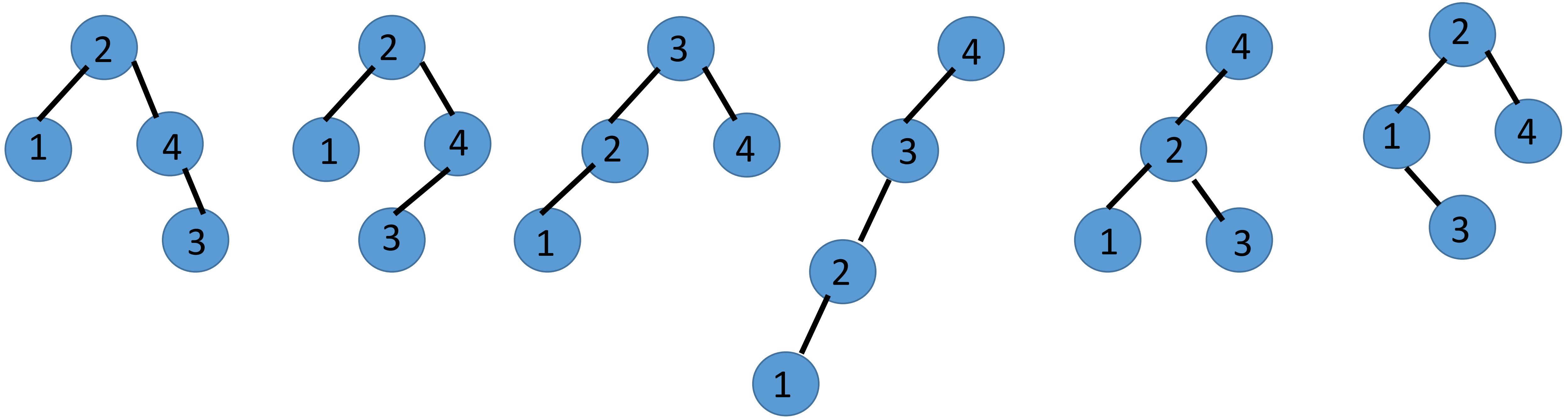
# AVL trees
# ≡
# Height balanced (augmented) BSTs

# Definitions, terminologies, operations

- AVL tree: Adelson-Velsky-Landis, a BST which is height balanced at every node.
- AVL tree:
  - The difference in the height of the LST and RST (at the root) is at most 1; i.e., $|h(LST) - h(RST)|=1$,
  - The LST and RST (of the root) are AVL trees.
- **Which of the following are AVL trees?**



# Definitions, terminologies, operations

- An AVL tree on $2^k-1$ nodes has a unique structure – perfect binary tree with the median of the corresponding set being the root for each subtree.
- An AVL tree of height h has at most $2^{h+1}-1$ nodes ↔ An AVL tree on n nodes has height at least $\log_2 n - 1$.
-

| Height of the AVL tree | Minimum number of nodes |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 7 |
| 4 | 12 |
| 5 | 20 |

If n(h) is the minimum number of nodes in an AVL tree of height h, then
$n(h) = n(h-1) + n(h-2) + 1$, for $h \geq 2$

and $n(0)=1$, $n(1)=2$.
Therefore,

$$n(h) = \left(\frac{1}{2}+\frac{3\sqrt{5}}{10}\right)\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^h + \left(\frac{1}{2}-\frac{3\sqrt{5}}{10}\right)\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^h + \frac{2\sqrt{5}}{5(1+\sqrt{5})}\left(-\frac{2}{1+\sqrt{5}}\right)^h - \frac{2\sqrt{5}}{5(1-\sqrt{5})}\left(-\frac{2}{1-\sqrt{5}}\right)^h - 1$$

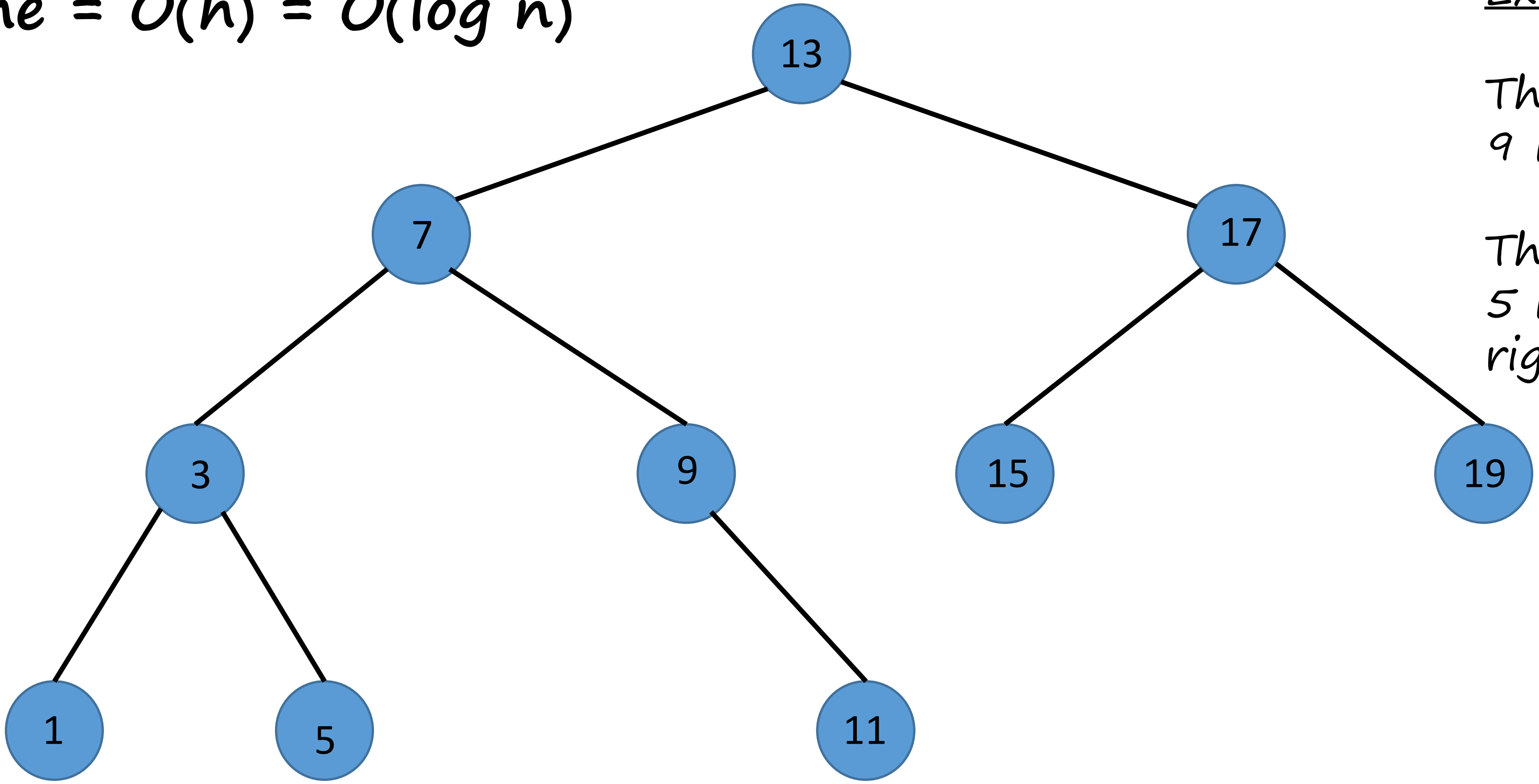
$$\approx 1.8944 \times 1.6180^h$$

↔ An AVL tree on n nodes has height at most $\log_{1.6180}\left(\frac{n}{1.8944}\right) \approx \log_{1.6180} n - 1.3277$

- Height of an AVL tree on n nodes is O(log n)

# Searching in an AVL tree

- Start from the root.
- Traverse as in a BST (AVL tree is a special kind of BST) to obtain the search path.
- Run-time = O(h) = O(log n)



<u>Example:</u>

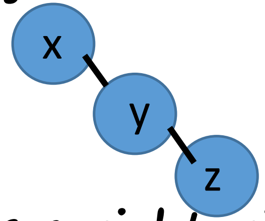
The search path for 9 is root-left-right.

The search path for 5 is root-left-left-right.

# Understanding conversion from BST to AVL tree

Creating AVL trees for permutations of {x, y, z}. Assume x < y < z.

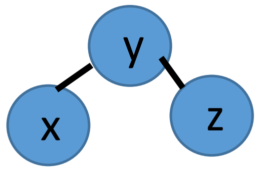
- For the permutation <x, y, z>
  - Step 1: Create a BST for <x, y, z>



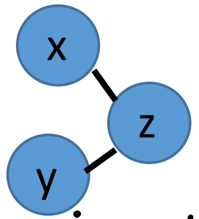
    This BST is not height balanced; i.e., is not an AVL tree. Has a right-right imbalance at x.

  - Step 2: Perform rotation(s) to balance the height.

    In this case, perform LEFT_ROTATE(x)



- For the permutation <x, z, y>
  - Step 1: Create a BST for <x, z, y>



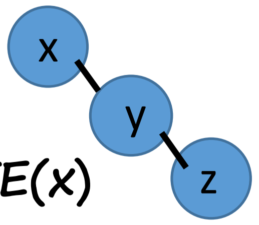
    This BST is not height balanced; i.e., is not an AVL tree. Has a right-left imbalance at x.

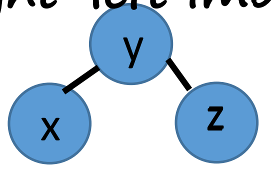
  - Step 2: Perform rotation(s) to balance the height.

    In this case, first perform RIGHT_ROTATE(z)

    This can now be height balanced by LEFT_ROTATE(x)

    This is called double rotation.



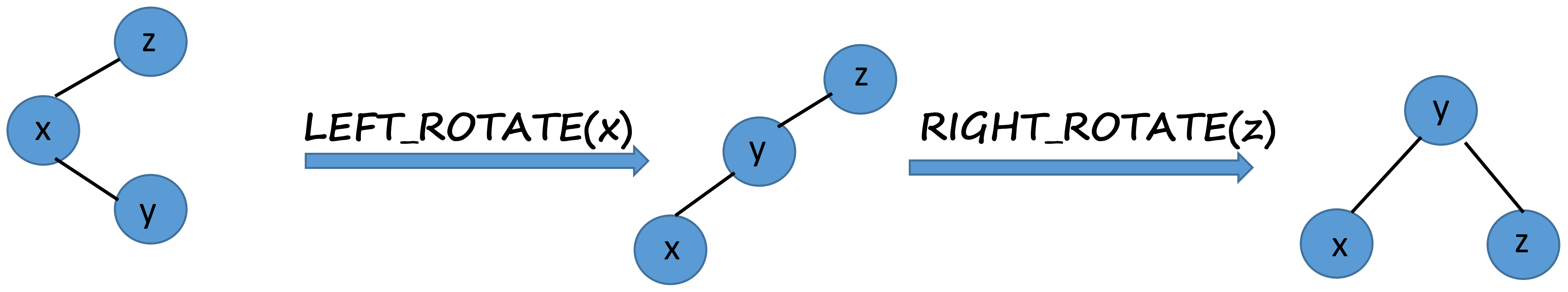


- For the permutations <y, x, z> and <y, z, x>
  - Step 1: Create a BST

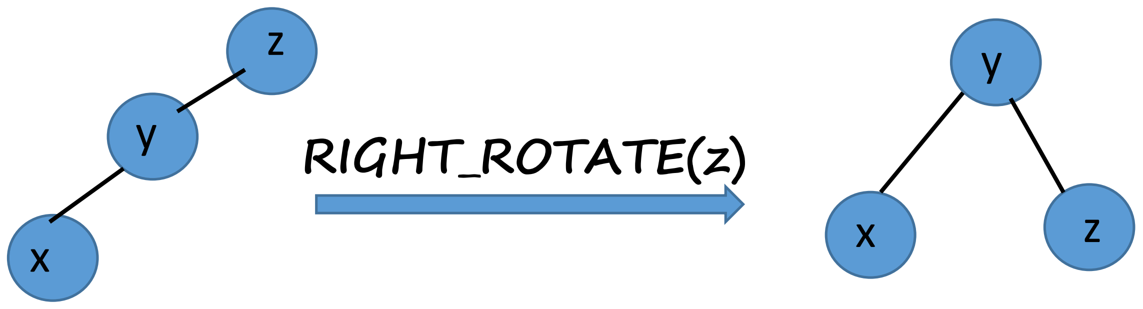
    The BST is height balanced and hence an AVL tree; i.e., no rotation is required.

# Understanding conversion from BST to AVL tree

- For the permutation <z, x, y>
  - The BST has left-right imbalance at z.
  - Perform double rotation – a left rotate and a right rotate.



- For the permutation <z, y, x>
  - The BST has left-left imbalance at z.
  - Perform a single rotation – a right rotate.

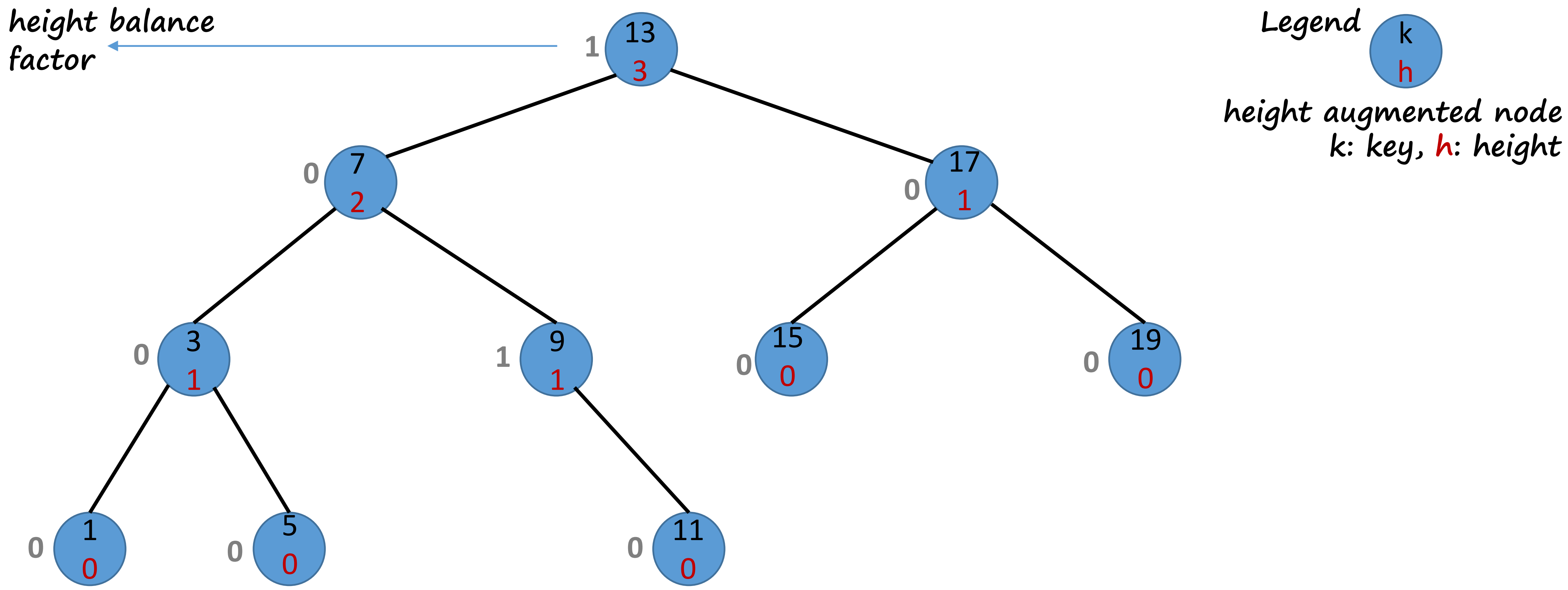


Imbalances can be removed by single or double rotations. The kind of imbalance dictates the kind of rotation to be used.

# Insertions into AVL trees

Insert {8, 10, 16, 4} into the following AVL tree

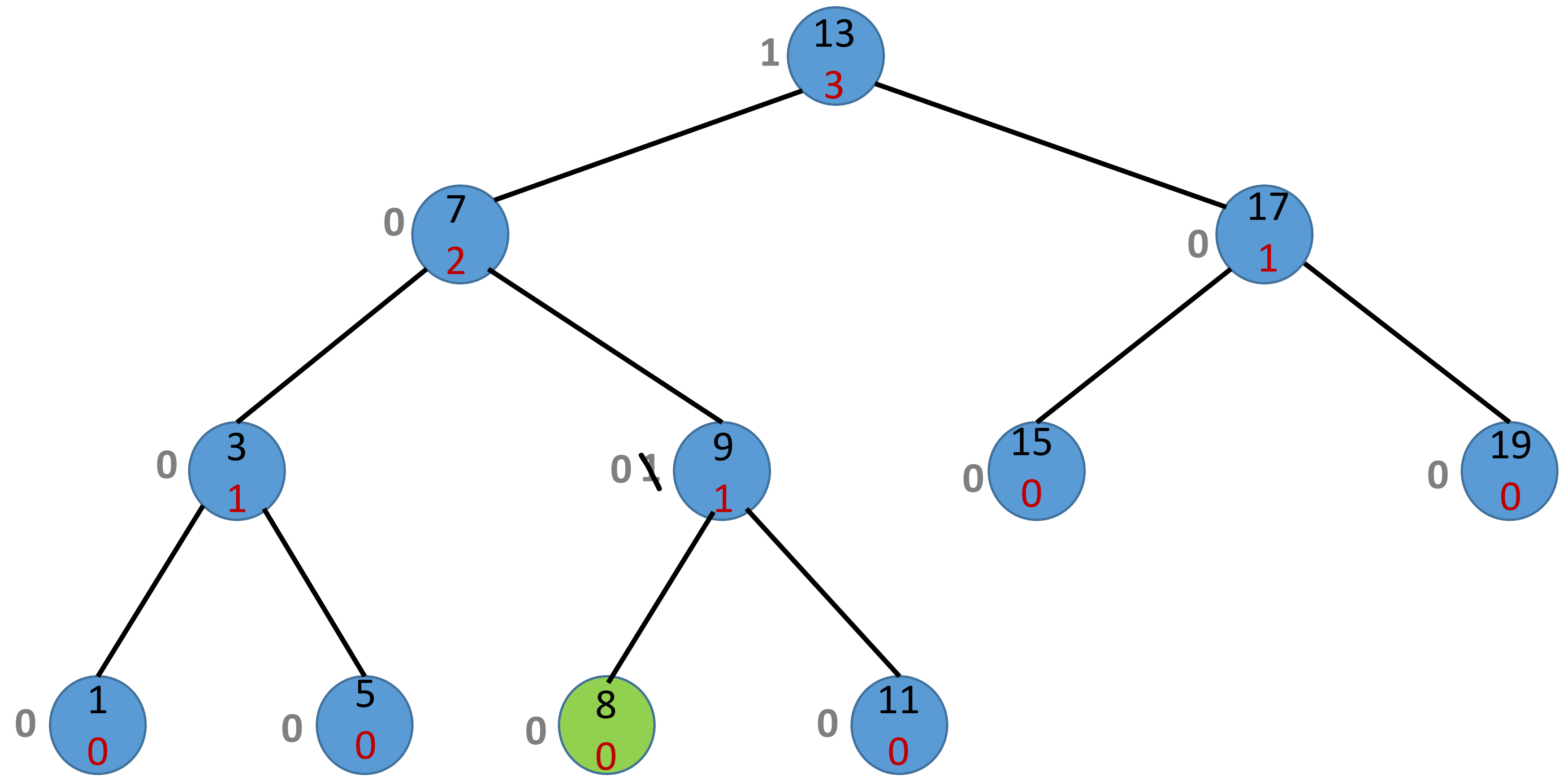
(such that the resultant after each insertion is an AVL tree)



# Insertions into AVL trees

To Insert 8 into the AVL tree

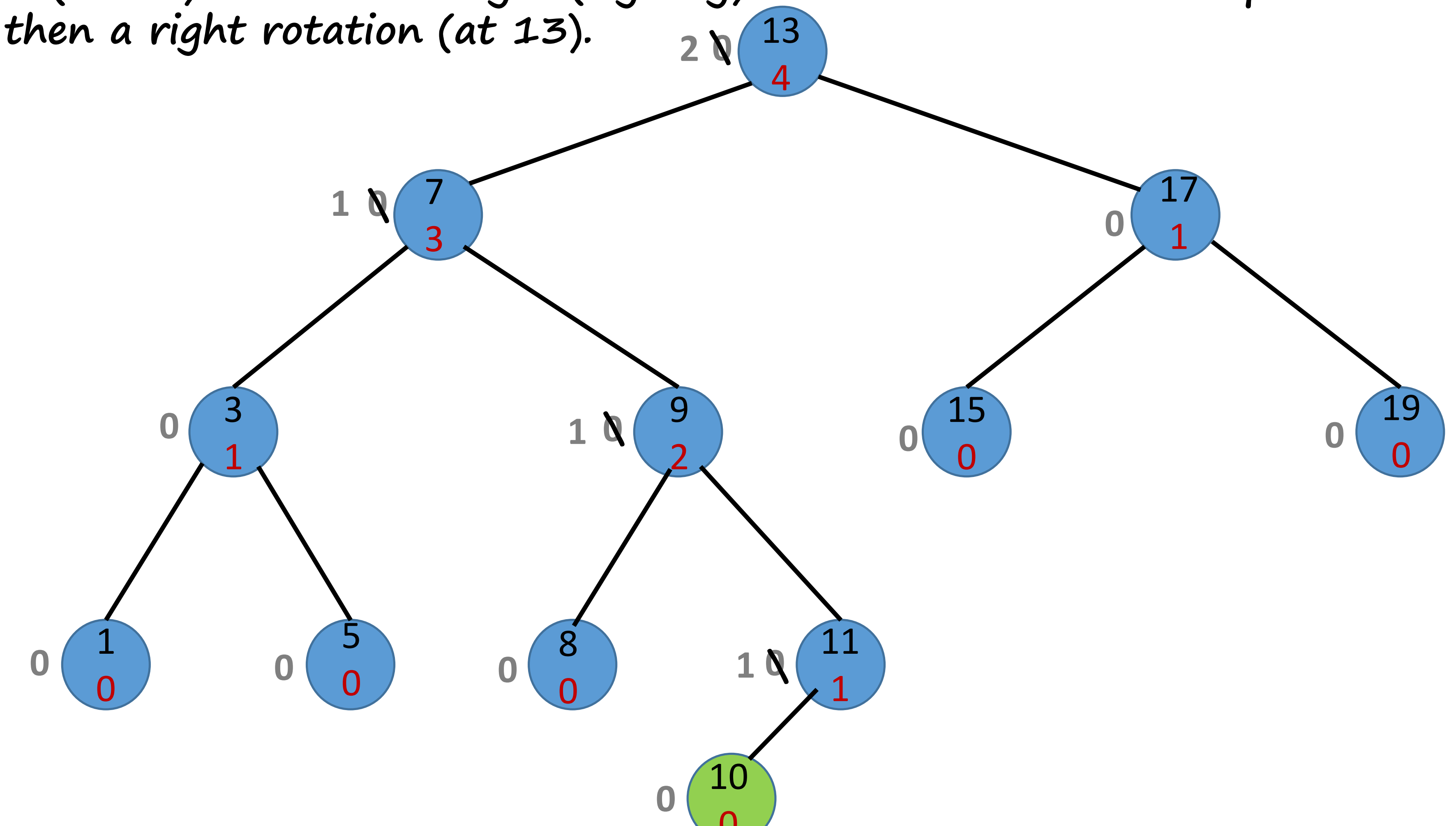
- Root-Left-Right-Left is the search/insertion path (for inserting 8 as the left-child of 9).
- This insertion does not create height imbalance at any node (though there is a change in the (height and hence the) height balance factor at some nodes on the search path) and hence remains an AVL tree after insertion.



# Insertions into AVL trees

To Insert 10 into the AVL tree

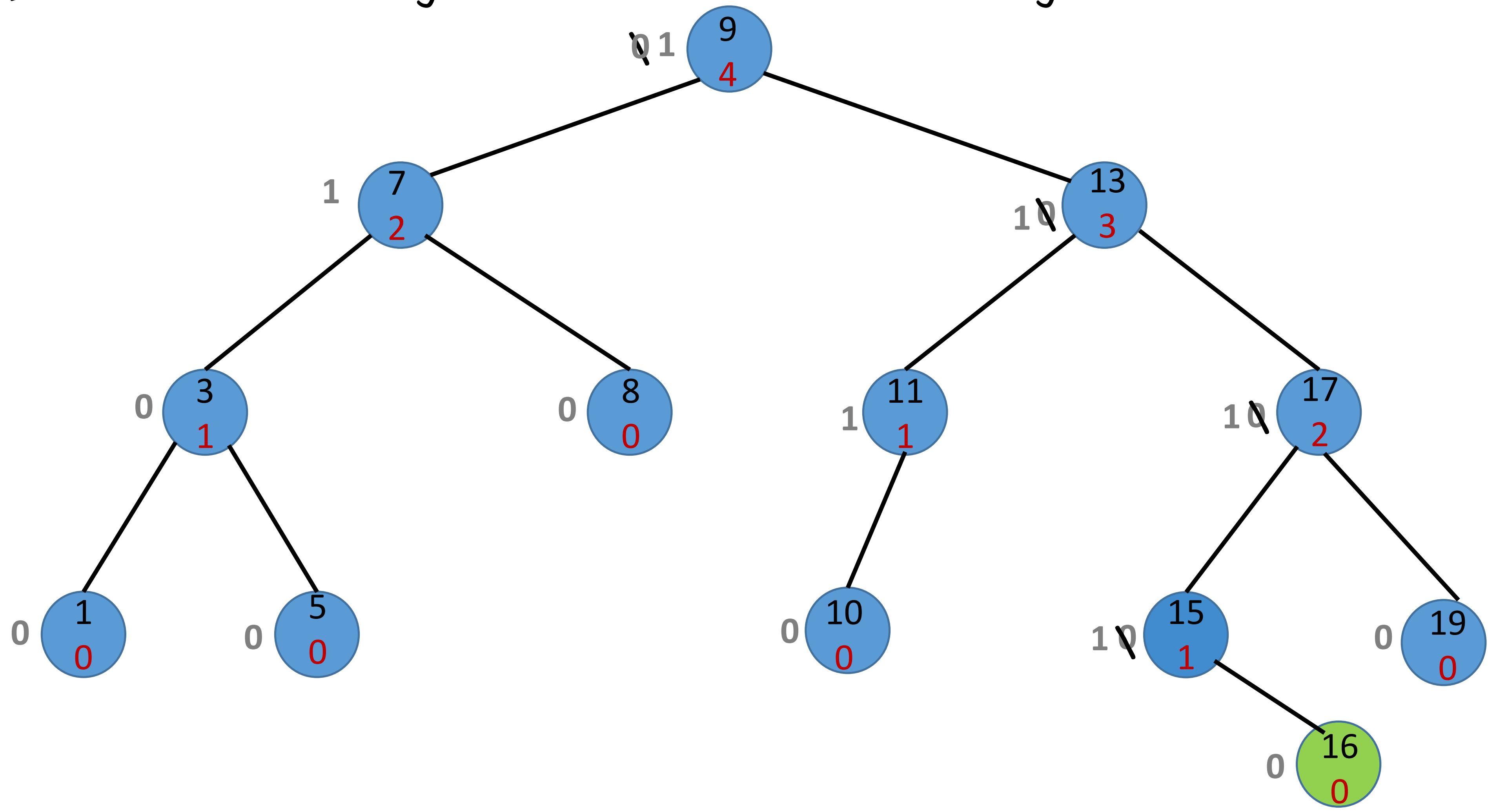
- Root-Left-Right-Right-Left is the search/insertion path (for inserting 10 as the left-child of 11).
- This insertion causes change in the height of the nodes on the search path and also creates a height imbalance at the root (at 13). It is a left-right (zig-zag) imbalance and hence requires a left rotation (at 7) and then a right rotation (at 13).



# Insertions into AVL trees

To Insert 16 into the AVL tree

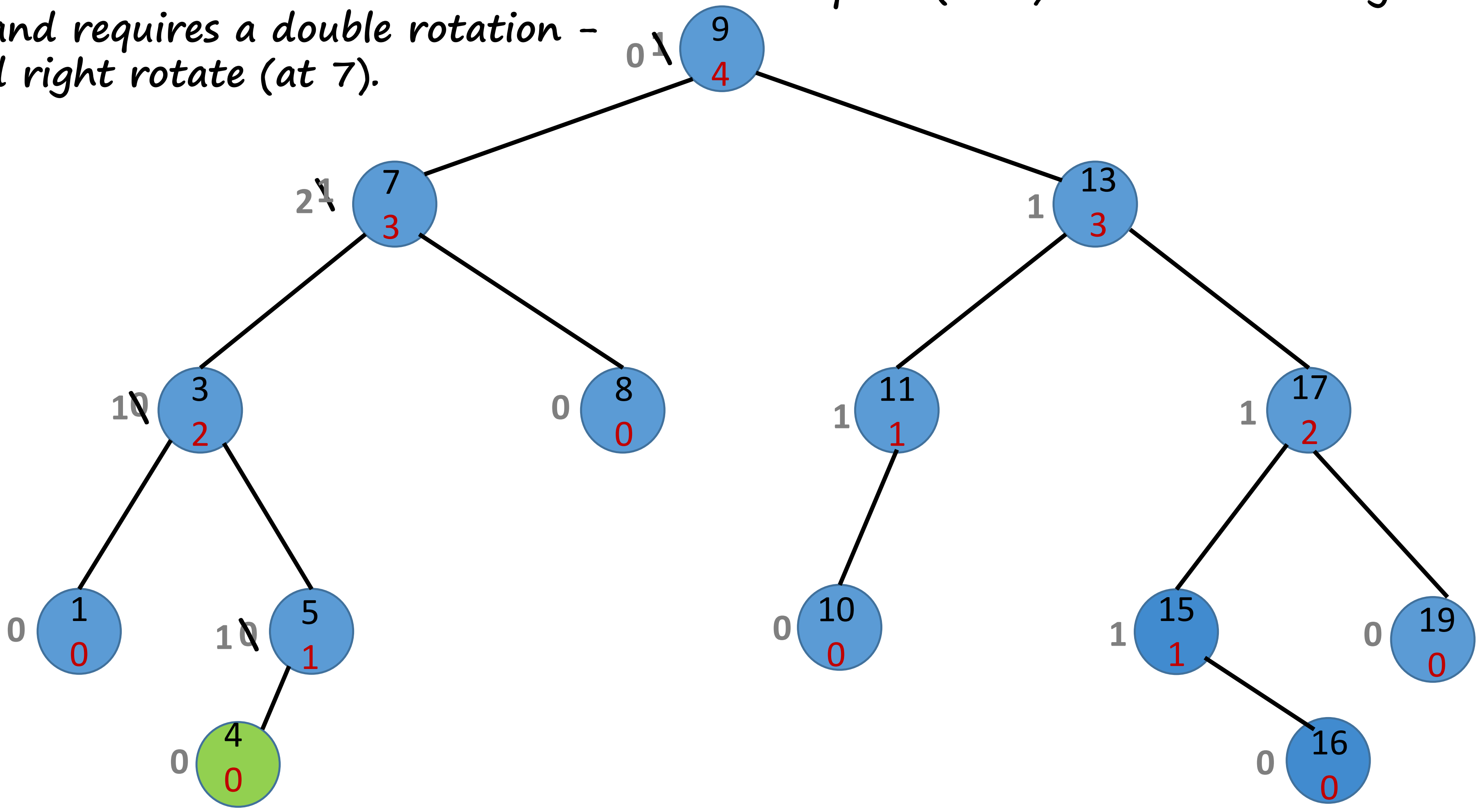
- Root-Right-Right-Left-Right is the search/insertion path (for inserting 16 as the right-child of 15).
- Though this causes some changes on the (height and hence the) height balance factor at some nodes on the insertion path, it does not cause any imbalance. The BST created by this insertion remains an AVL tree.



# Insertions into AVL trees

To Insert 4 into the AVL tree

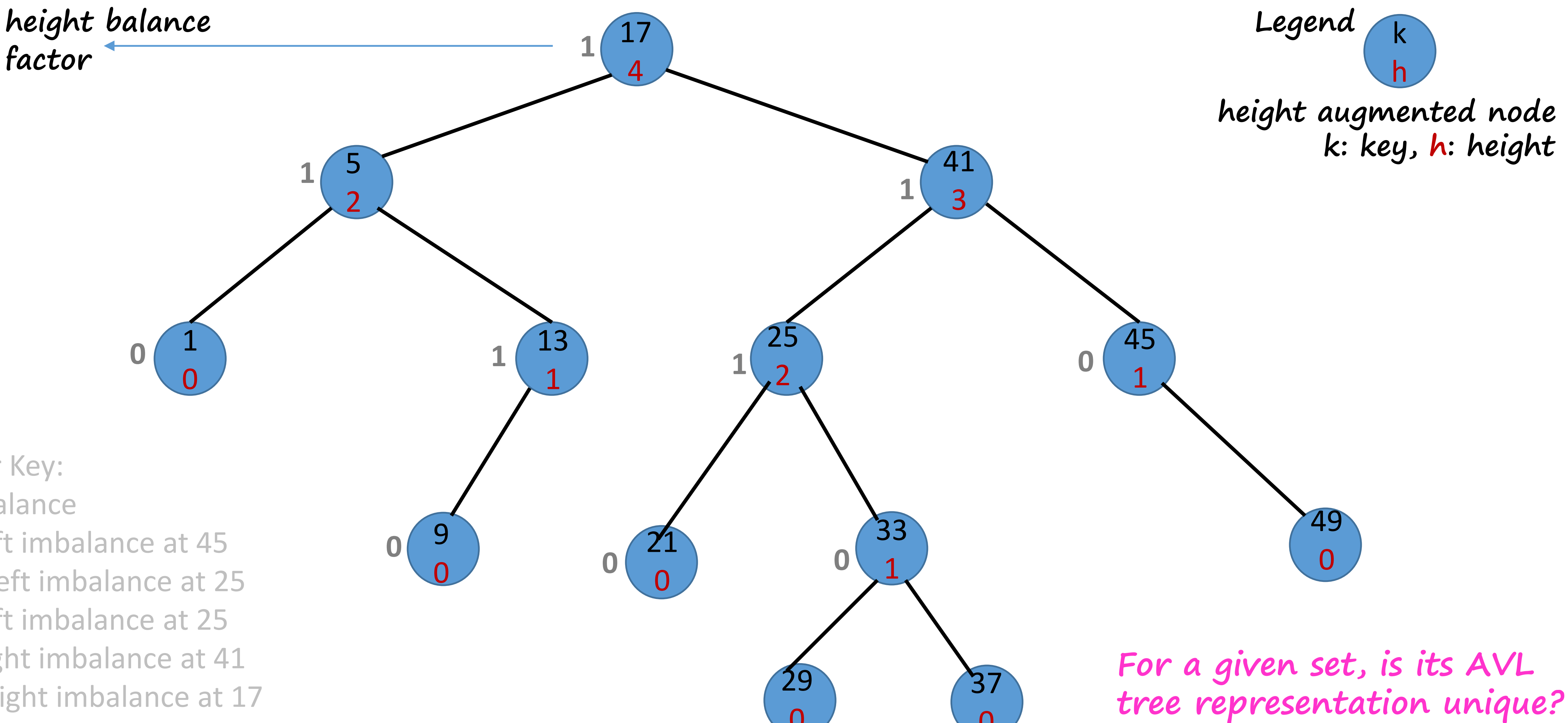
- Root-Left-Left-Right-Left is the search/insertion path (for inserting 4 as the left-child of 5).
- This causes some changes on the (height and hence the) height balance factor at some nodes on the insertion path and an imbalance at some nodes on the insertion path (at 7). This is a left-right imbalance (zig-zag) and requires a double rotation – left rotate (at 3) and right rotate (at 7).



# Insertions into AVL trees (Exercise)

Insert {15, 53, 31, 19, 39, 100} into the following AVL tree

(such that the resultant after each insertion is an AVL tree)



Answer Key:
No rebalance
Left-Left imbalance at 45
Right-Left imbalance at 25
Left-Left imbalance at 25
Left-Right imbalance at 41
Right-Right imbalance at 17

For a given set, is its AVL tree representation unique?

# Insertions into AVL trees

- Step-1: Insert as in a BST. $O(h)$
- Step-2: Check if there is any height imbalance. $O(n^2)$ w/o augmentation, $O(1)$ with augmentation
  - Insertion increases the height of the tree by at most one.
  - If there is an increase in the height (of the tree) then there is a height imbalance at some nodes on the insertion path; conversely, if there is a height imbalance at a node (on the insertion path) then there is an increase in the height of some subtrees on the insertion path.

Note: Height imbalance, if any, will be on some nodes on the insertion path.

- Step-2.1: Rebalance the closest ancestor (of the inserted node) at which the imbalance is present. $O(1)$
  - The kind of rebalancing; i.e., single or double rotation, is determined by the kind of imbalance.
  - This rebalancing does not cause any new imbalances (why?)
- Repeat Step-2 until no imbalances remain. $O(h)$

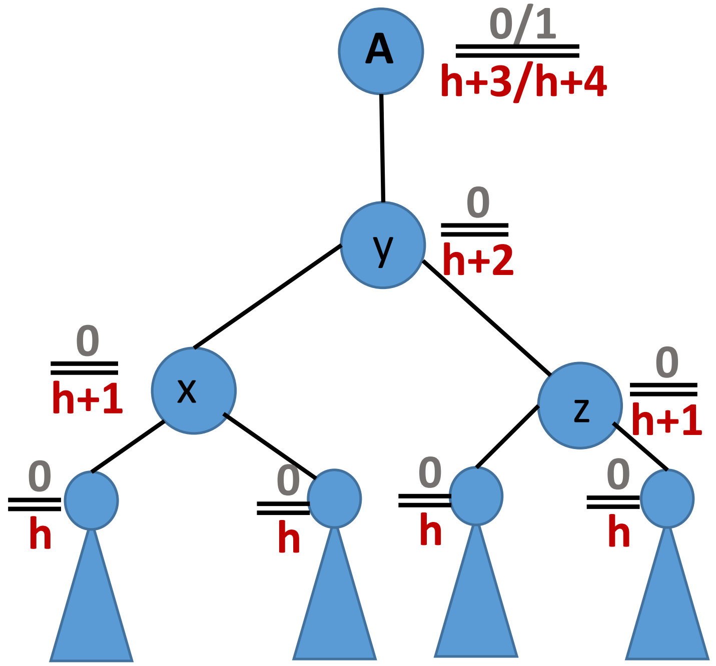
Run-time = $O(h)$ = $O(\log n)$

# Proof of correctness of the algorithm

- Is it true that the algorithm terminates? Is it true that the resultant is an AVL tree?

Sufficient to show that a rebalancing at one node does not cause any new imbalances (anywhere).

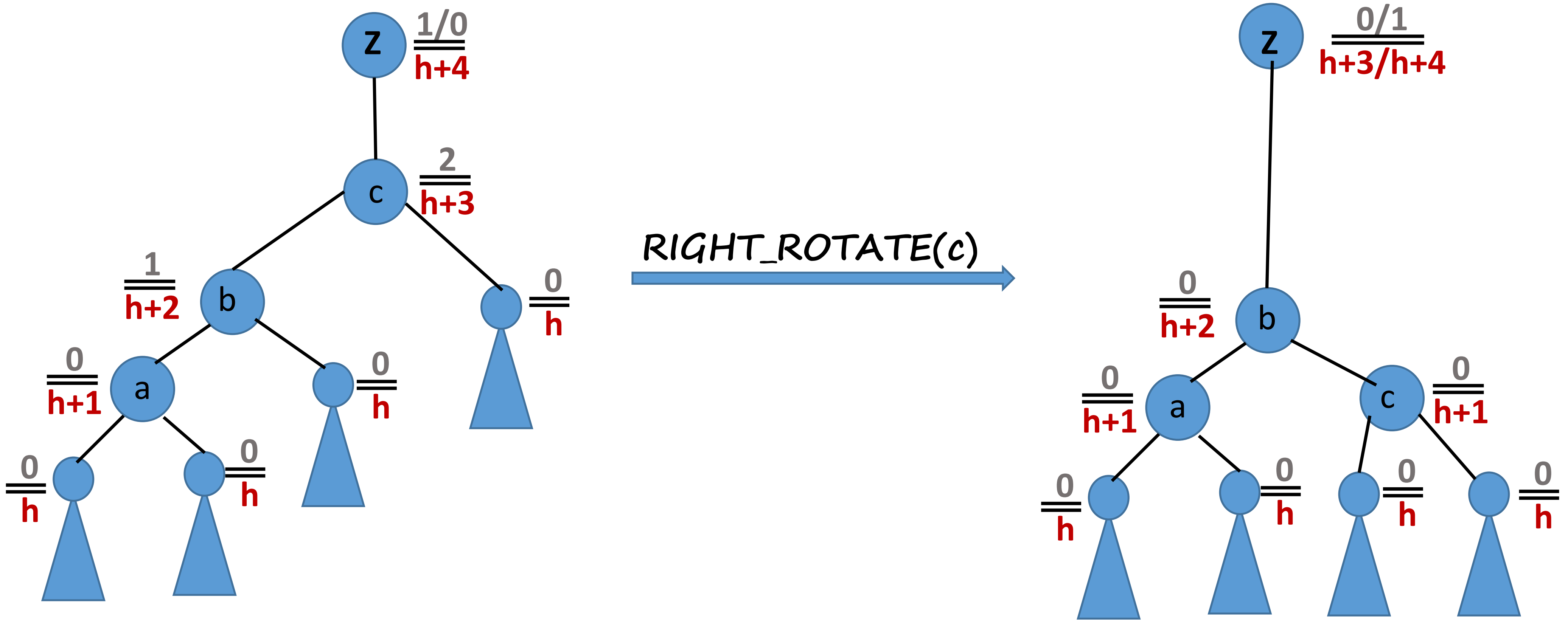
Assume that, before insertion, the balance factor (on the insertion subpath) is 0. [Similar arguments hold if the balance factor is 1.]



# Proof of correctness of the algorithm

- Case 1: Left-Left imbalance/Zig-Zig imbalance

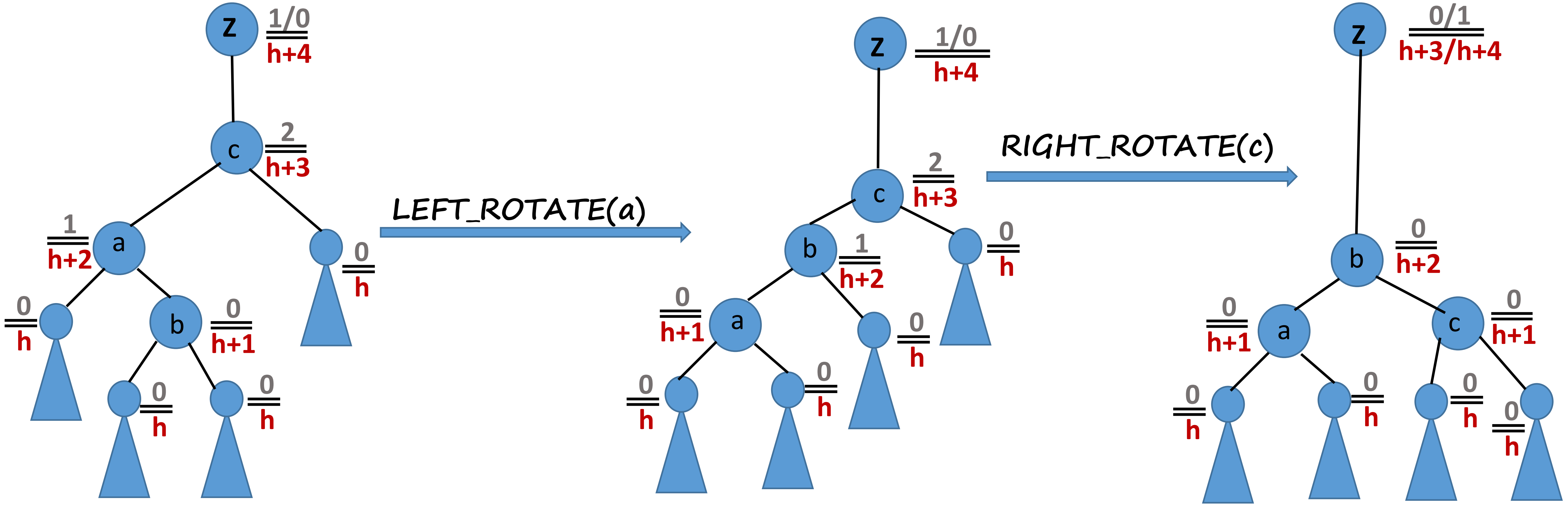
This imbalance is corrected by a single rotation – a right rotation at the node with imbalance.



# Proof of correctness of the algorithm

- *Case 2: Left-Right imbalance/Zig-Zag imbalance*

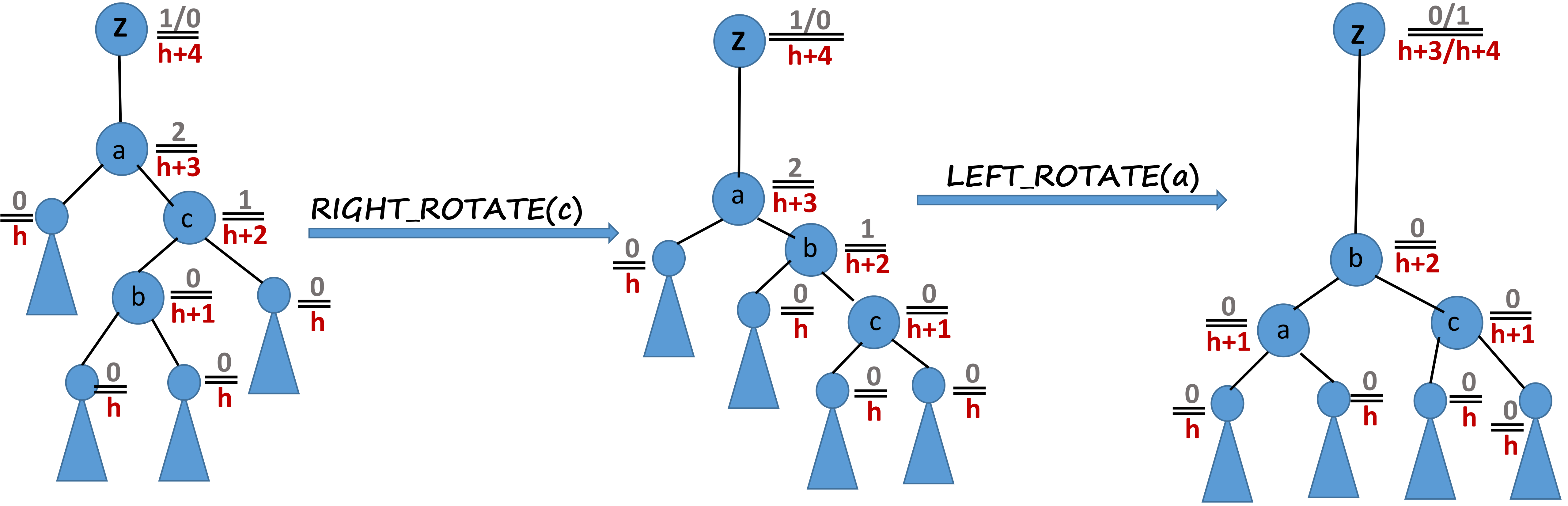
This imbalance is corrected by a double rotation – a left rotation at the left child of the imbalance and a right rotation at imbalance.



# Proof of correctness of the algorithm

- Case 3: Right-Left imbalance/Zig-Zag imbalance

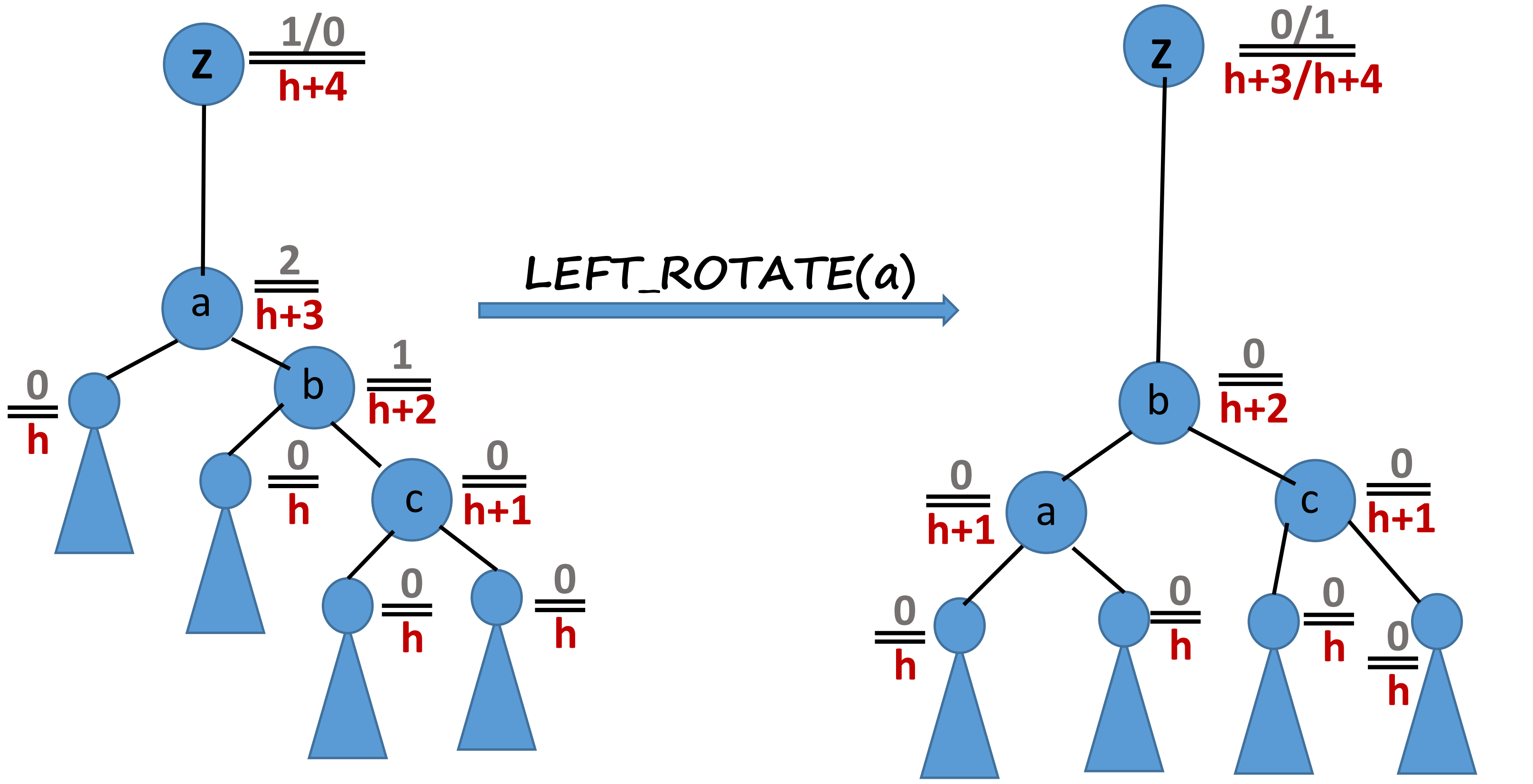
This imbalance is corrected by a double rotation – a right rotation at the right child of the imbalance and a left rotation at imbalance.



# Proof of correctness of the algorithm

- Case 4: Right-Right imbalance/Zig-Zig imbalance

This imbalance is corrected by a single rotation – a left rotation at the node with imbalance.



# Deletions in AVL trees

- Step-1: Delete as in a BST. $O(h)$
- Step-2: Check if there is any height imbalance. $O(n^2)$ w/o augmentation, $O(1)$ with augmentation
  - Insertion decreases the height of the tree by at most one.
  - If there is a decrease in the height (of the tree) then there is a height imbalance at some nodes on the search path; conversely, if there is a height imbalance at a node (on the search path) then there is a decrease in the height of some subtrees on the search path.

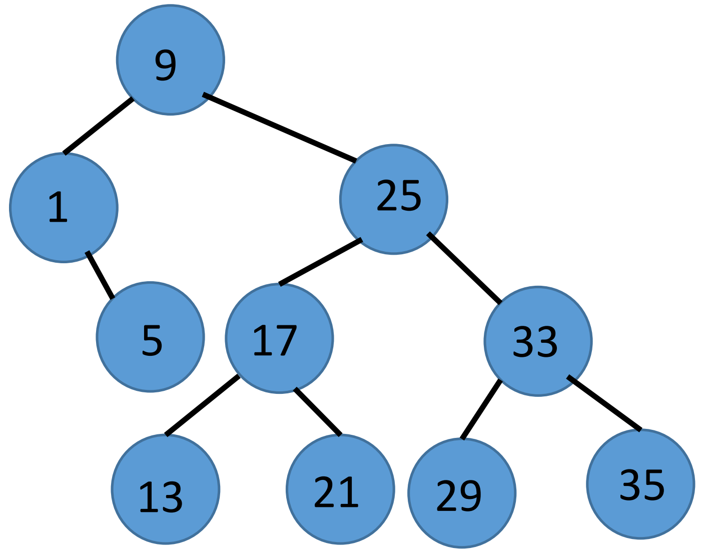
Note: Height imbalance, if any, will be on some nodes on the search path.

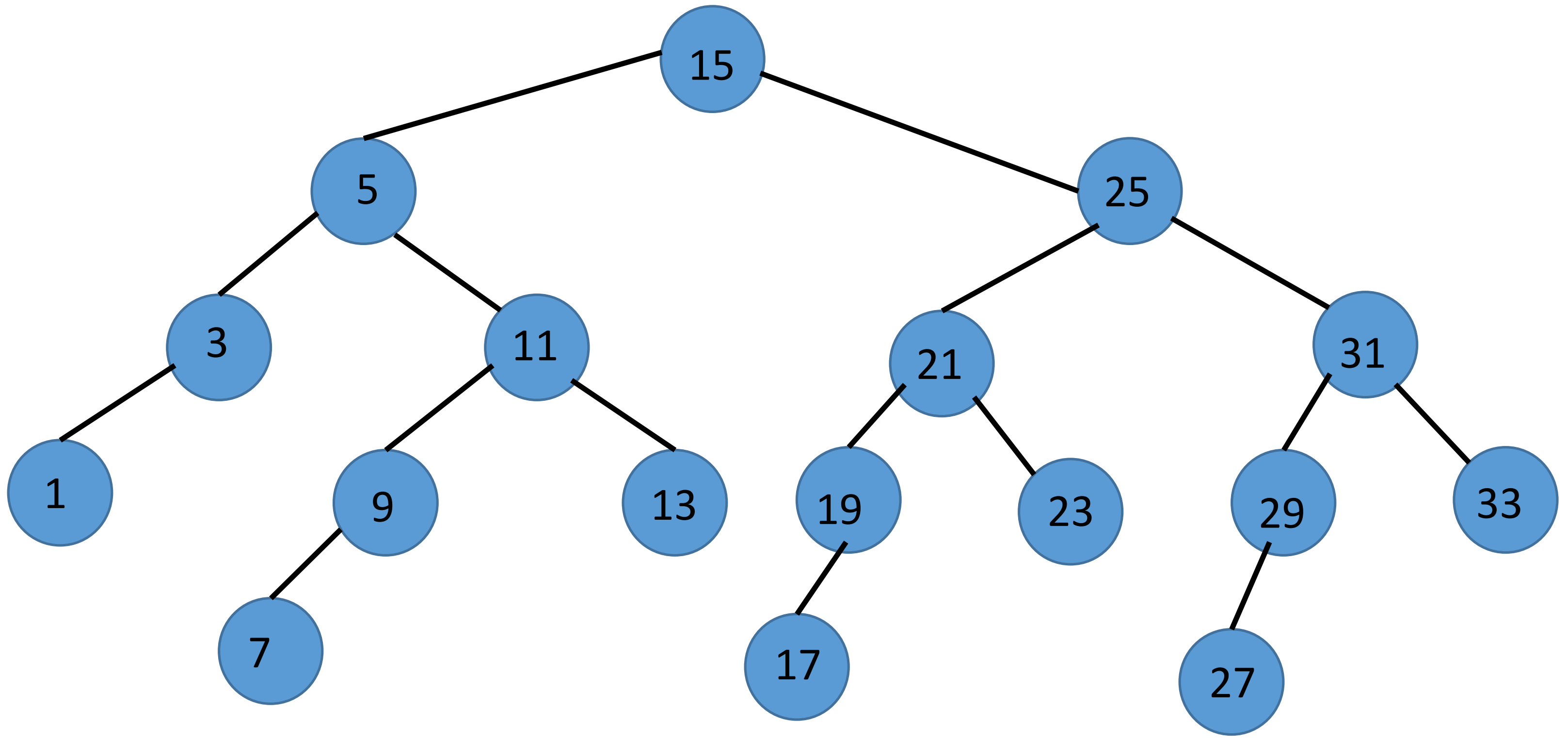
- Step-2.1: Rebalance the deepest node at which the imbalance is present. $O(1)$
  - The kind of rebalancing; i.e., single or double rotation, is determined by the kind of imbalance.
  - Since imbalance is caused due to reduction in height, a rebalancing may propagate (upwards) the imbalance to an ancestor along the search path
- Repeat Step-2 until no imbalances remain. $O(h)$

Run-time = $O(h) = O(\log n)$

# Deletions into AVL trees (Exercise)

Delete 1 from the following AVL trees





| Node | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 33 | 35 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Parent | 3 | 5 | 15 | 9 | 5 | 9 | 11 | 41 | 21 | 17 | 31 | 25 | 21 | 25 | 27 | 15 | 35 | 31 |
| **Node** | 37 | 39 | 41 | 43 | 45 | 47 | 49 | 51 | 53 | 55 | 57 | 59 | 61 | 63 | 65 | 67 | 69 | 71 |
| **Parent** | 35 | 37 | -1 | 45 | 51 | 45 | 47 | 67 | 55 | 61 | 59 | 55 | 51 | 65 | 61 | 41 | 73 | 69 |
| **Node** | 73 | 75 | 77 | 79 | 81 | 83 | 85 | 87 | 89 | 91 | 93 | 95 | 97 | 99 | 101 | 103 | 105 | 107 |
| **Parent** | 83 | 79 | 75 | 73 | 79 | 67 | 89 | 85 | 93 | 89 | 83 | 97 | 103 | 101 | 97 | 93 | 103 | 105 |