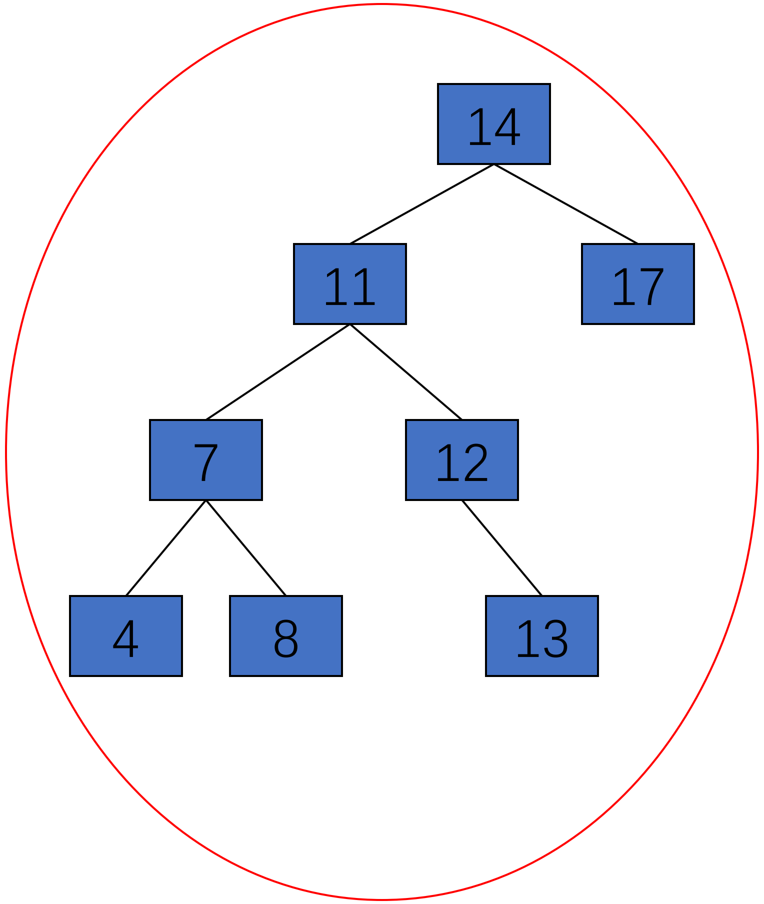
AVL Tree Exercise:

- Now delete 53



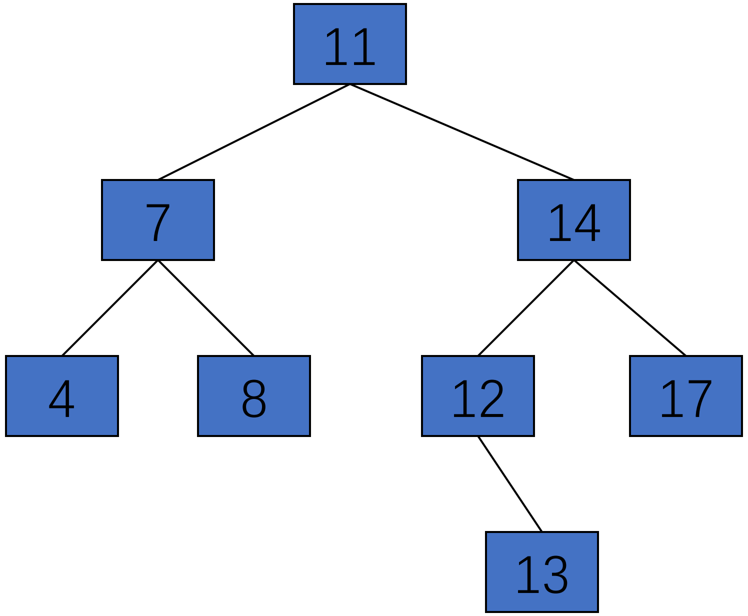
AVL Tree Exercise:

- Now delete 53, unbalanced



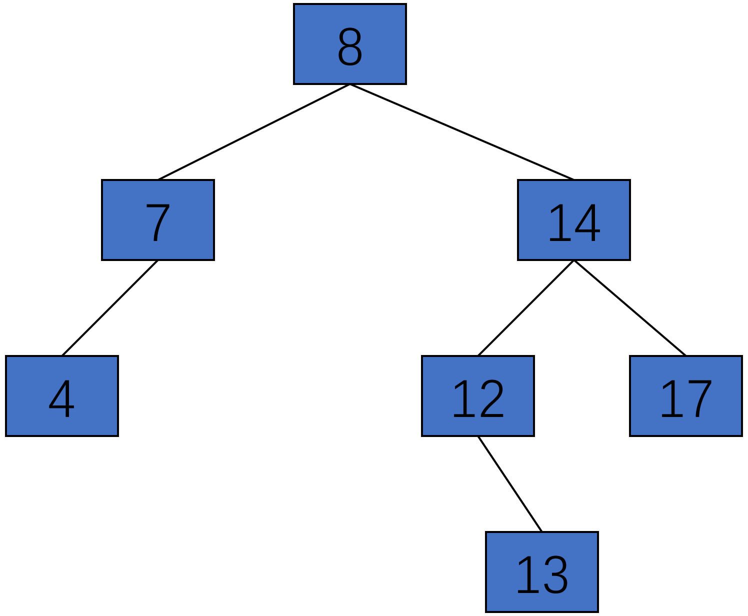
AVL Tree Exercise:

- Now the AVL tree is balanced.



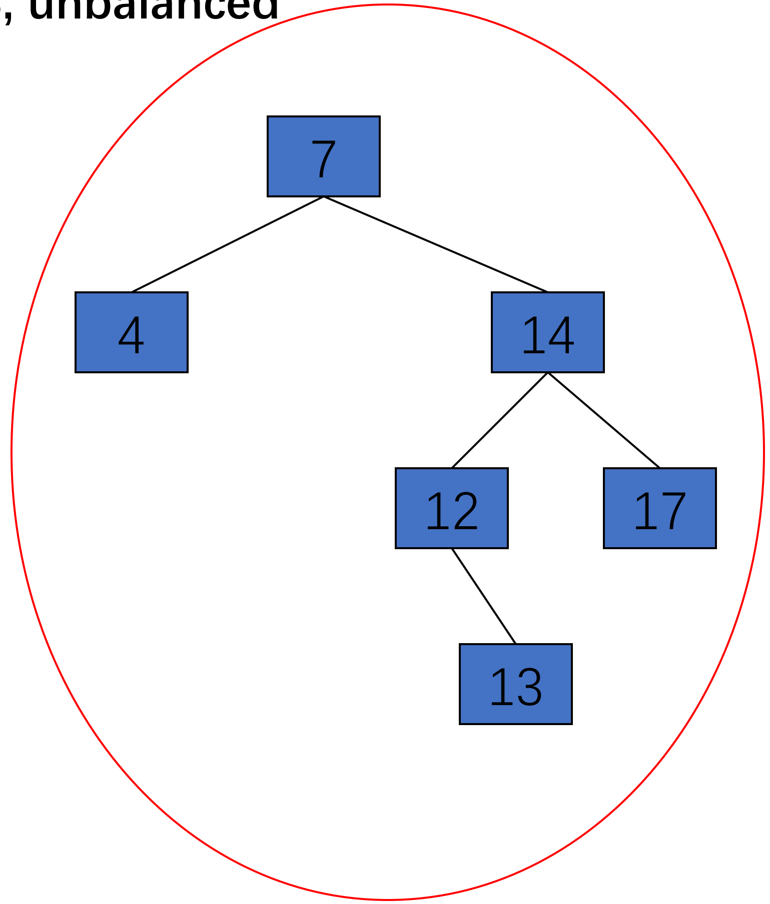
AVL Tree Exercise:

- Now delete 11, replace it with the largest in its left branch



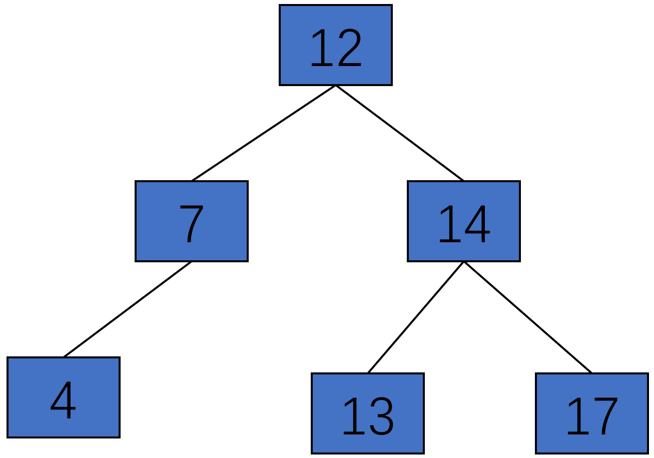
AVL Tree Exercise:

- Now delete 8, unbalanced



AVL Tree Exercise:

- **Balanced!!**



Example 2

- AVL tree insertion and deletion exercise
- LeetCode P1382 Balance a BST

  https://leetcode.cn/problems/balance-a-binary-search-tree/