FAST SEARCHING / BALANCED TREES

Stefan Grünberger, Markus Weninger
Martin Schobesberger, Dari Trendafilov

Institute of Pervasive Computing
Johannes Kepler University Linz
teaching@pervasive.jku.at

Algorithms and Data Structures 2
Exercise – 2021W

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

BALANCED TREES

Motivation

- Real data is usually not randomly distributed
- Prevent **degeneration** of binary search trees into linear lists!
- Search/Insert in **O(log(n))**

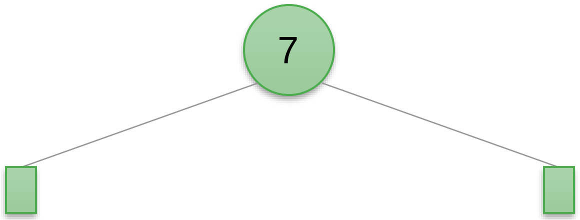
Approach

- Monitor tree structure
- Insert/Remove may require restructuring

Binary search trees that guarantee the execution of search, insert and delete operations in **O(log(n))** even in worst case → height-balanced trees (e.g.: **AVL tree**)

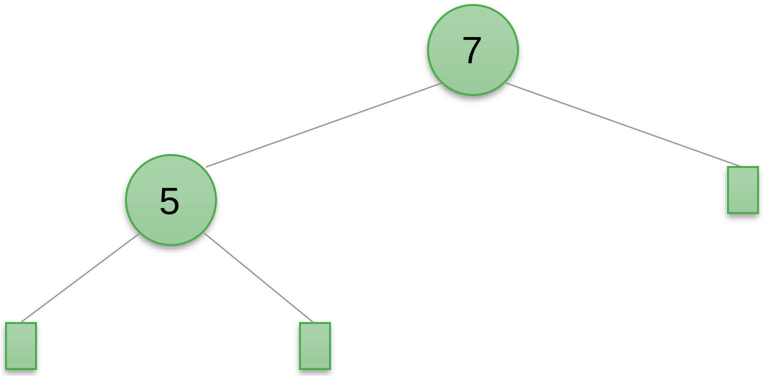
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



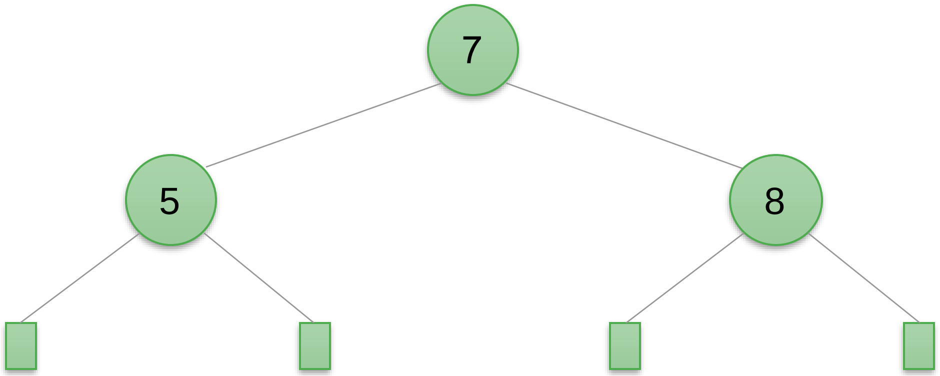
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



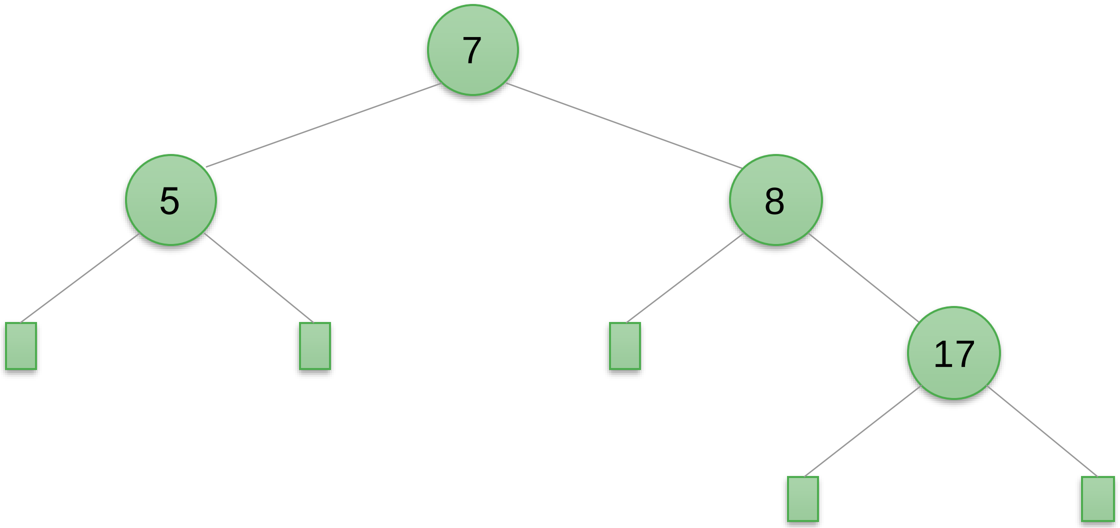
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



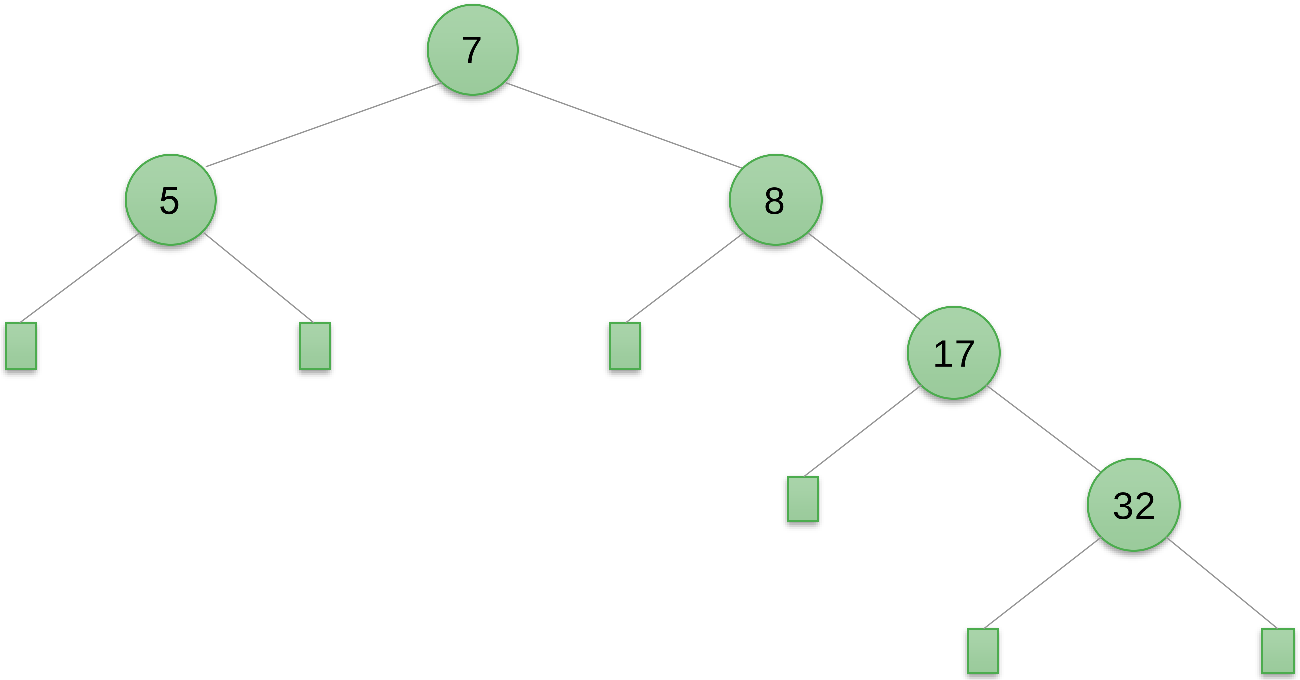
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



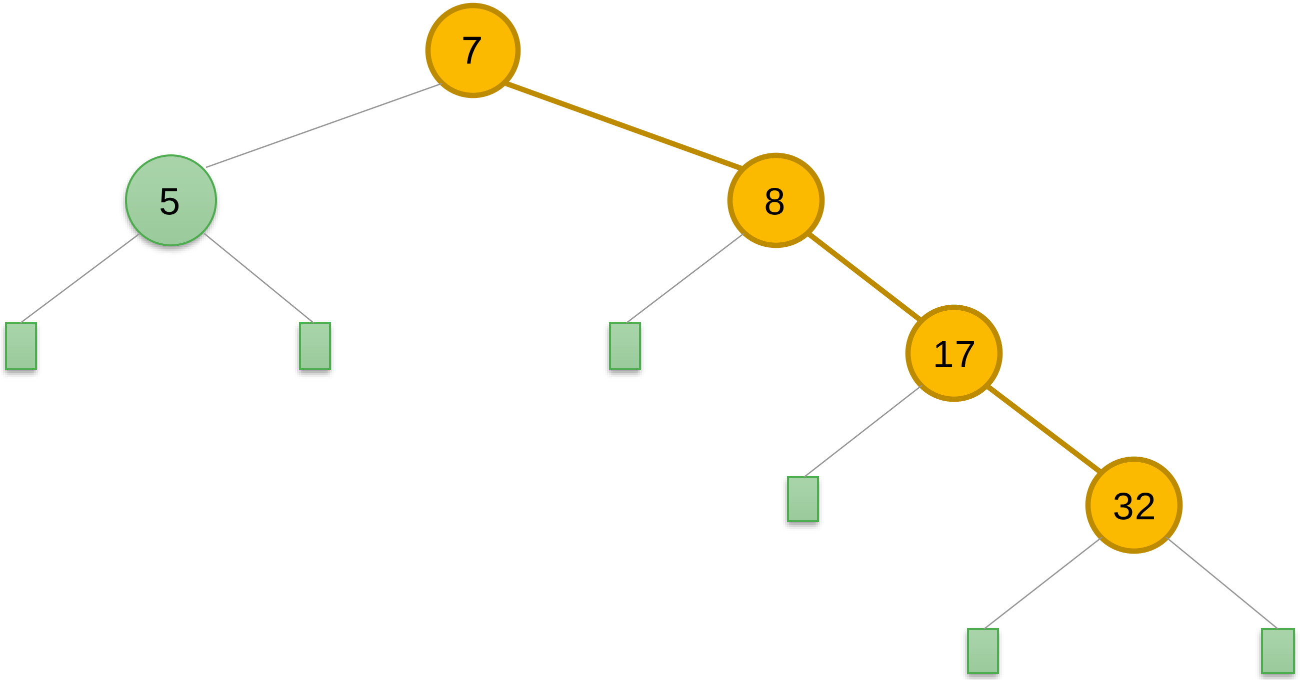
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



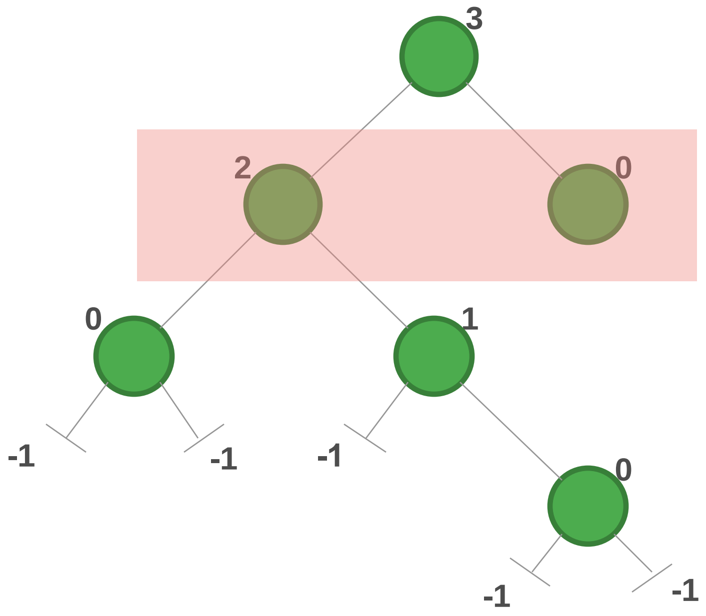
List: Access O(n)

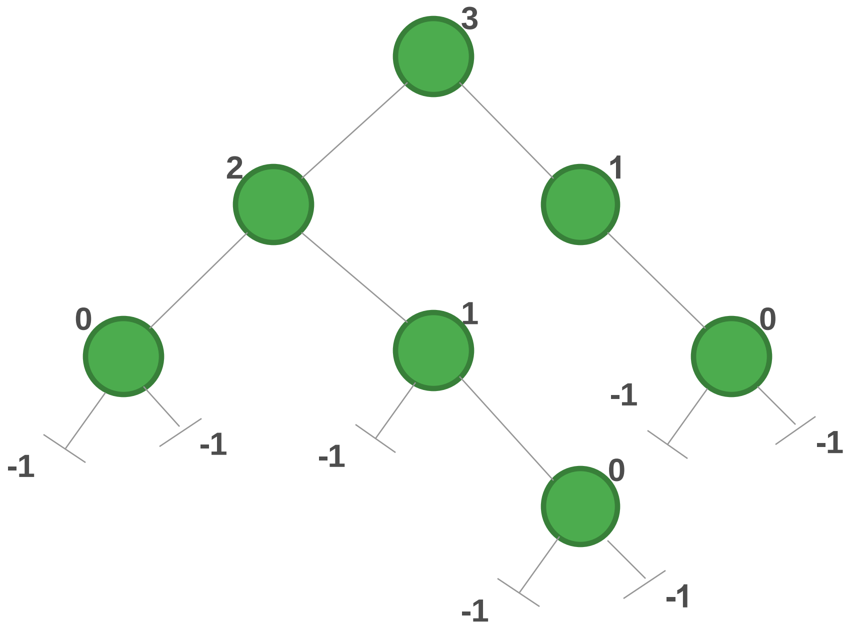
AVL TREE

Properties

- Binary search tree
- for each node, the heights of its two subtrees differ by not more than 1 („balanced“)

Examples





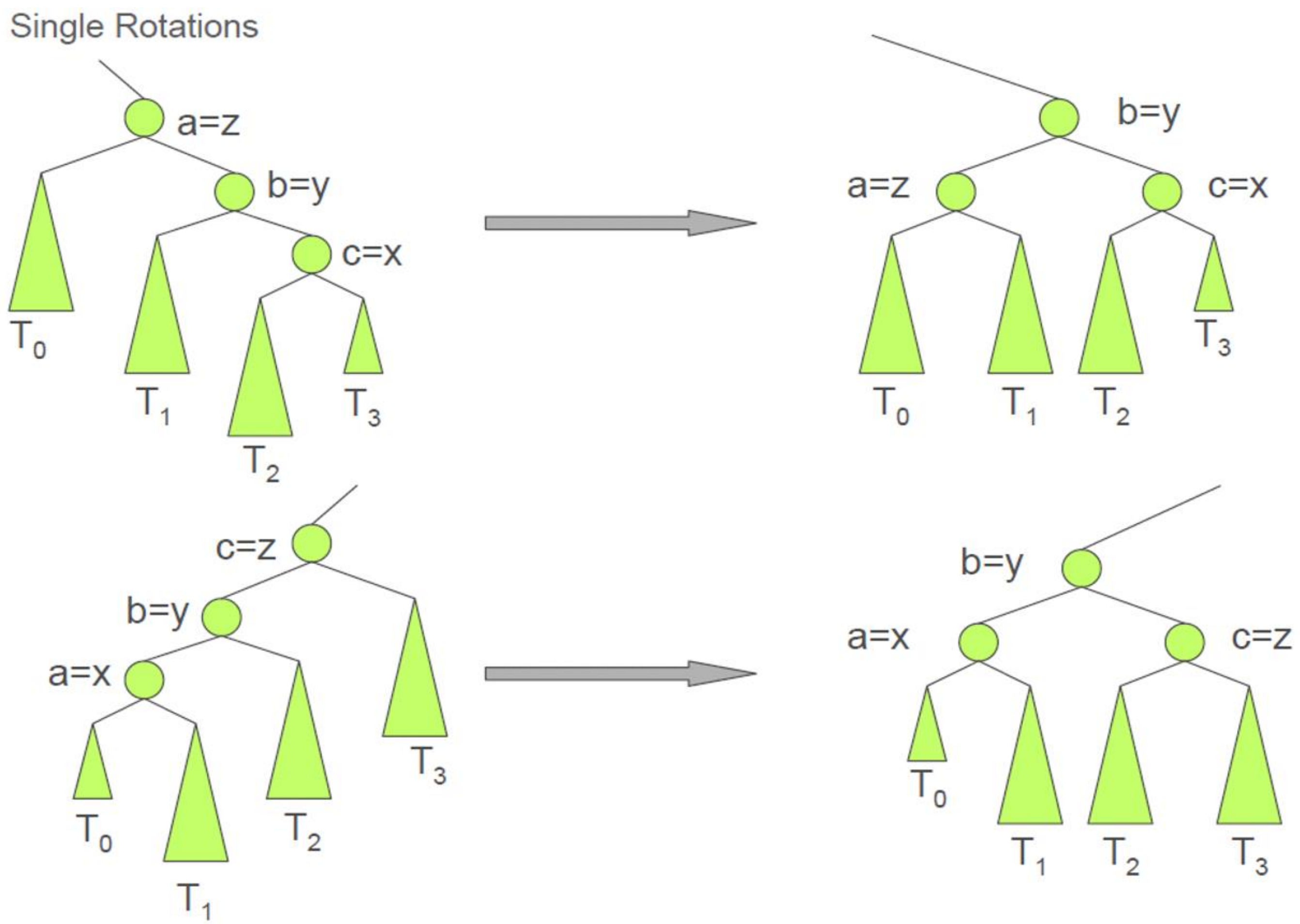
AVL TREE :: INSERT

Insert is in general the same as for the binary search tree but may cause the AVL tree to become unbalanced → restructuring required!

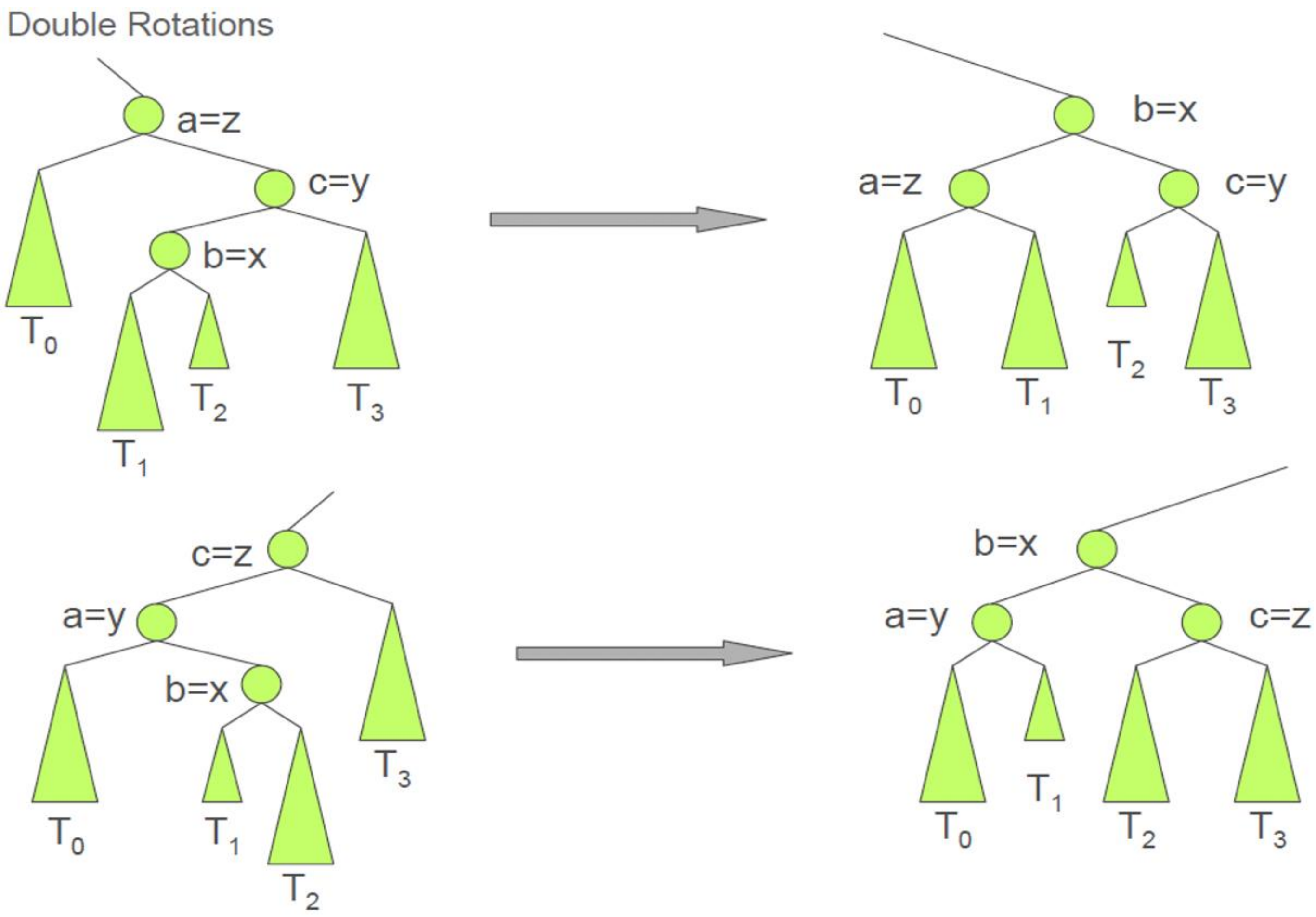
Restructuring

1. Go up from the new node in the tree until the first node **x** is found, whose grandparent **z** is an unbalanced node
2. Define **y** as child of **z** (= the node we passed on the way to z);
   height(y) = height(sibling(y))+2
3. Define **x** as child of **y**
4. Rename **x**,**y**,**z** in **a**,**b**,**c** (according to Inorder traversal!)
5. Replace **z** by **b**
6. Children of **b** are now **a** (left) and **c** (right)
7. Children of **a** and **c** are the subtrees T_0 … T_3, which have been children of **x**, **y** and **z** before → reassign and distinguish **4 cases…**

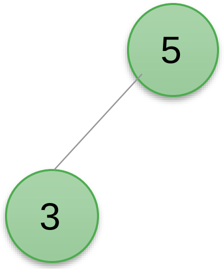
AVL TREE :: ROTATIONS



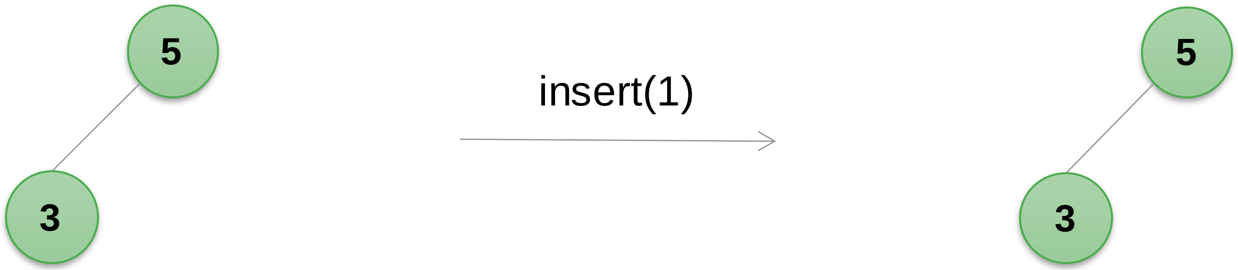
AVL TREE :: ROTATIONS



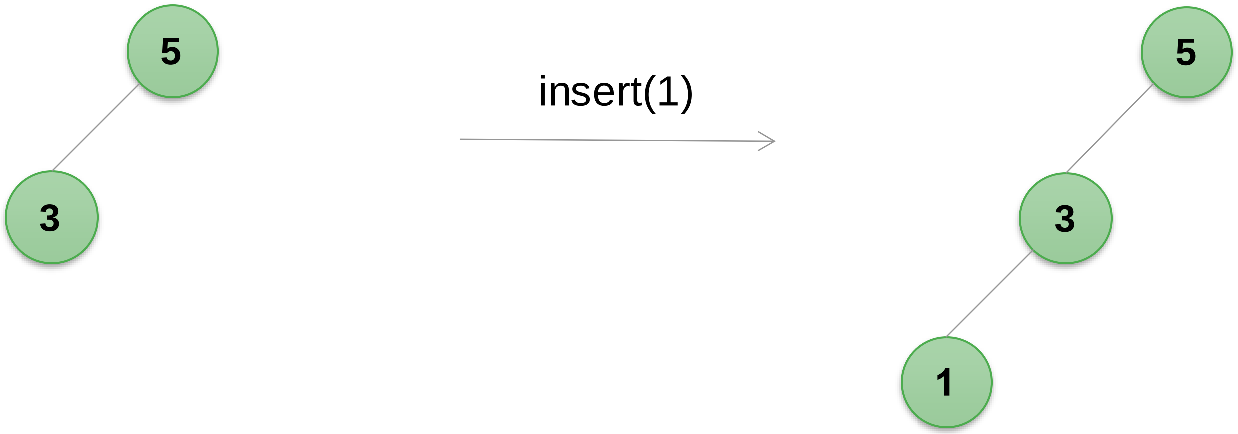
AVL TREE :: ROTATIONS



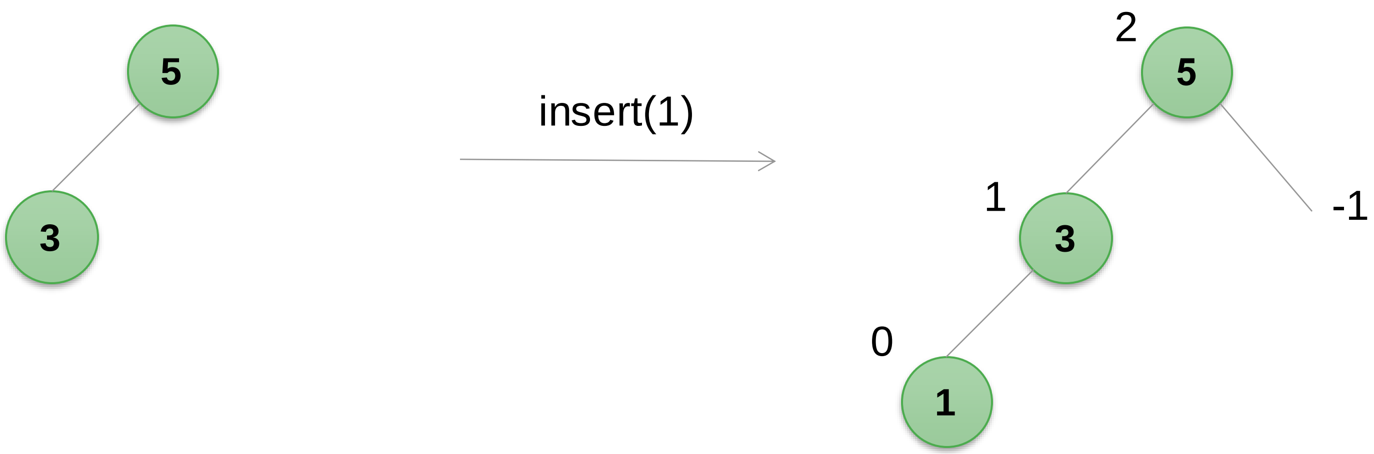
AVL TREE :: ROTATIONS



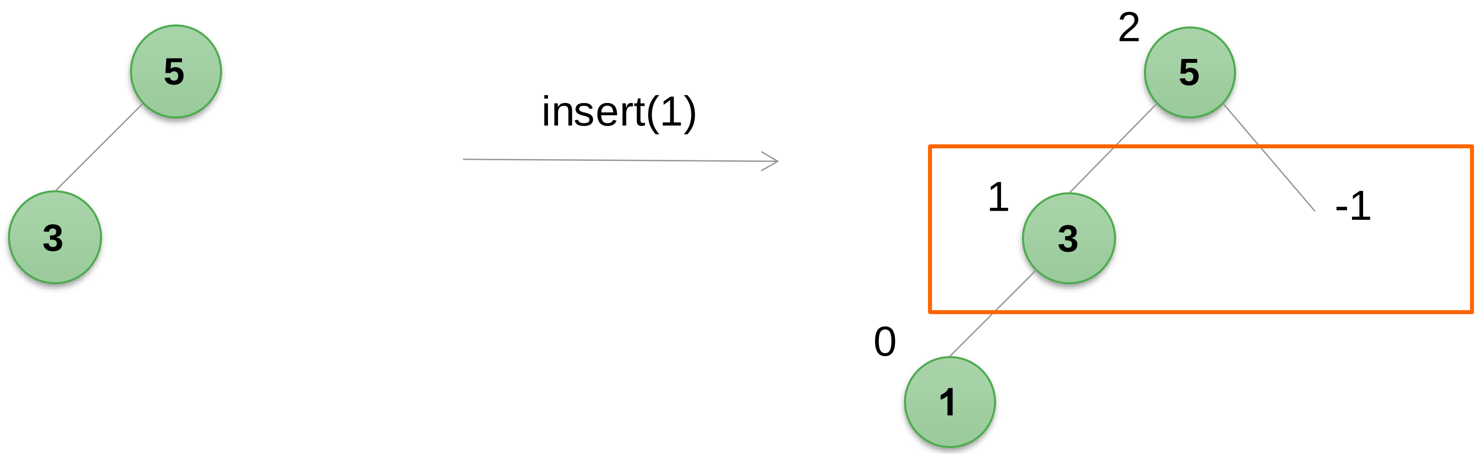
AVL TREE :: ROTATIONS



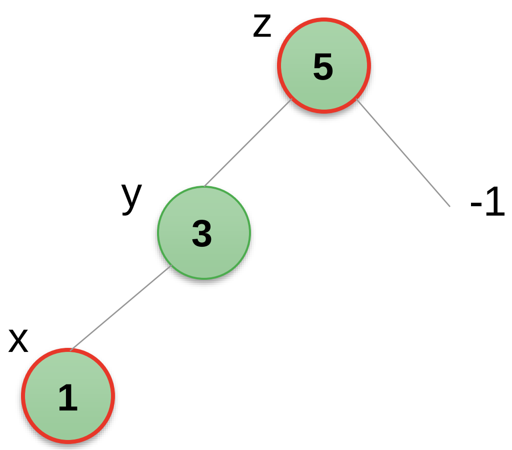
AVL TREE :: ROTATIONS



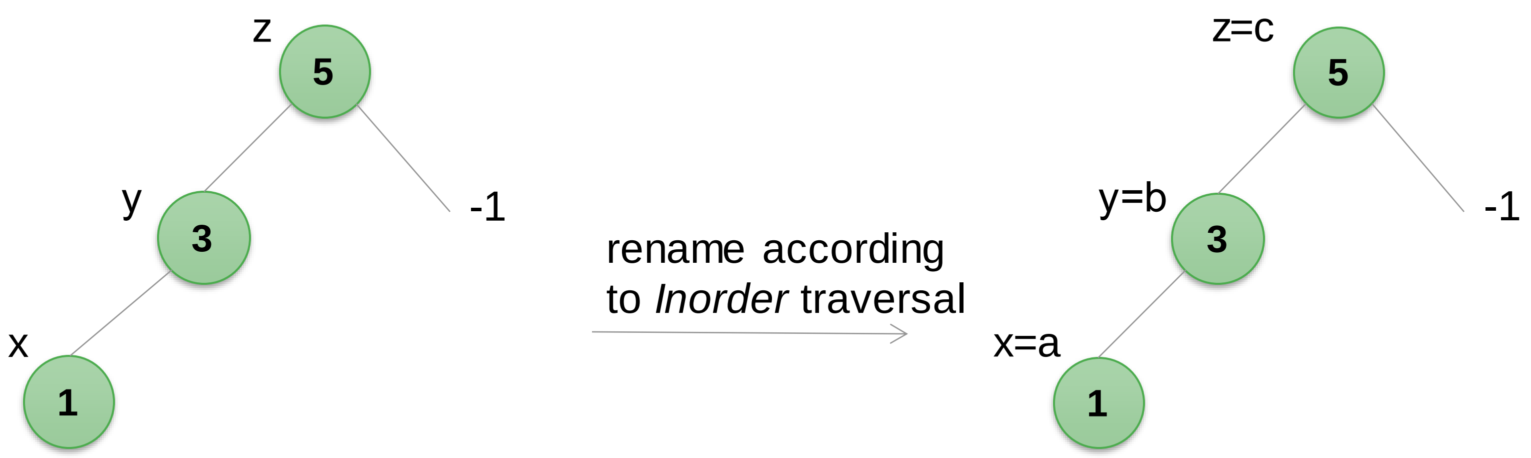
AVL TREE :: ROTATIONS



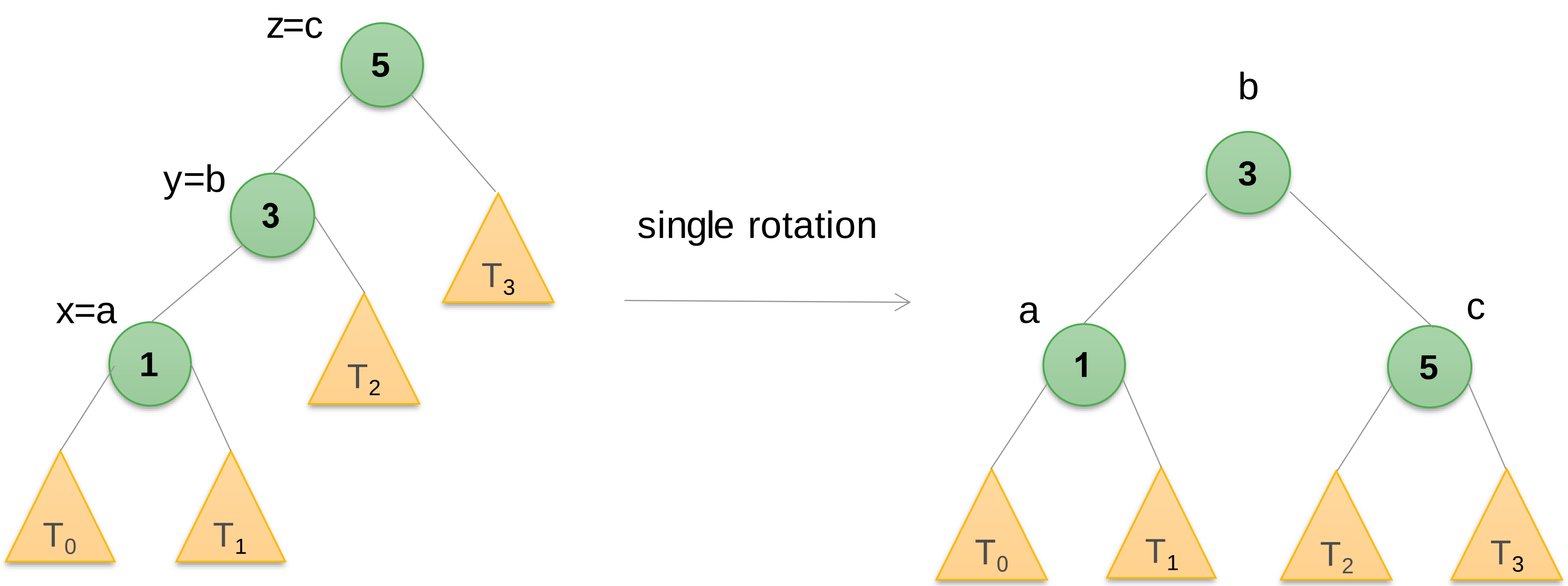
AVL TREE :: ROTATIONS



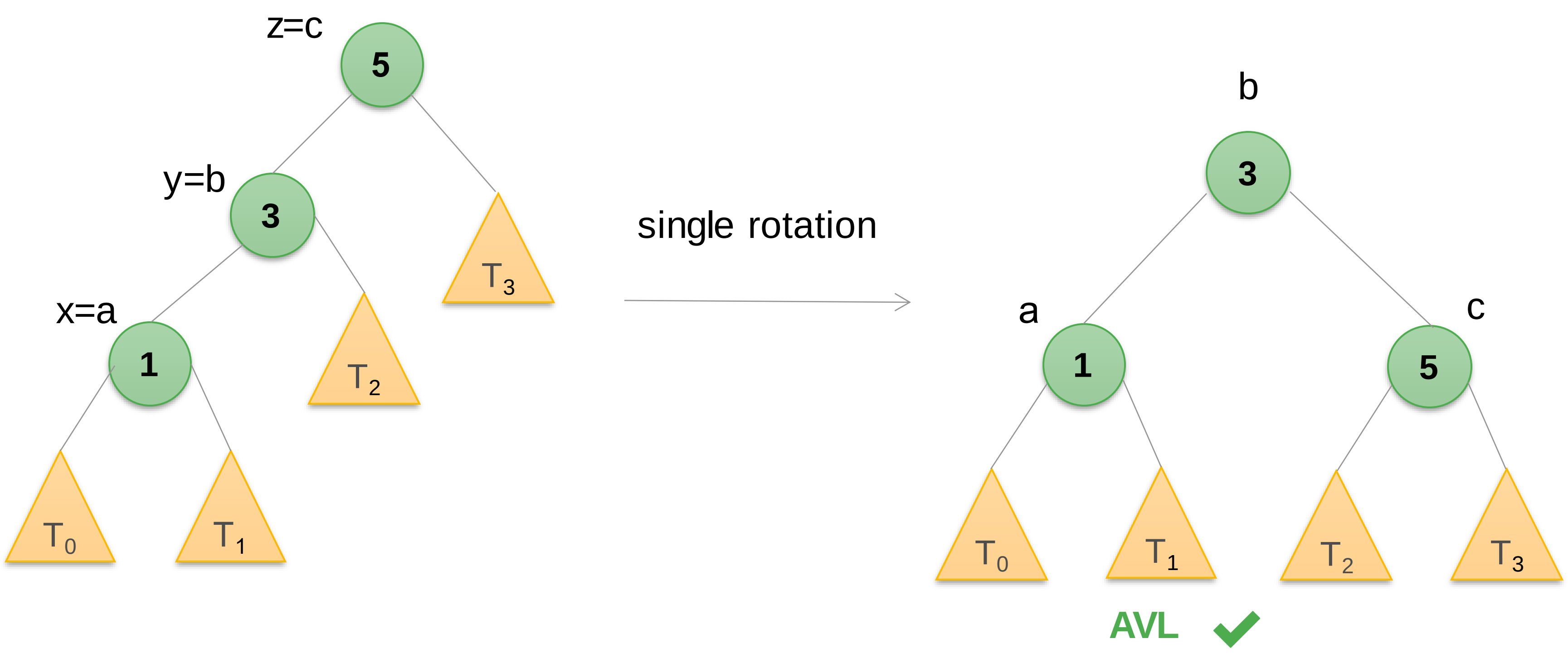
AVL TREE :: ROTATIONS



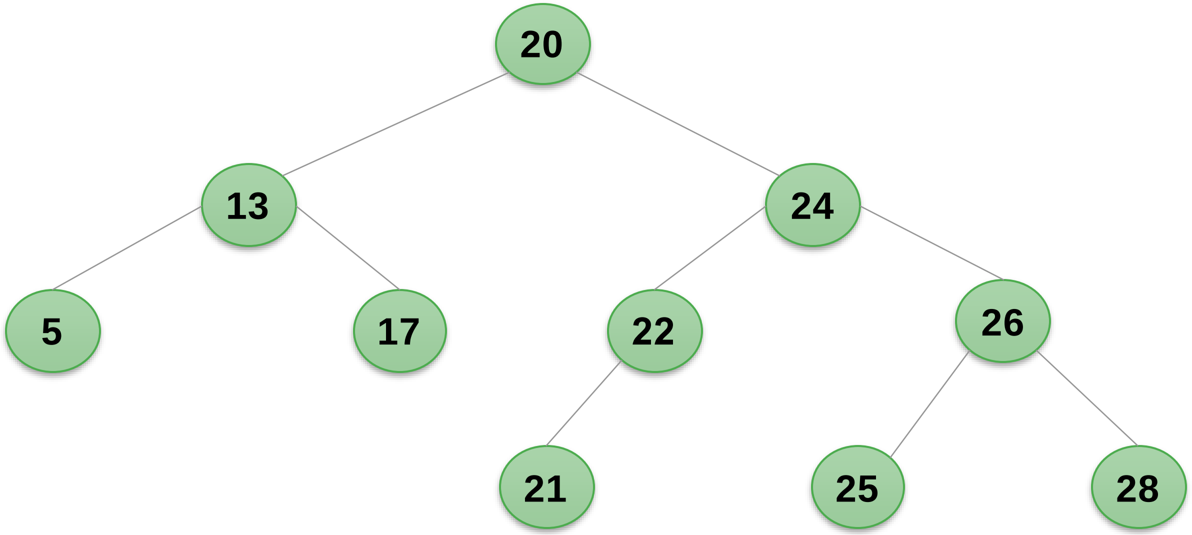
AVL TREE :: ROTATIONS



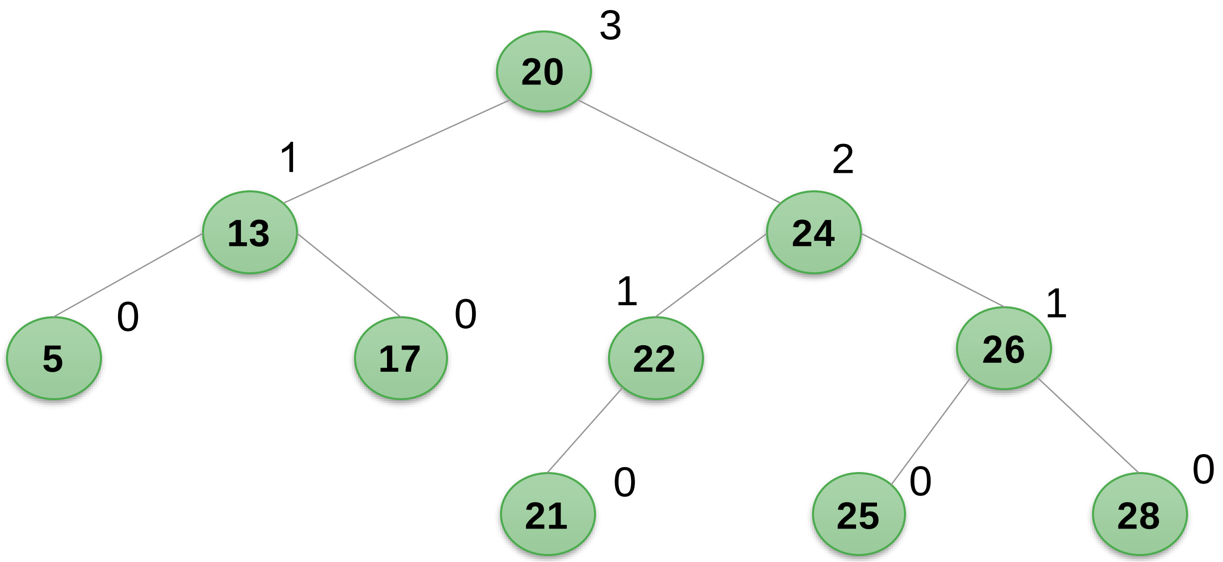
AVL TREE :: ROTATIONS



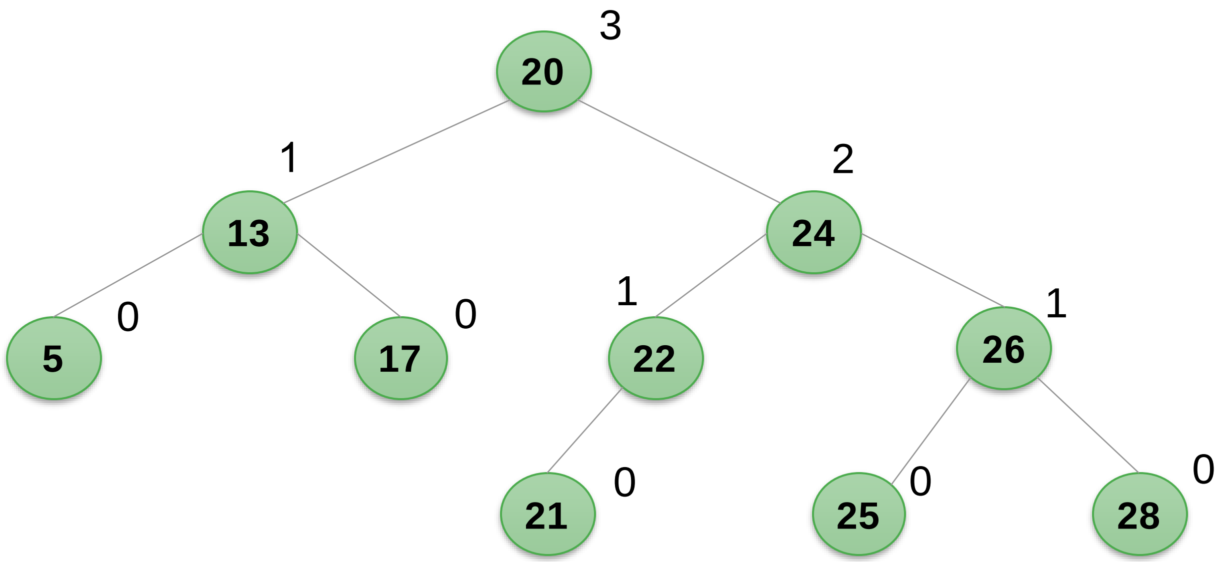
AVL TREE :: SINGLE ROTATION EXAMPLE



AVL TREE :: SINGLE ROTATION EXAMPLE

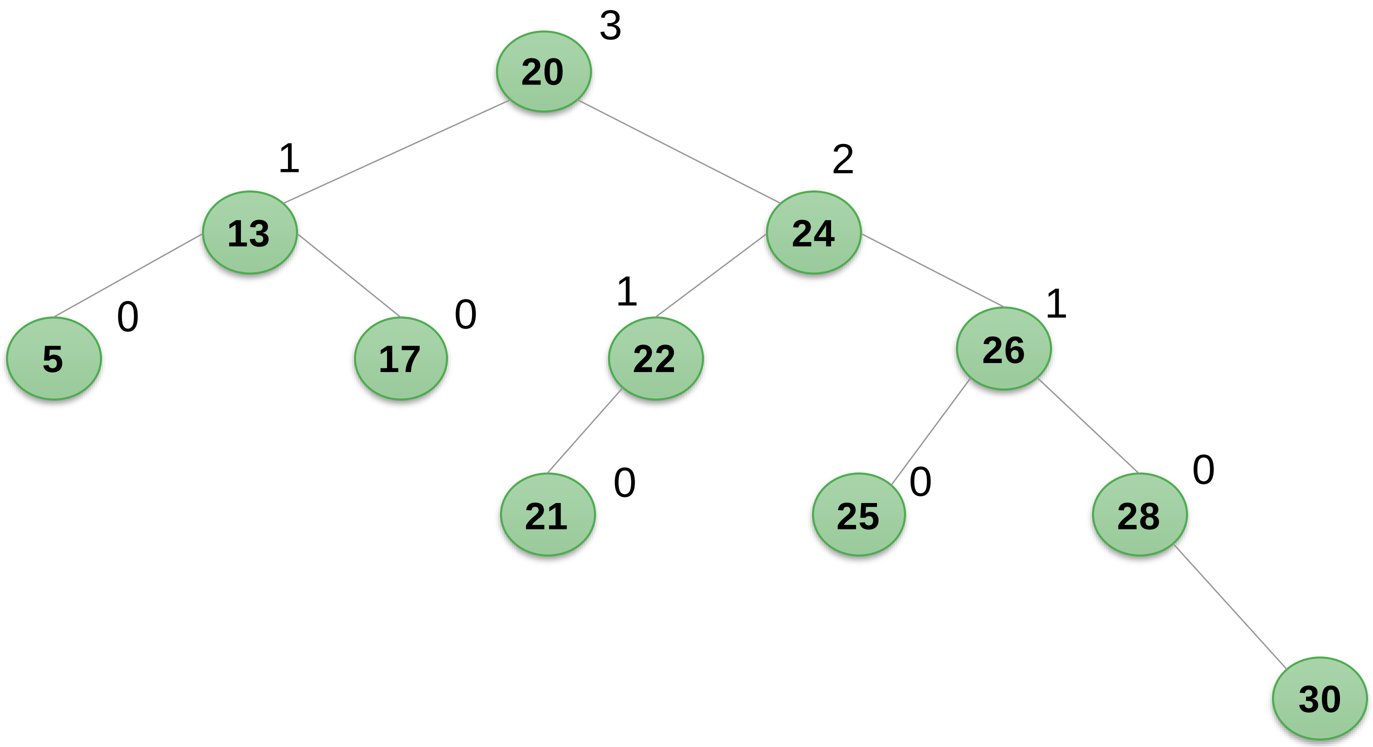


AVL TREE :: SINGLE ROTATION EXAMPLE



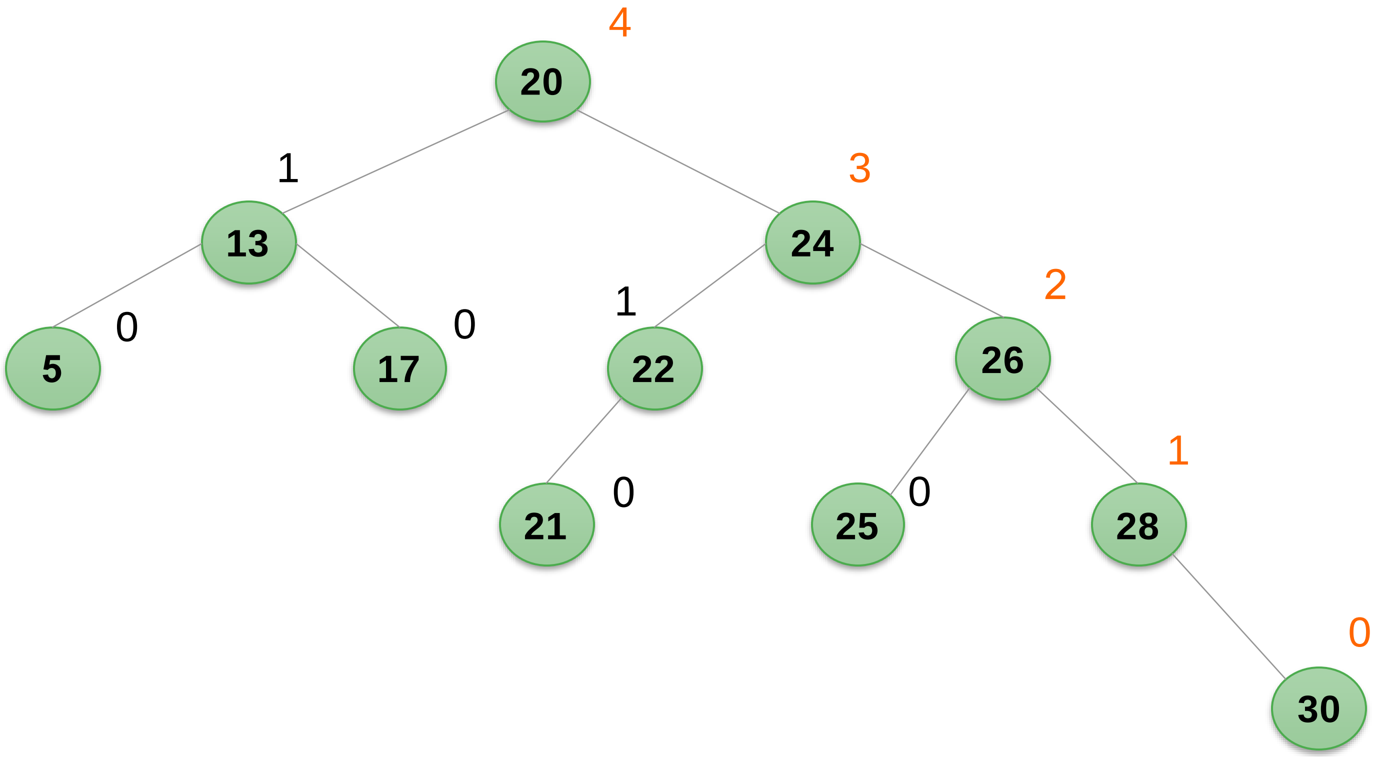
insert ‘30’

AVL TREE :: SINGLE ROTATION EXAMPLE



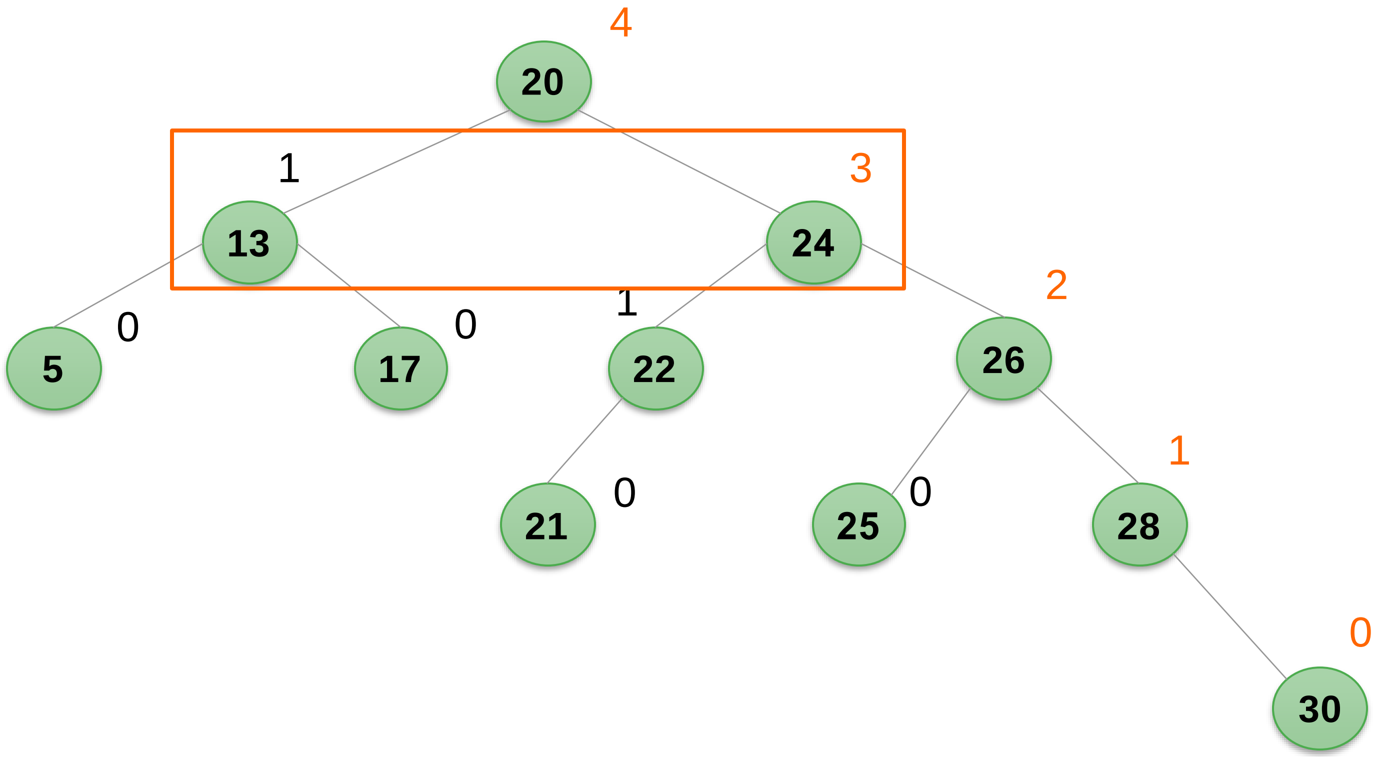
insert '30'

AVL TREE :: SINGLE ROTATION EXAMPLE



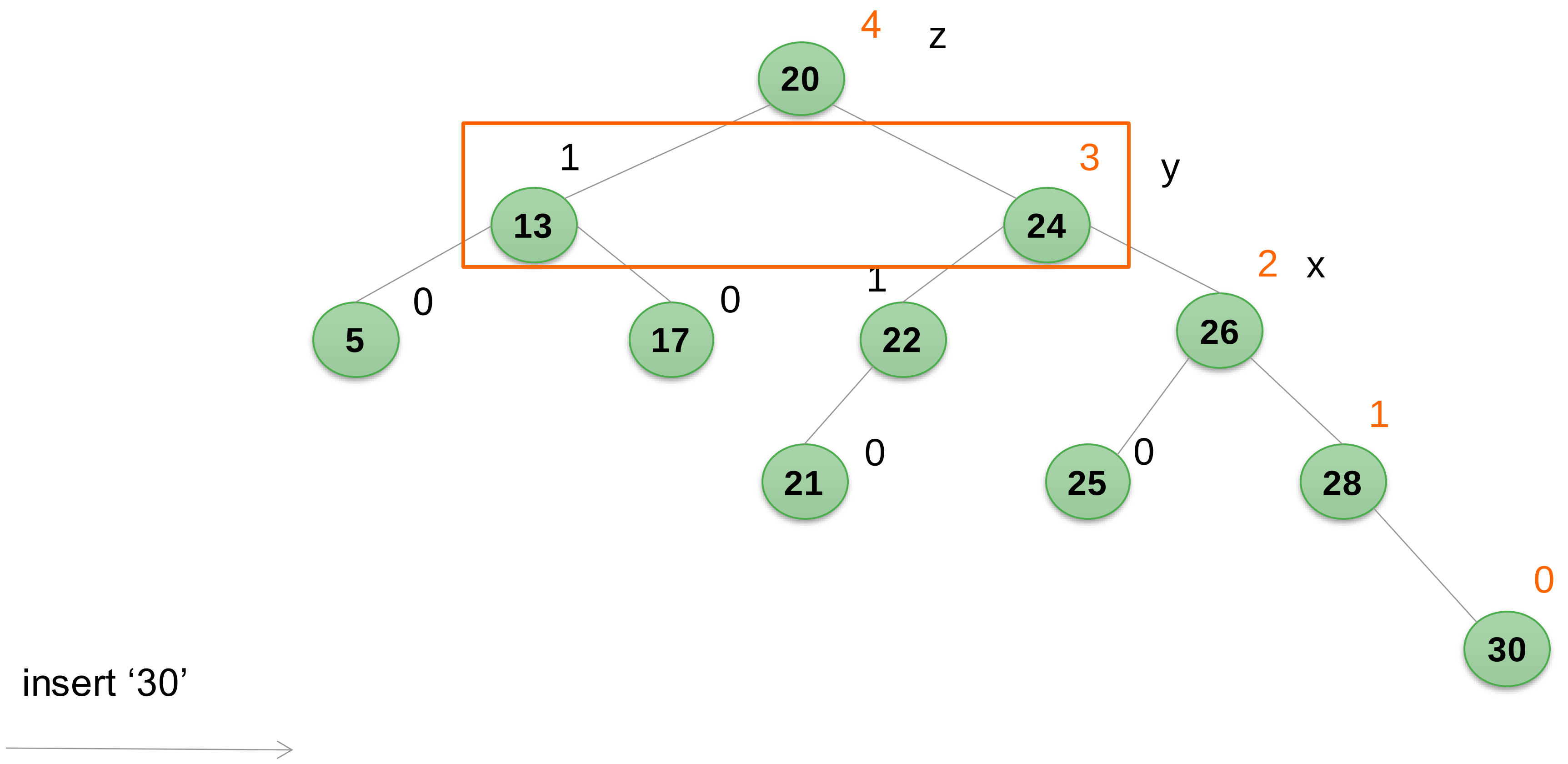
insert ‘30’

AVL TREE :: SINGLE ROTATION EXAMPLE

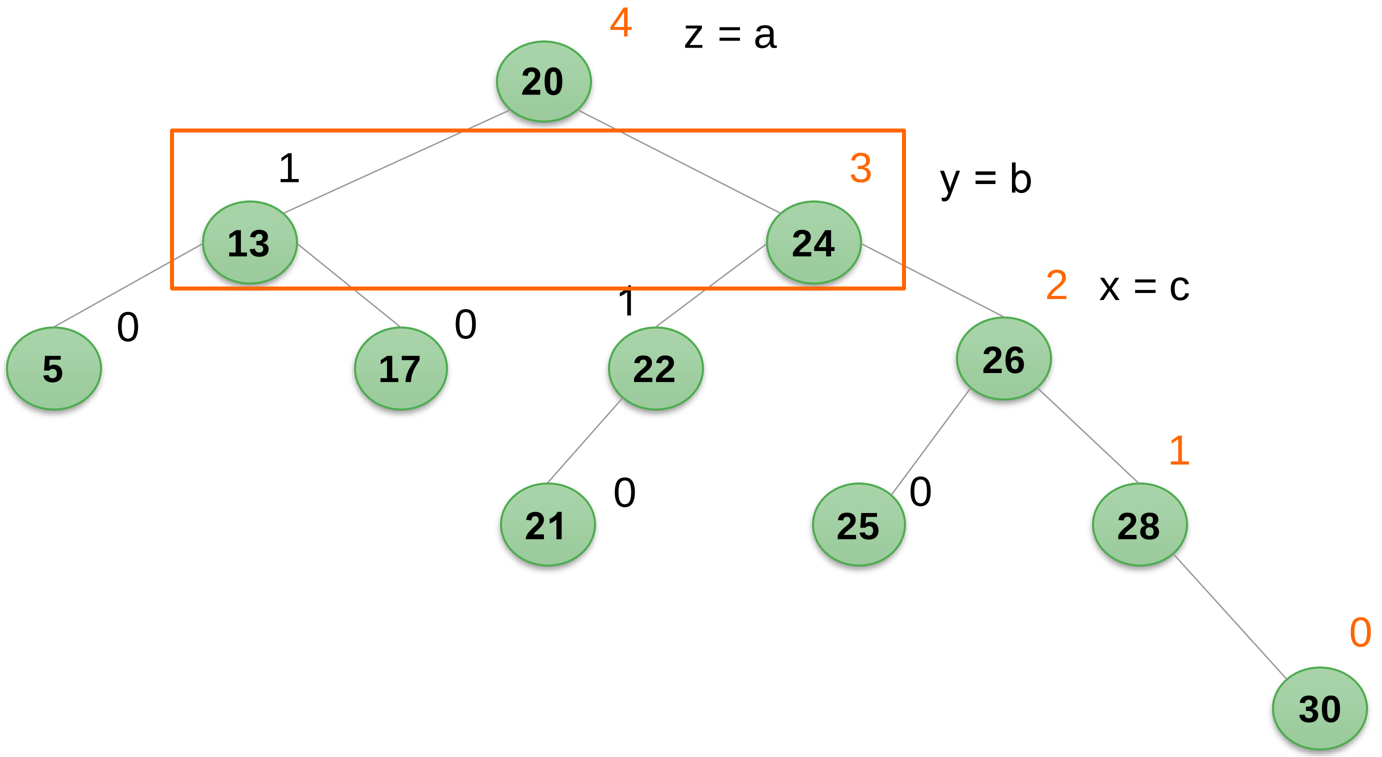


insert ‘30’

AVL TREE :: SINGLE ROTATION EXAMPLE

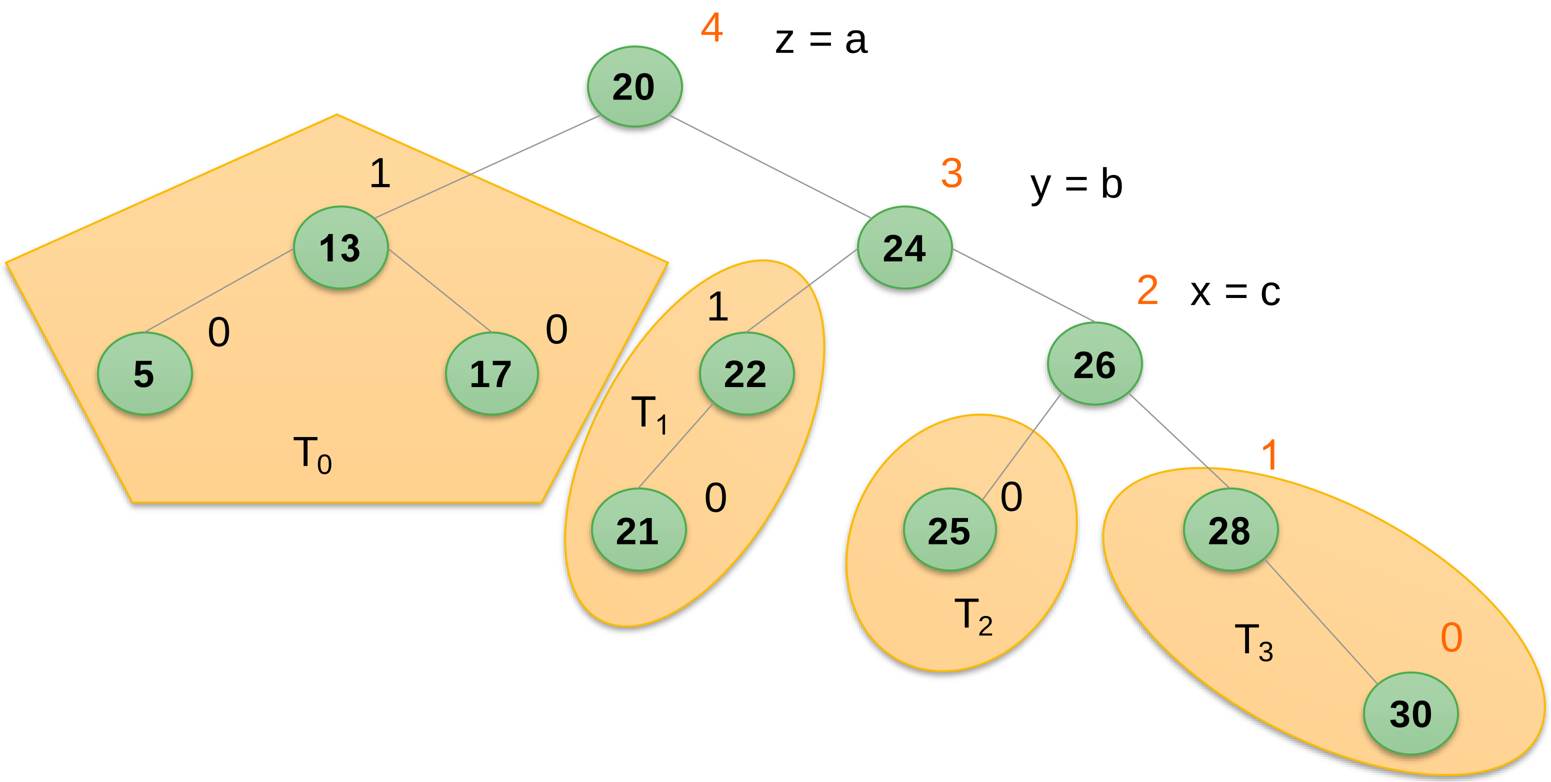


AVL TREE :: SINGLE ROTATION EXAMPLE

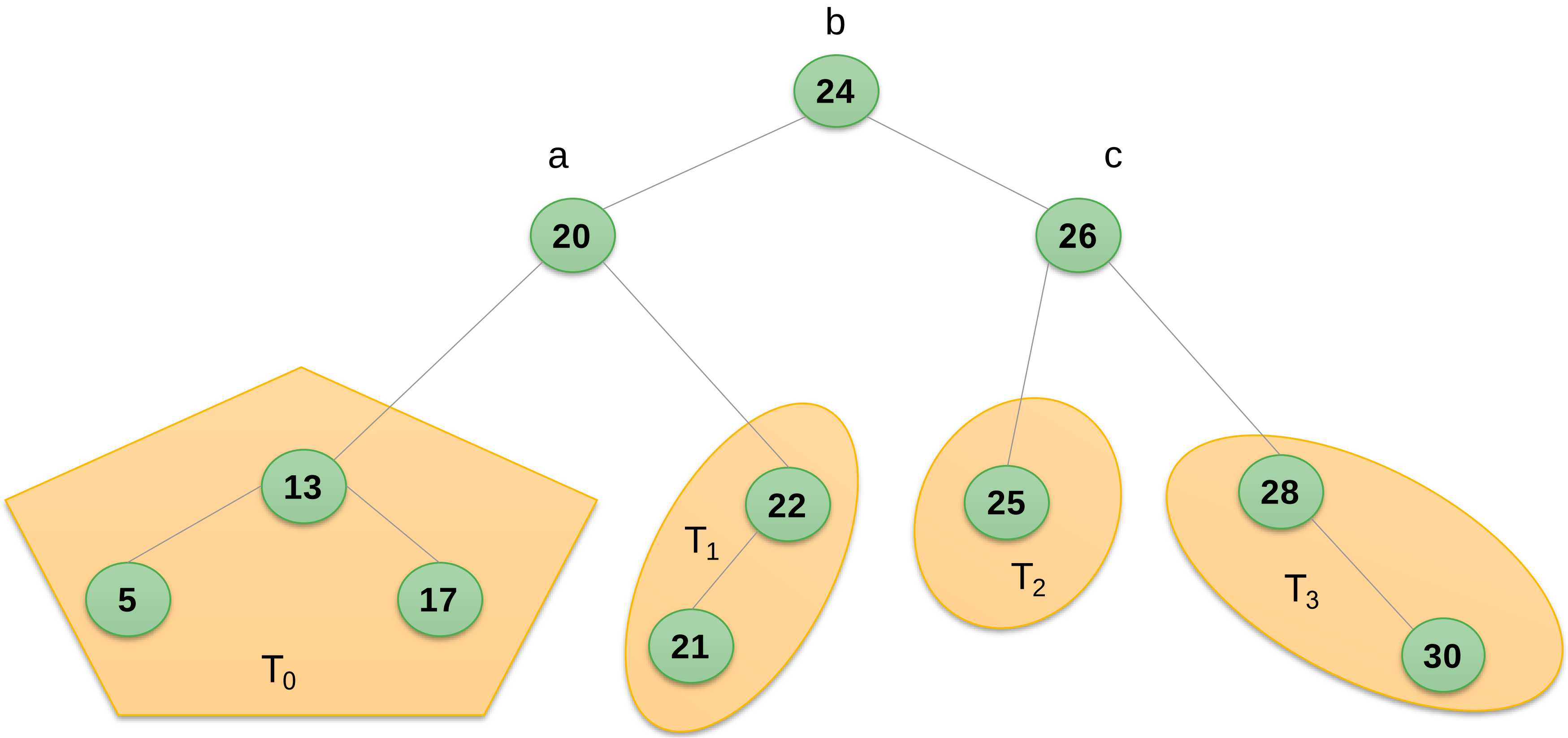


insert ‘30’

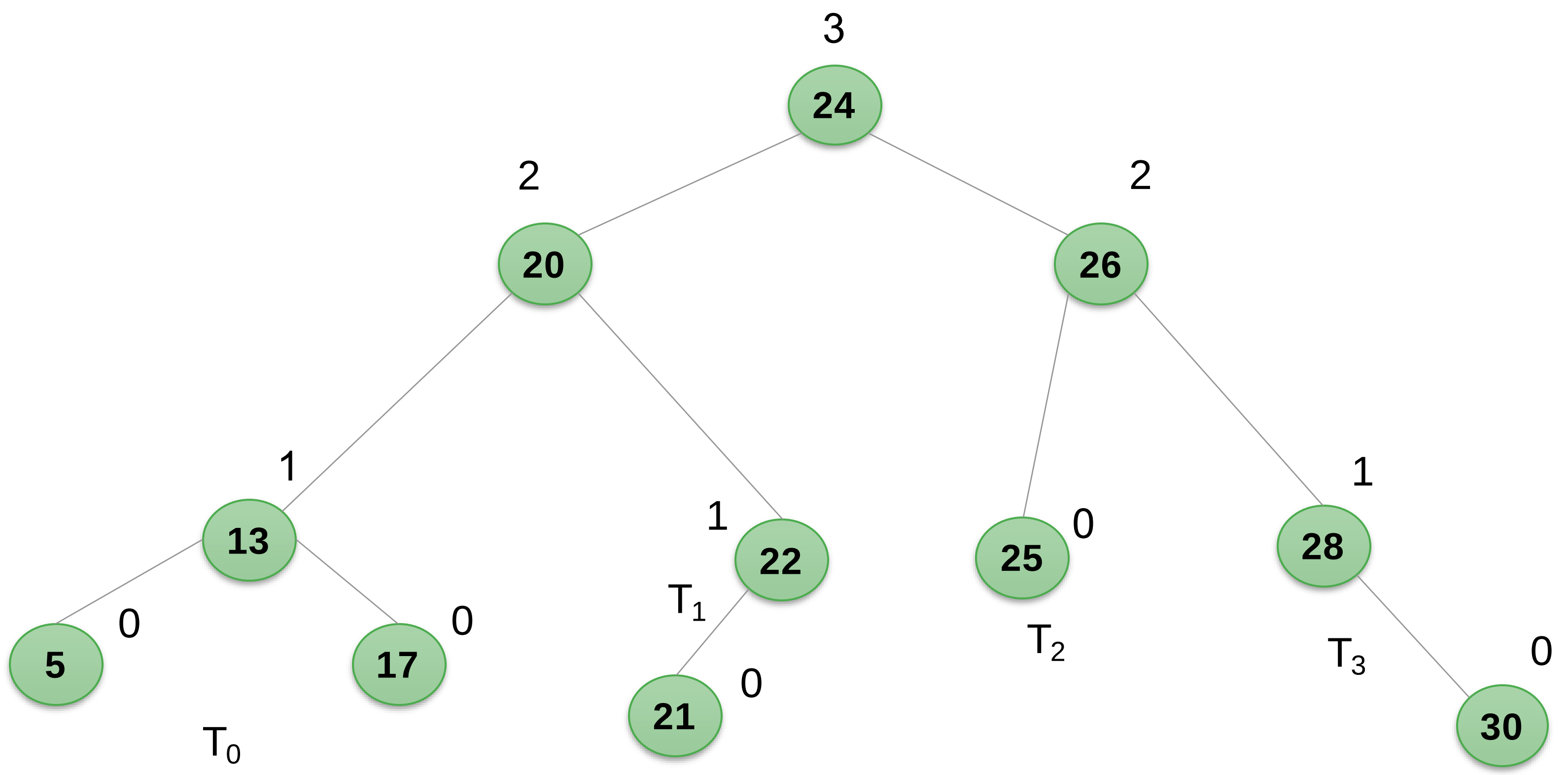
AVL TREE :: SINGLE ROTATION EXAMPLE



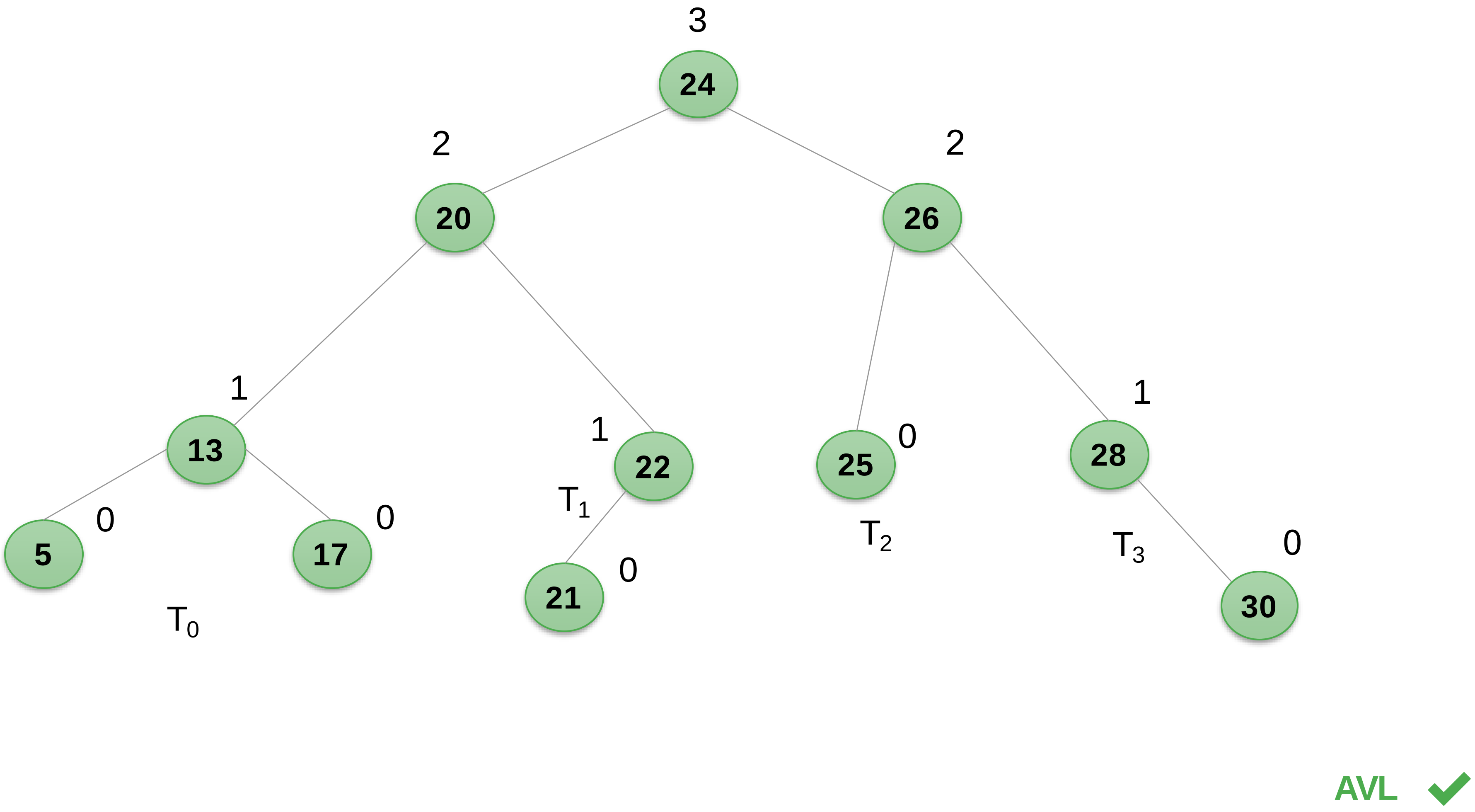
AVL TREE :: SINGLE ROTATION EXAMPLE



AVL TREE :: SINGLE ROTATION EXAMPLE



AVL TREE :: SINGLE ROTATION EXAMPLE



AVL TREES :: CUT & LINK RESTRUCTURING

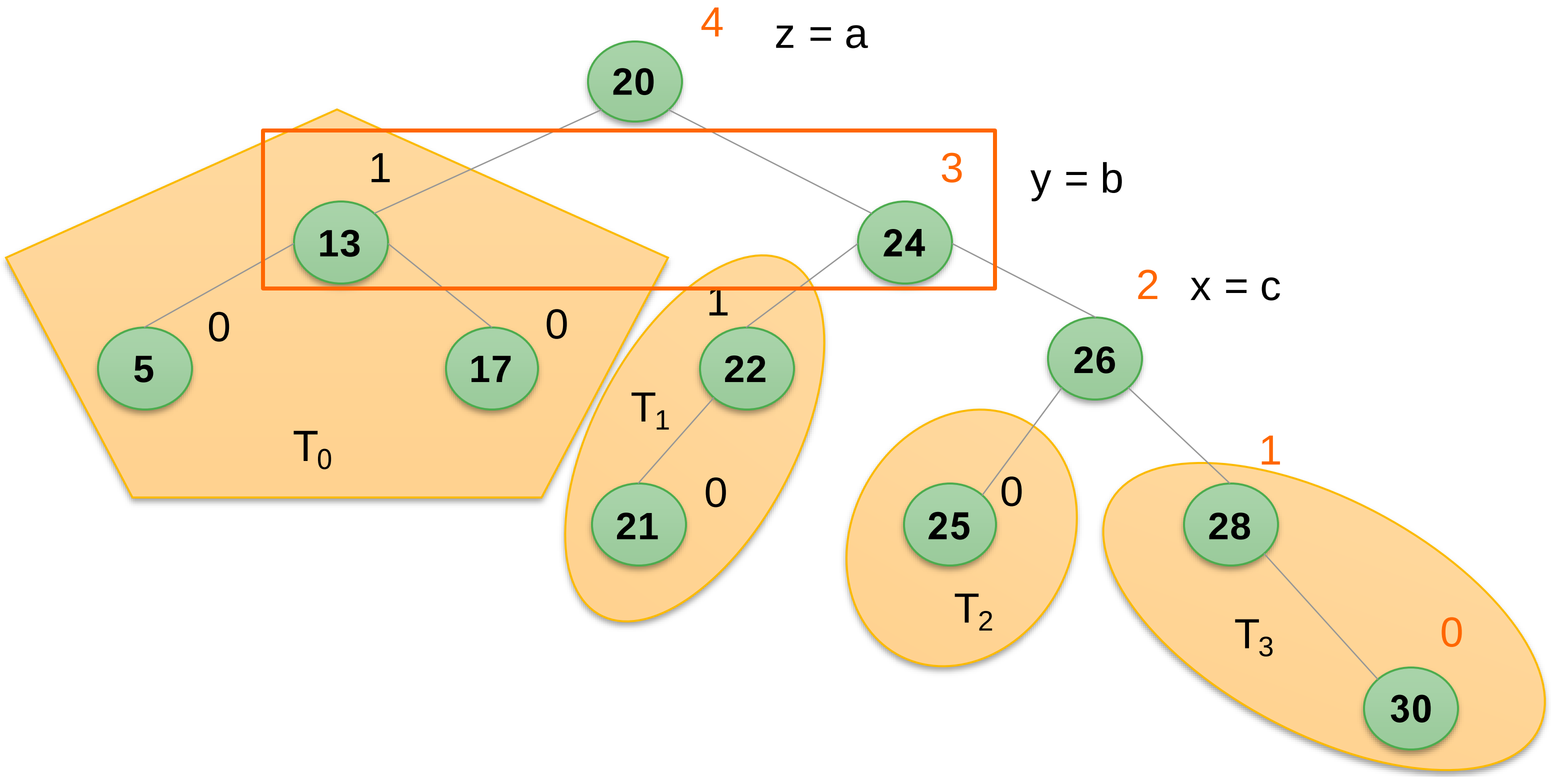
Procedure

1. Number 7 parts according to Inorder traversal

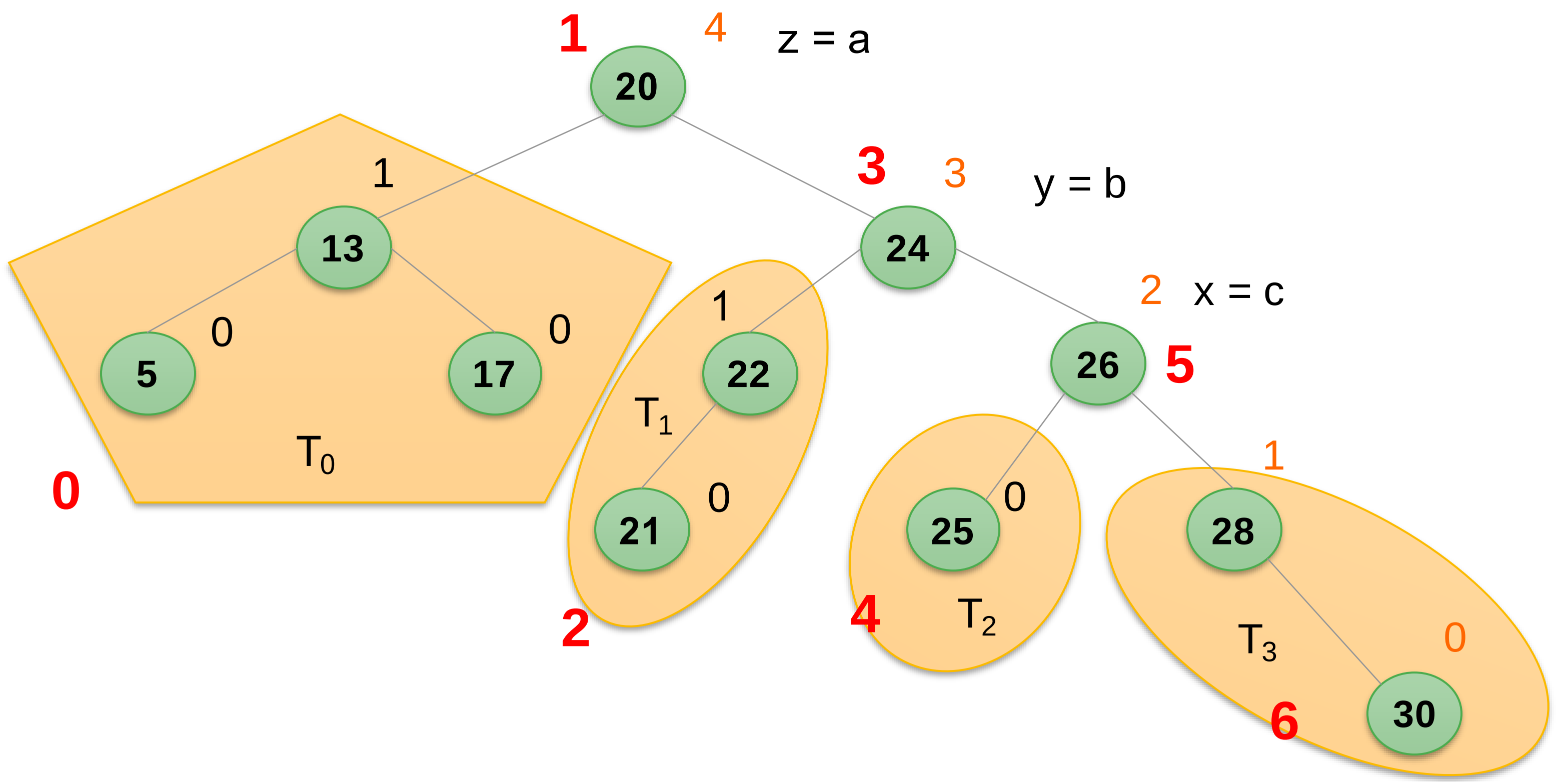
2. Create an array with the indices 0..6, „*cut*“ the 4 subtrees as well as the nodes x, y and z out and put them into the array according to their numbering.

3. (Re)Link the subtrees by setting the element on position 3 as root, those on position 1 and 5 as left and right child of 3, and finally 0, 2, 4 and 6 as children of 1 and 5.

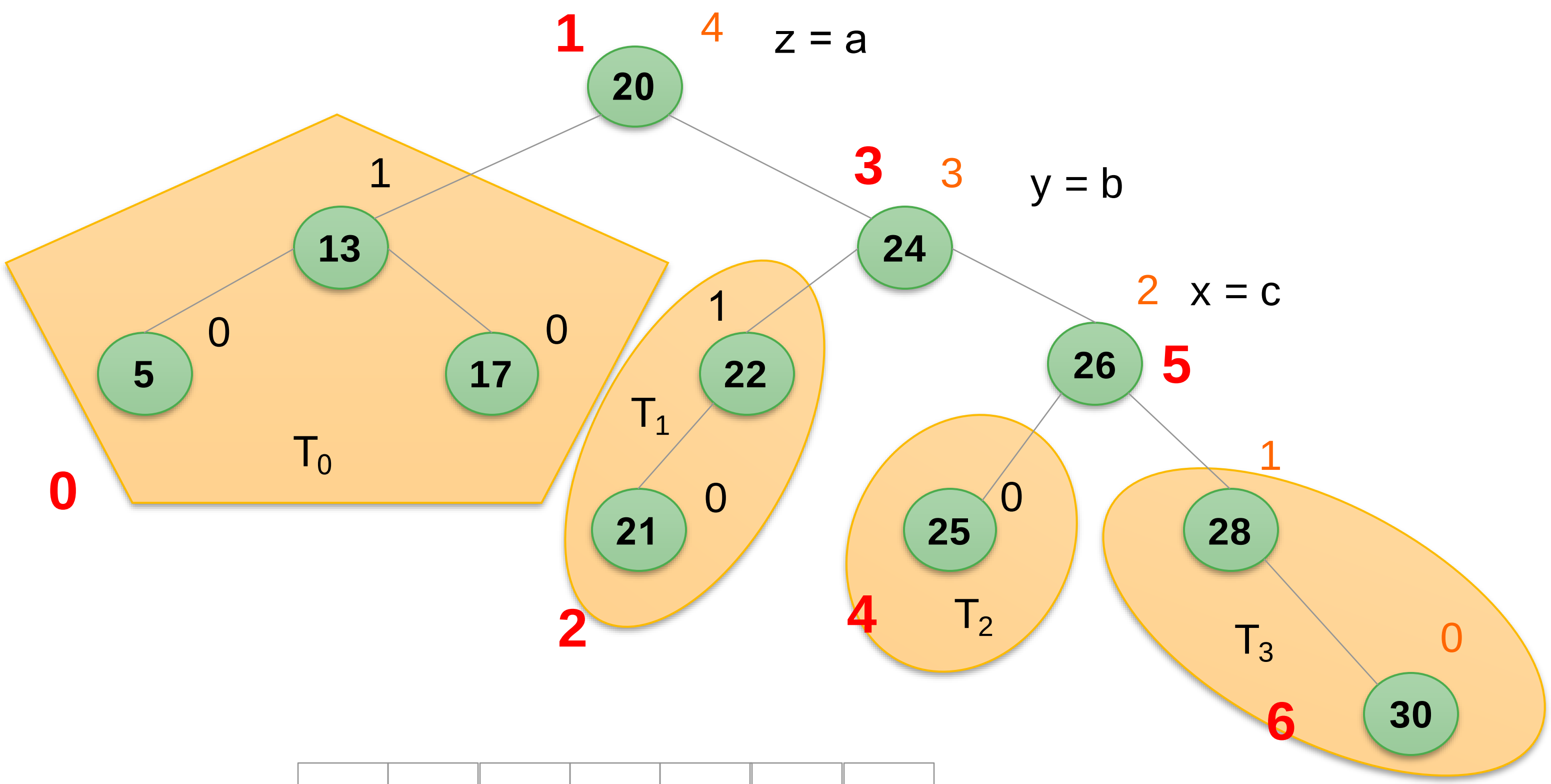
AVL TREE :: SINGLE ROTATION EXAMPLE



AVL TREE :: SINGLE ROTATION EXAMPLE

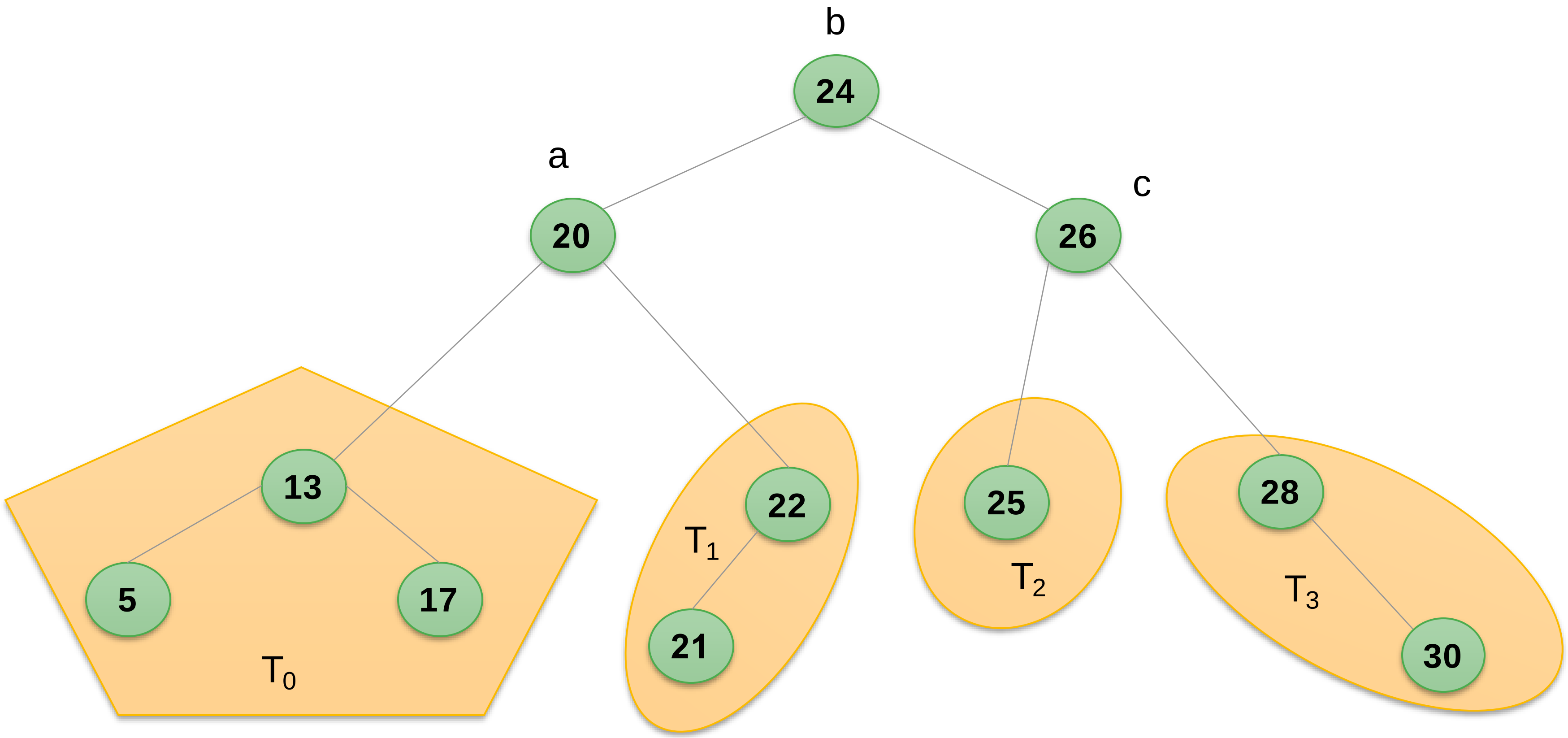


AVL TREE :: SINGLE ROTATION EXAMPLE



0	1	2	3	4	5	6
T_0	z=a	T_1	y=b	T_2	x=c	T_3

AVL TREE :: SINGLE ROTATION EXAMPLE



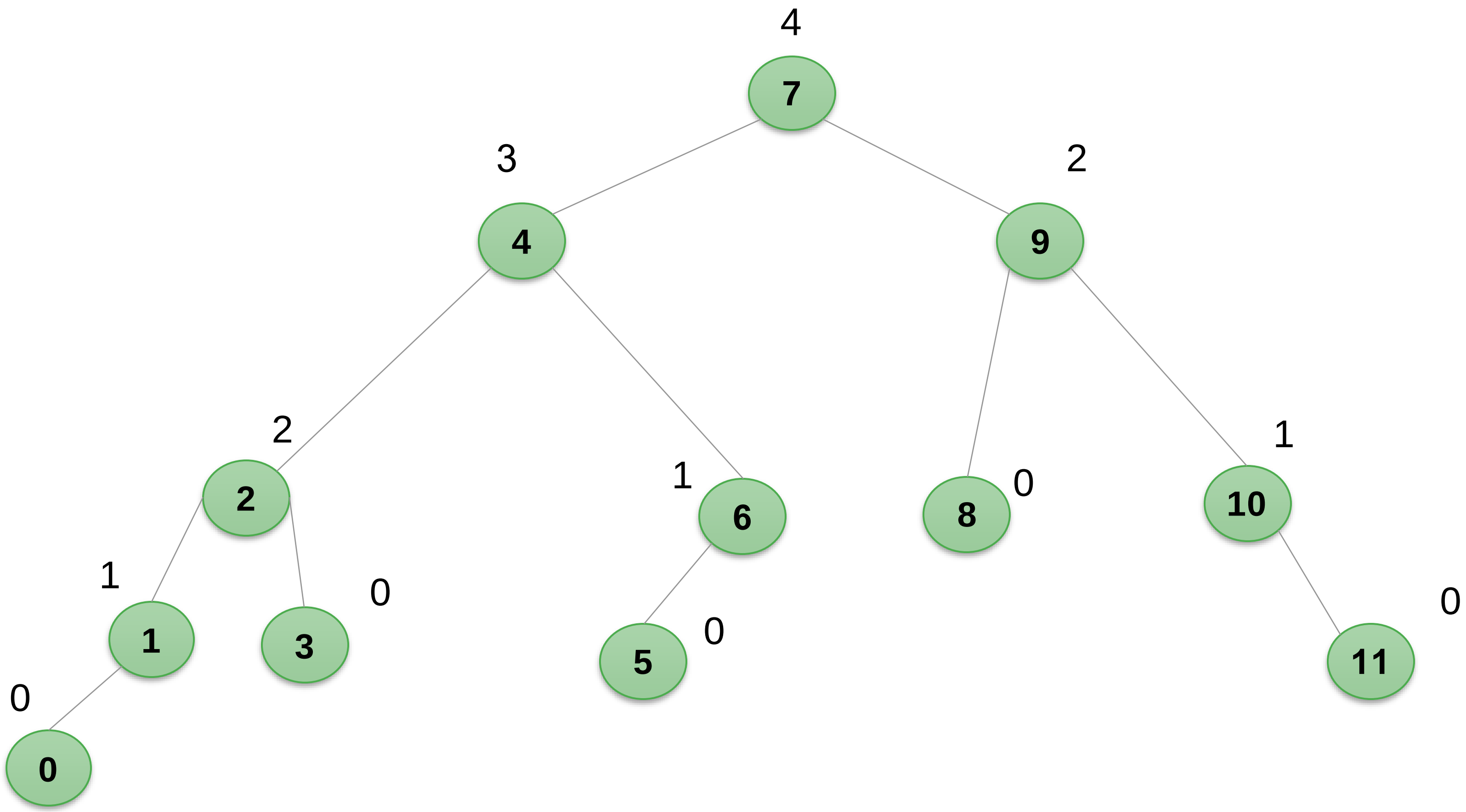
AVL TREES :: REMOVE

- Remove as in binary search tree
- Check the balance starting from the parent node of the removed **Inorder** successor to the root.
- **Restructure** if necessary

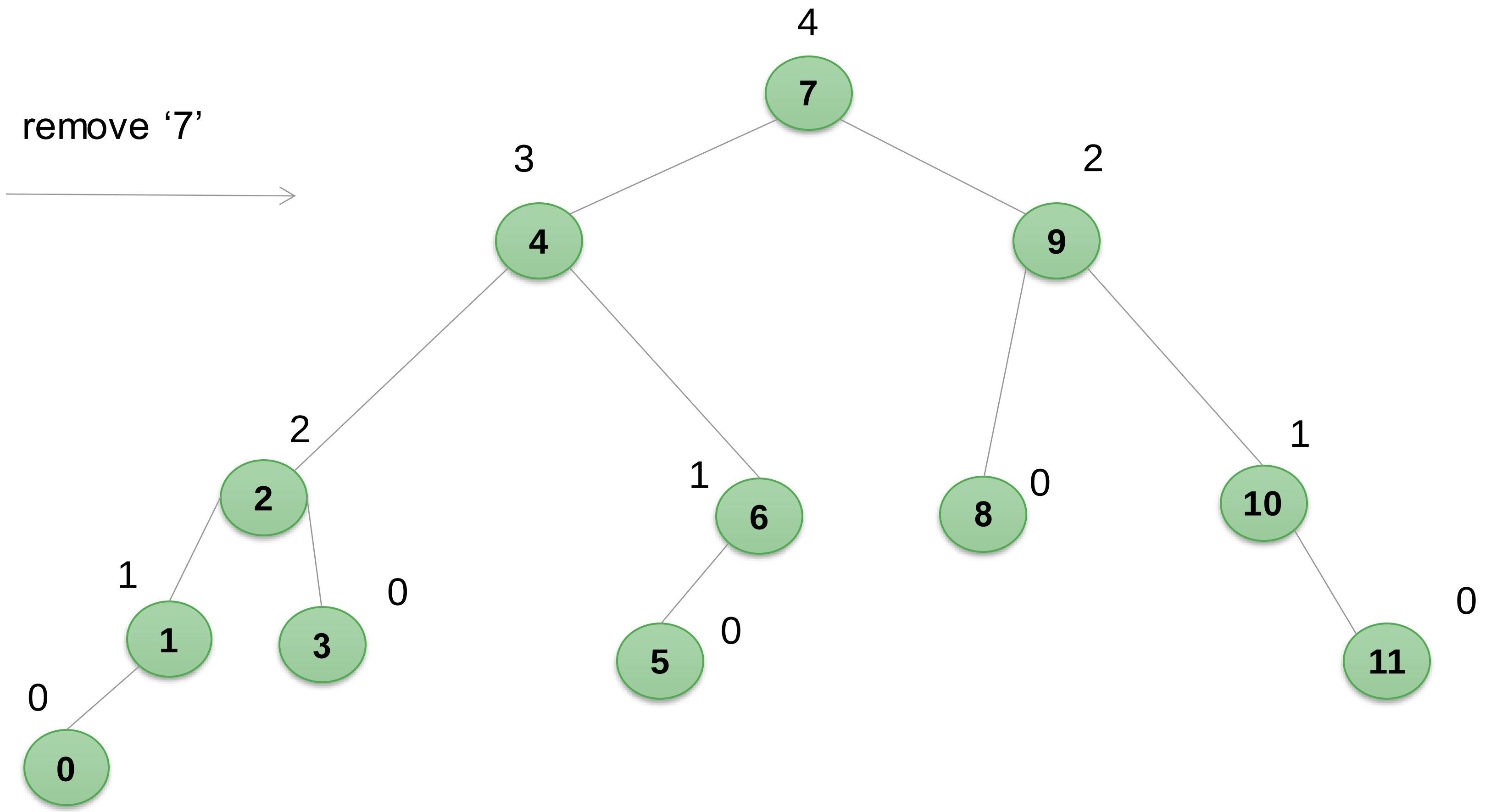
Procedure

1. Search for the 1. unbalanced node z
2. Put y on child of z with greatest height
3. Put x on child of y with greatest height

AVL TREES :: REMOVE

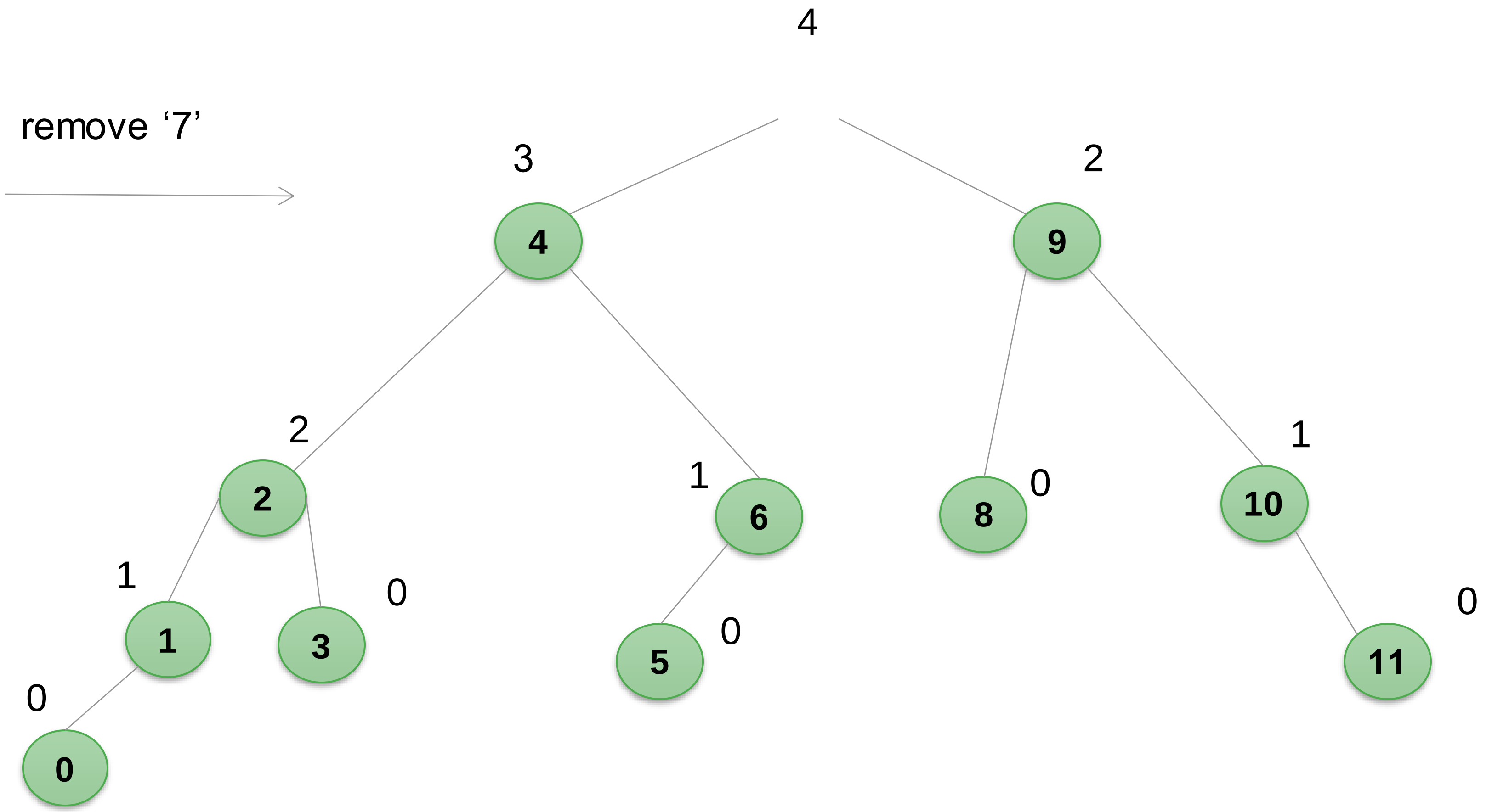


AVL TREES :: REMOVE



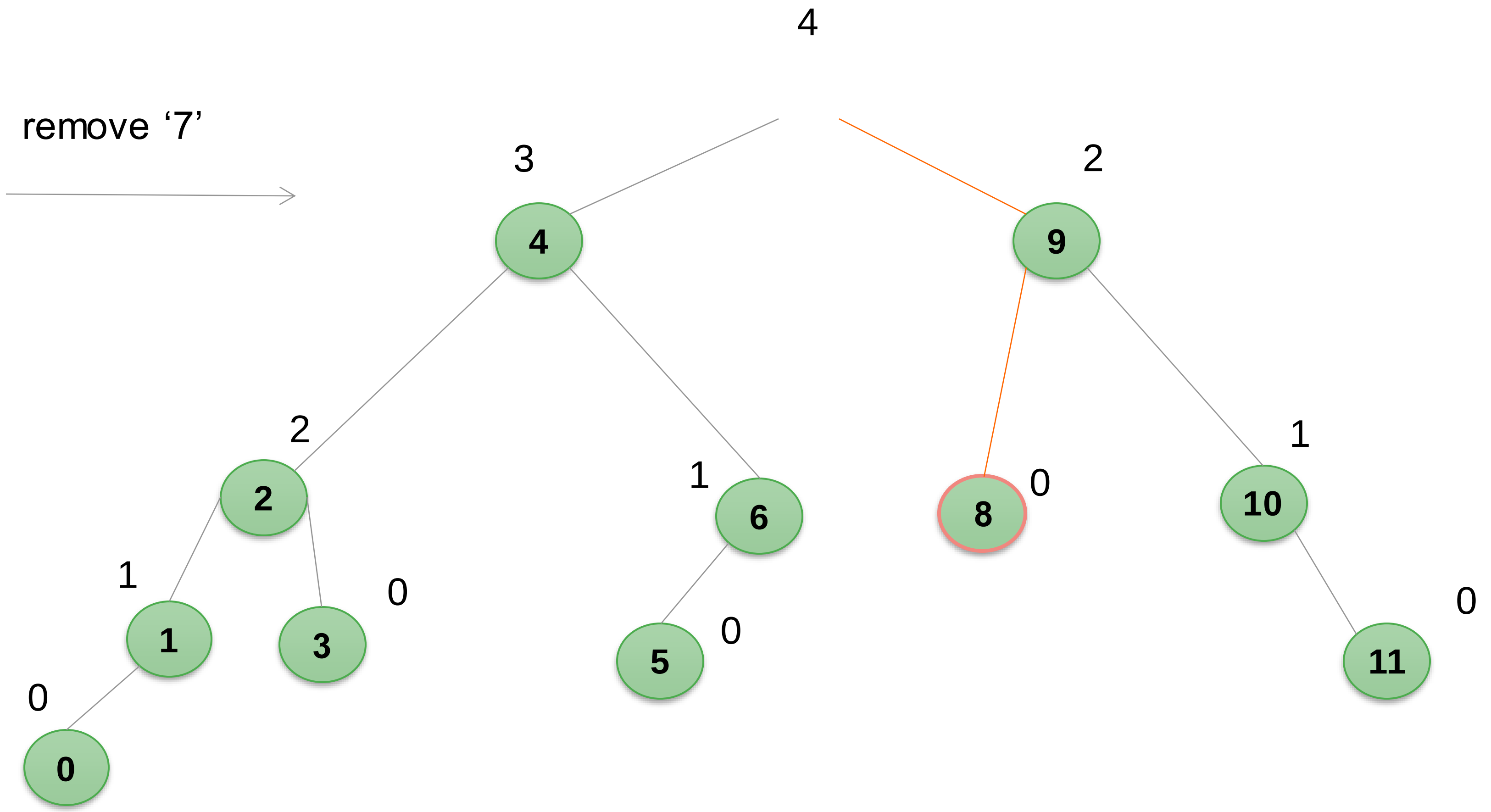
AVL TREES :: REMOVE

remove ‘7’



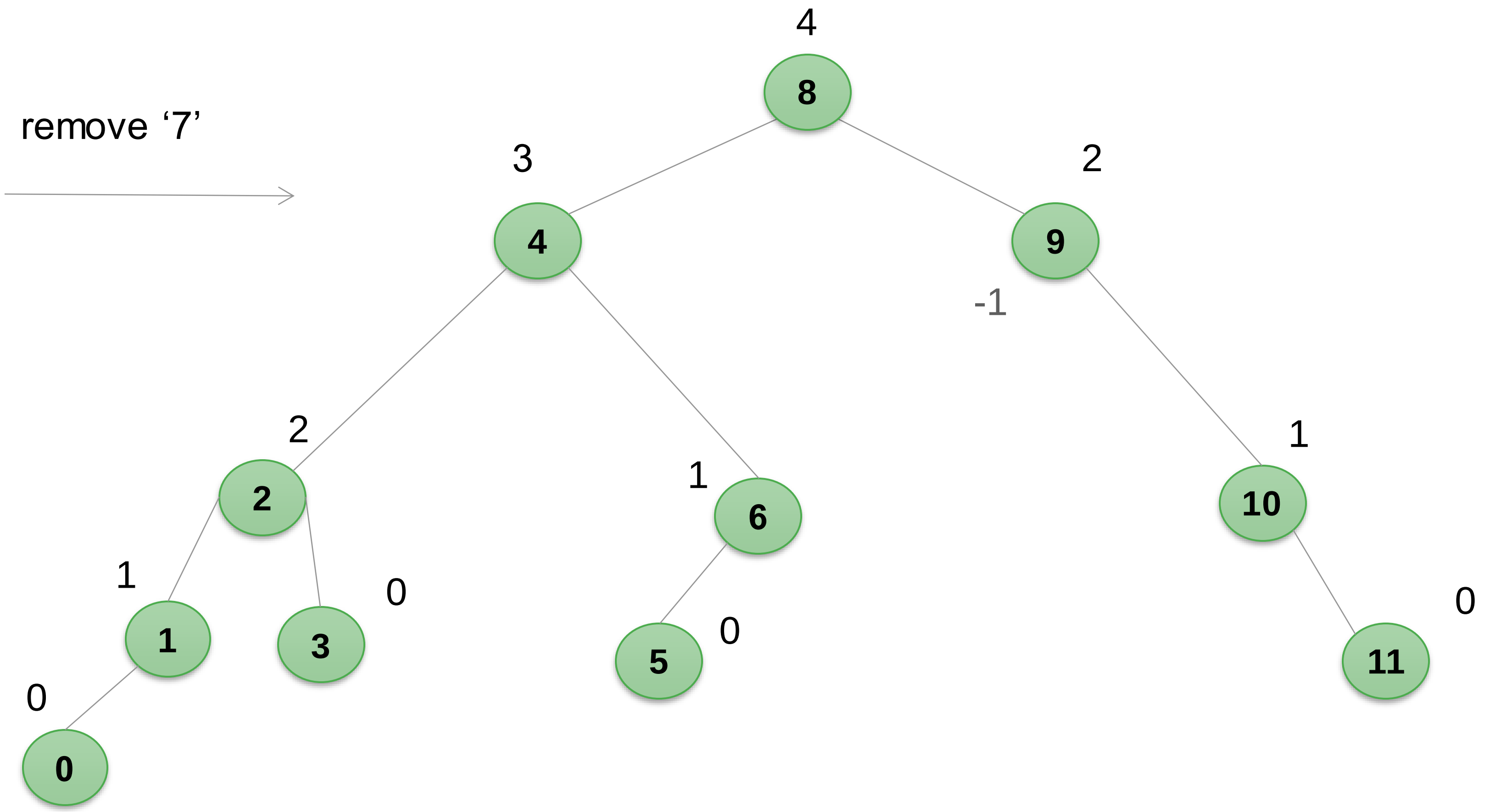
AVL TREES :: REMOVE

remove '7'

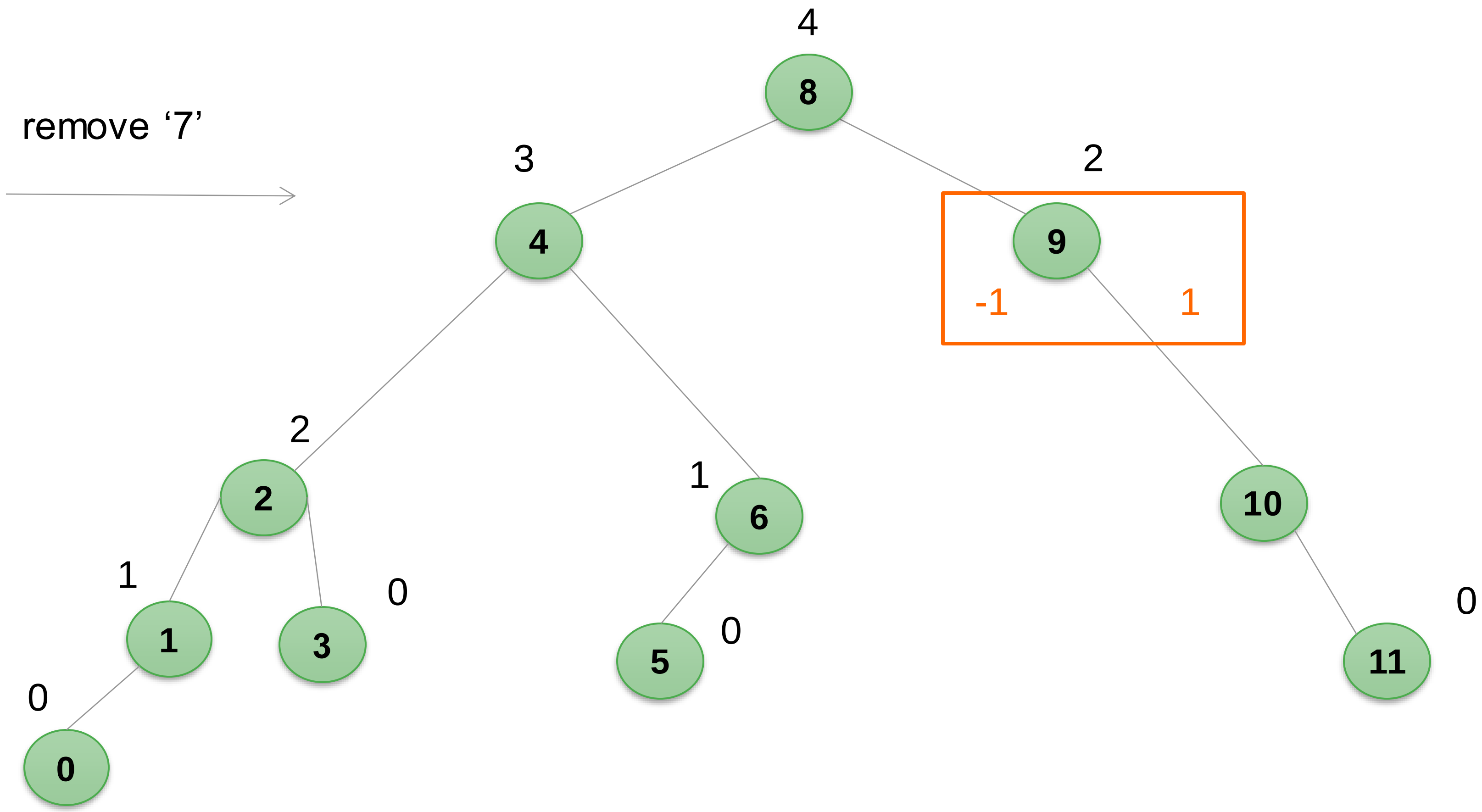


AVL TREES :: REMOVE

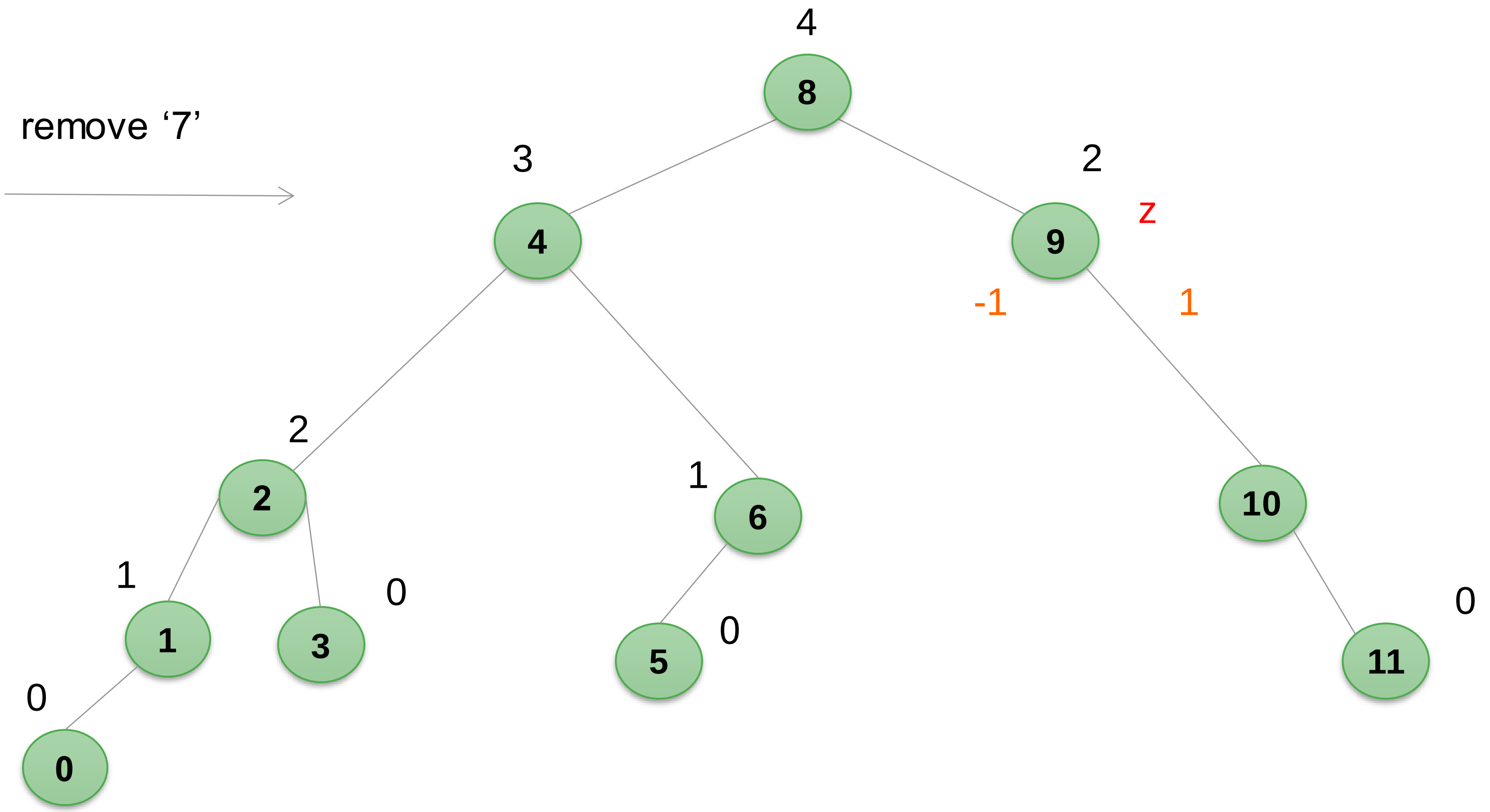
remove ‘7’



AVL TREES :: REMOVE

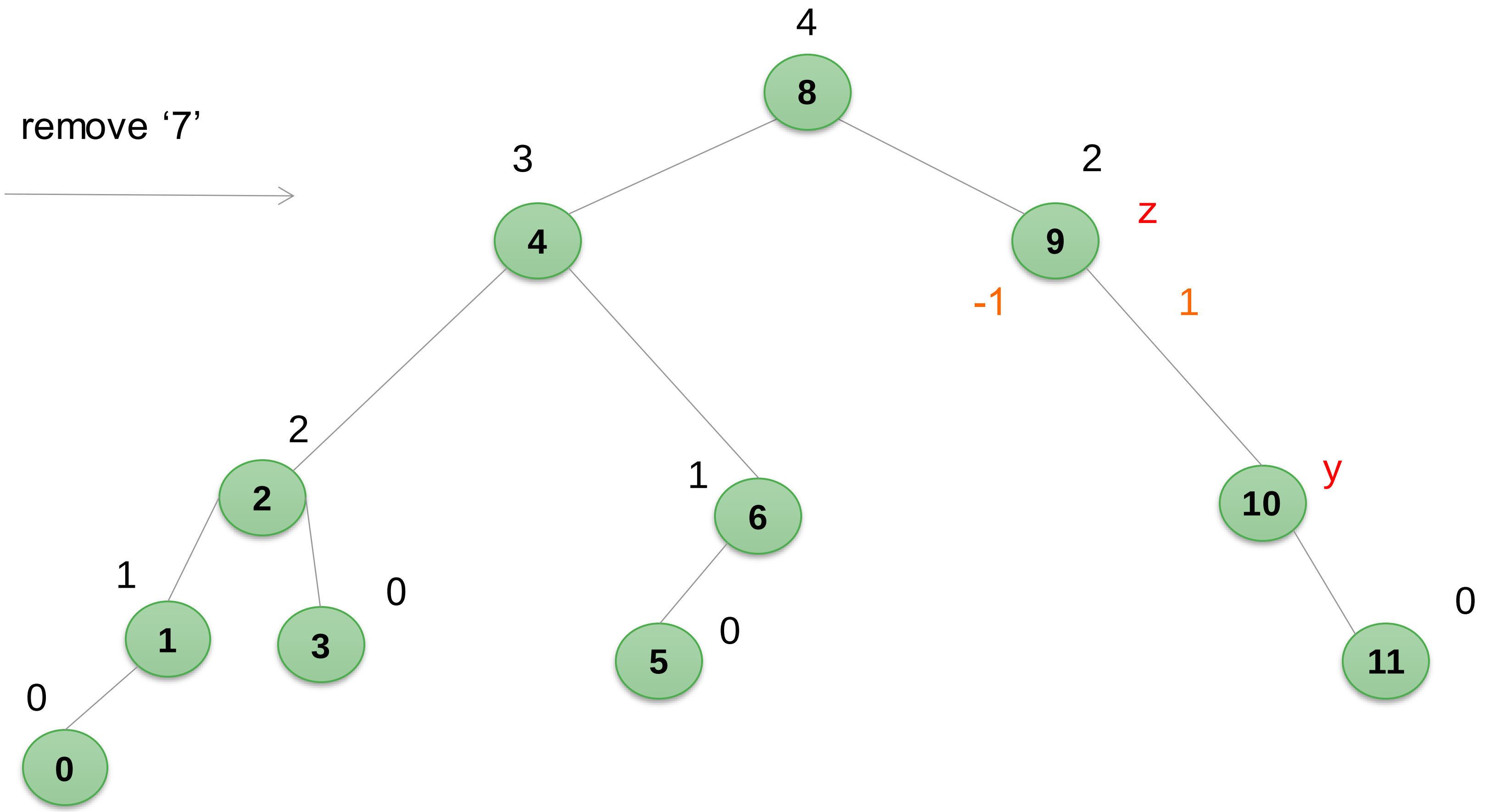


AVL TREES :: REMOVE



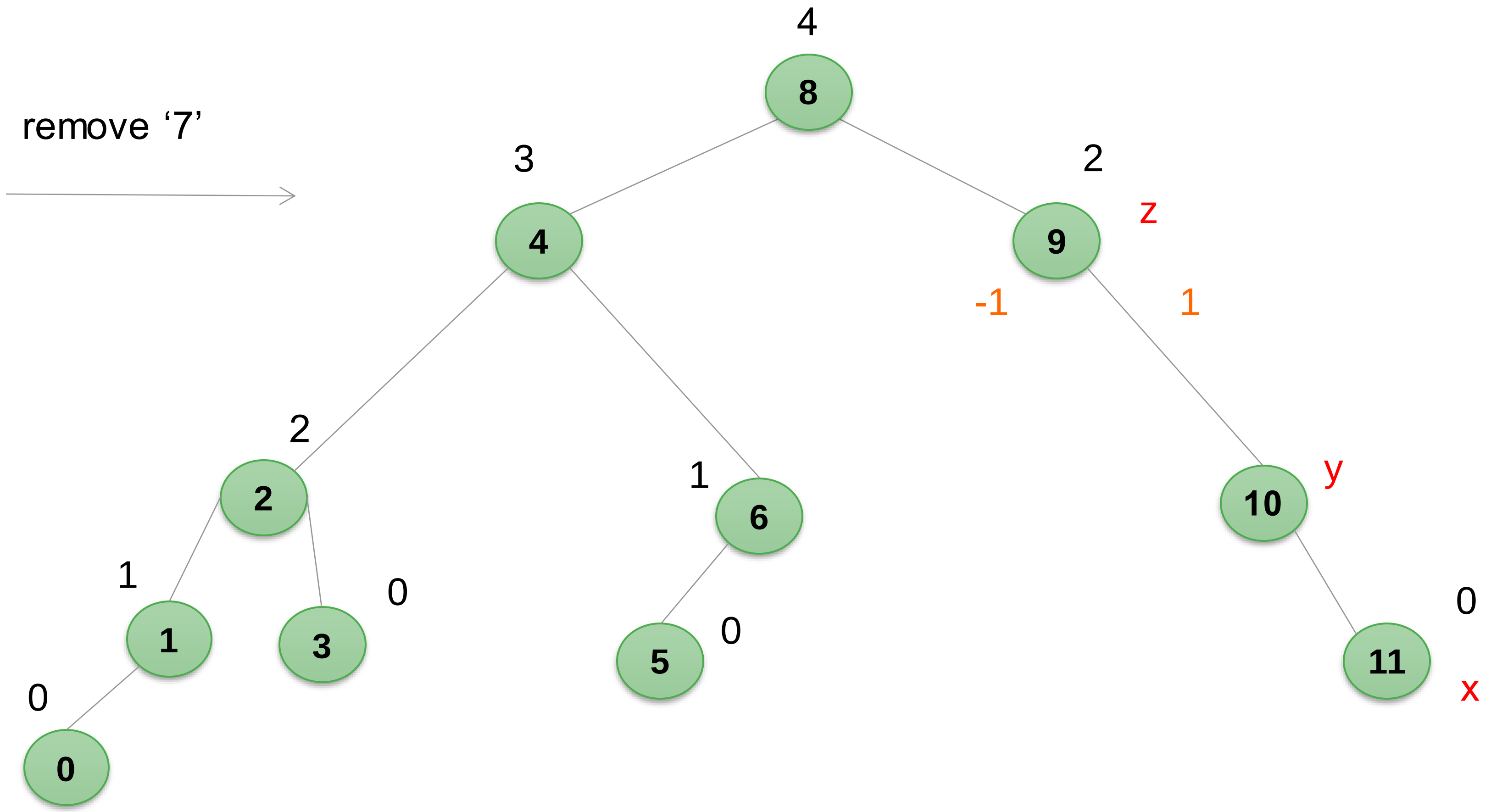
AVL TREES :: REMOVE

remove ‘7’

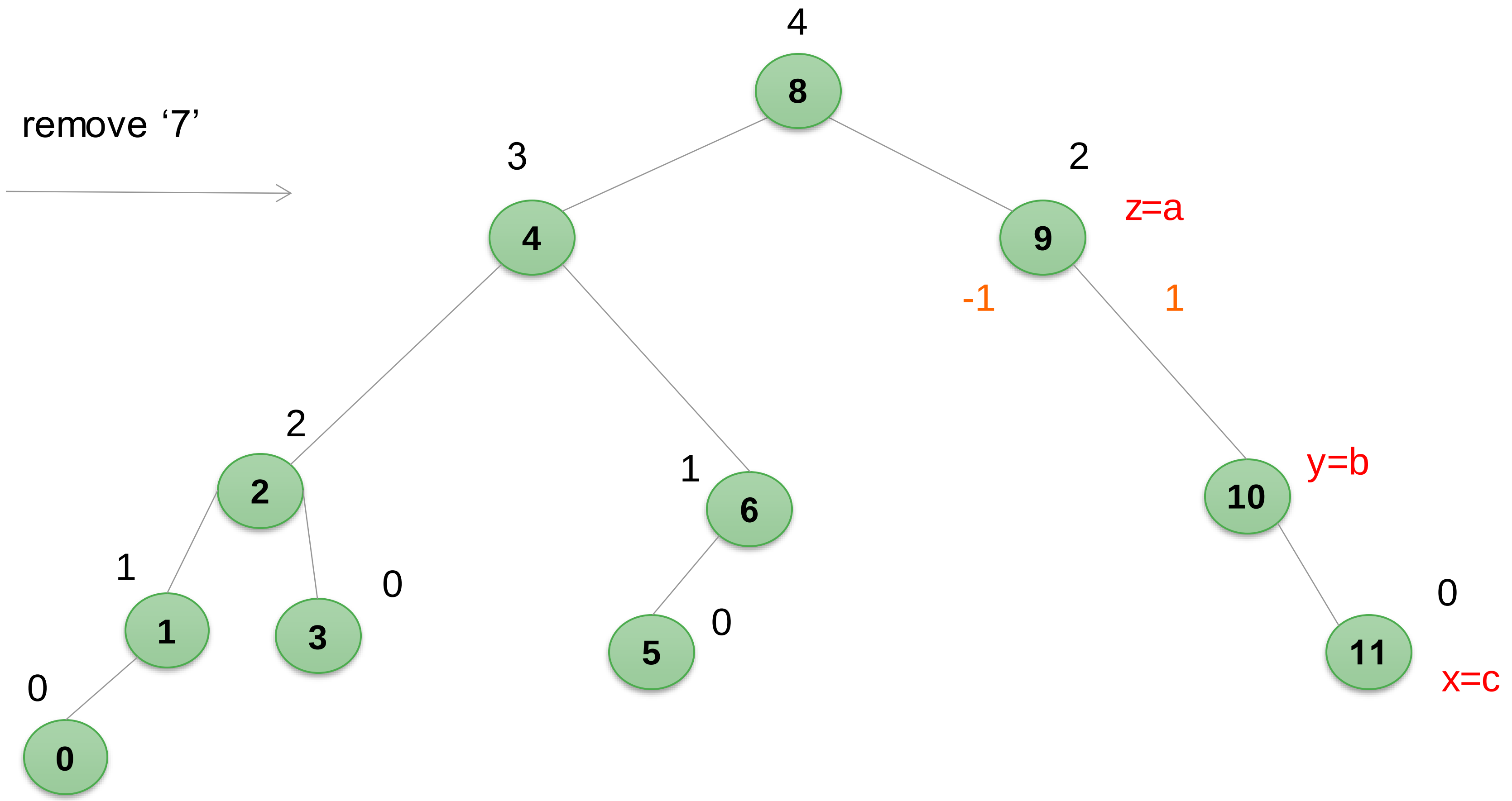


AVL TREES :: REMOVE

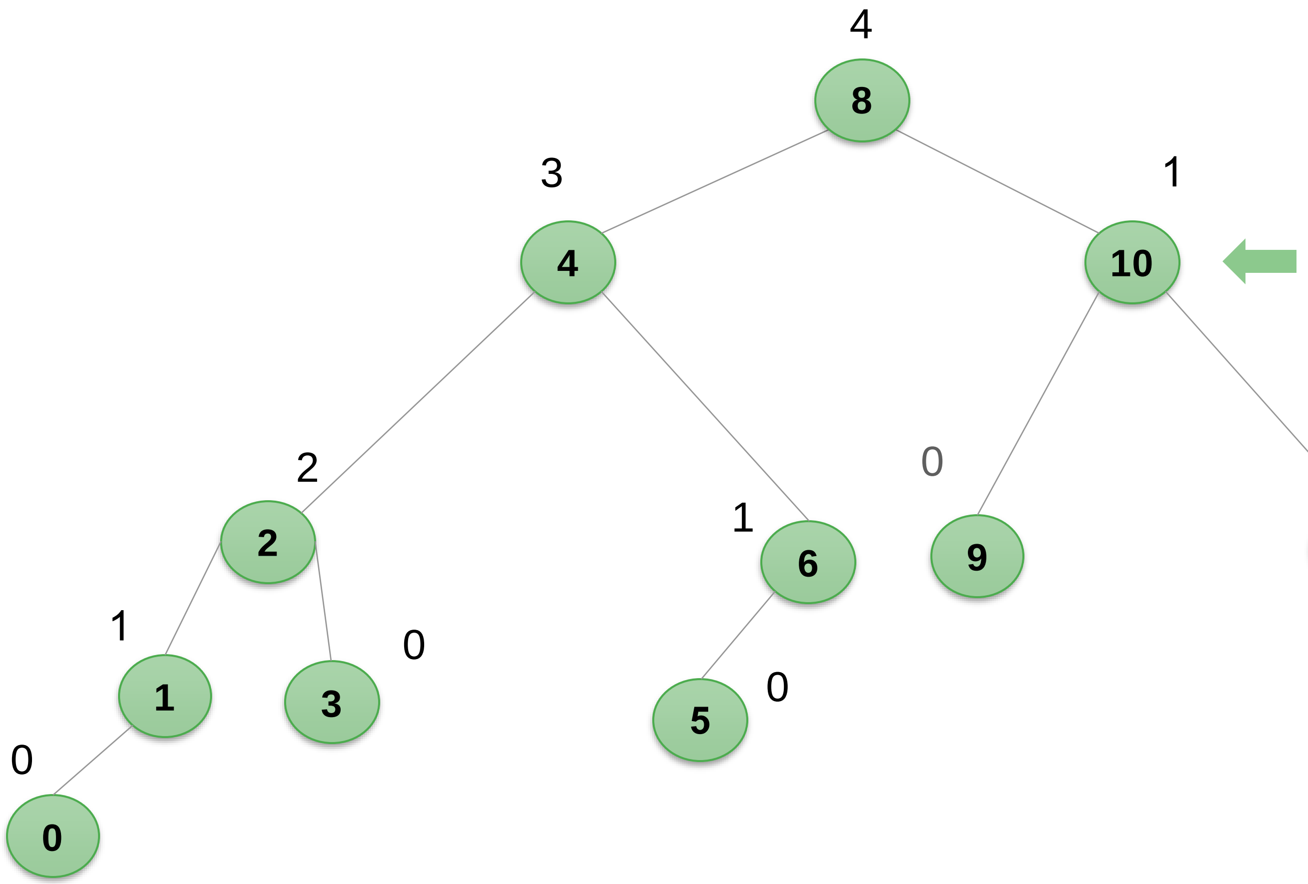
remove '7'



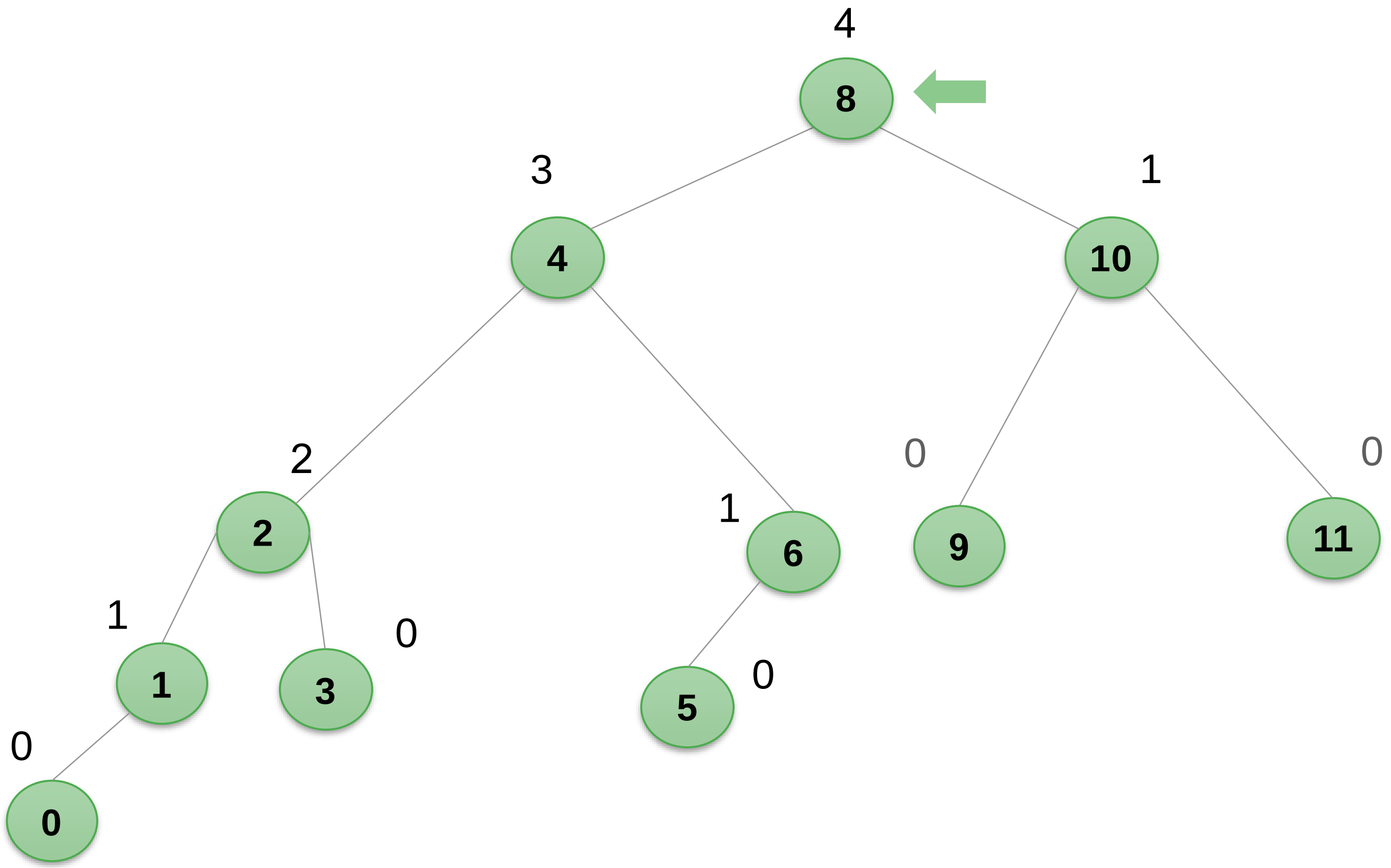
AVL TREES :: REMOVE



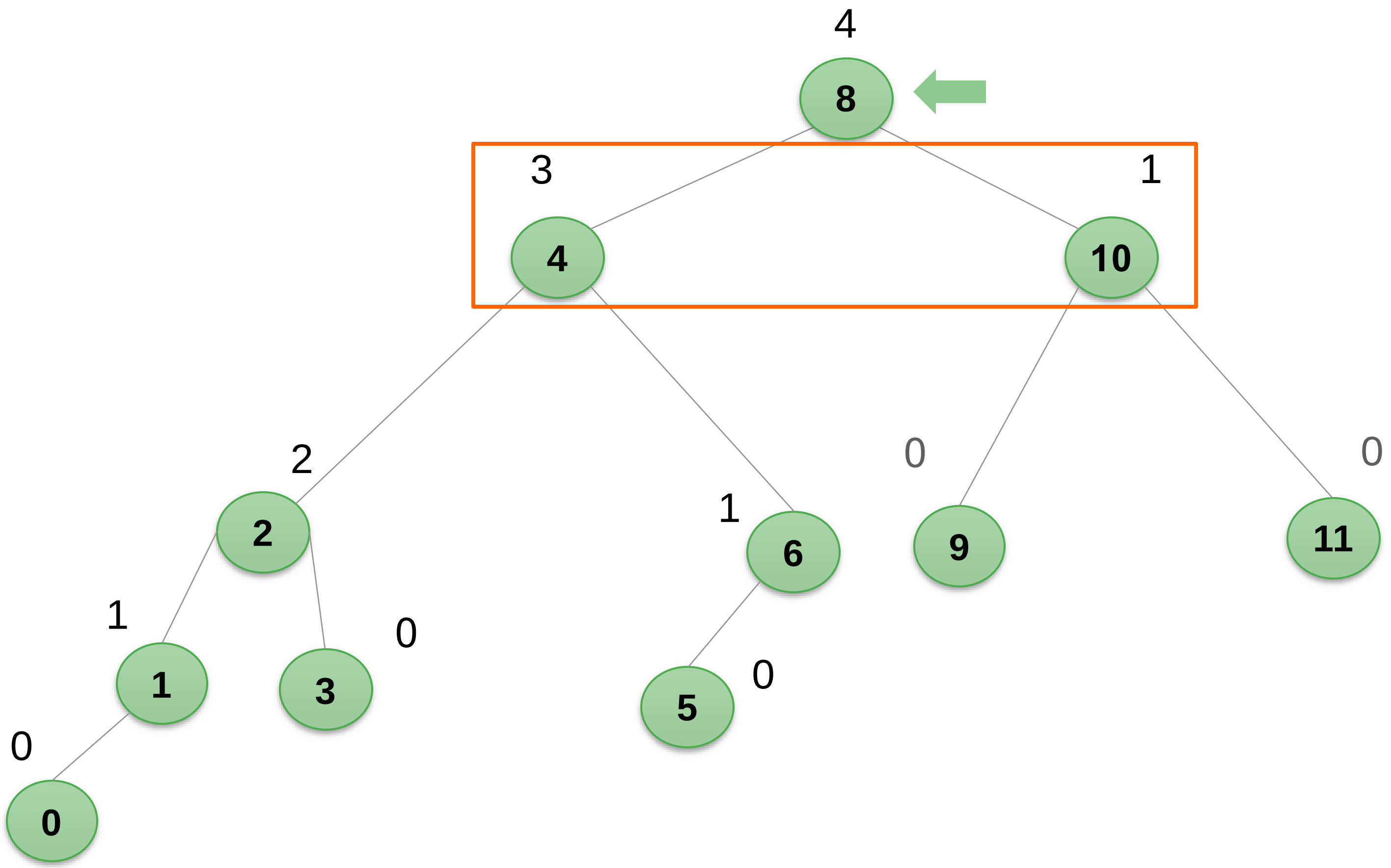
AVL TREES :: REMOVE



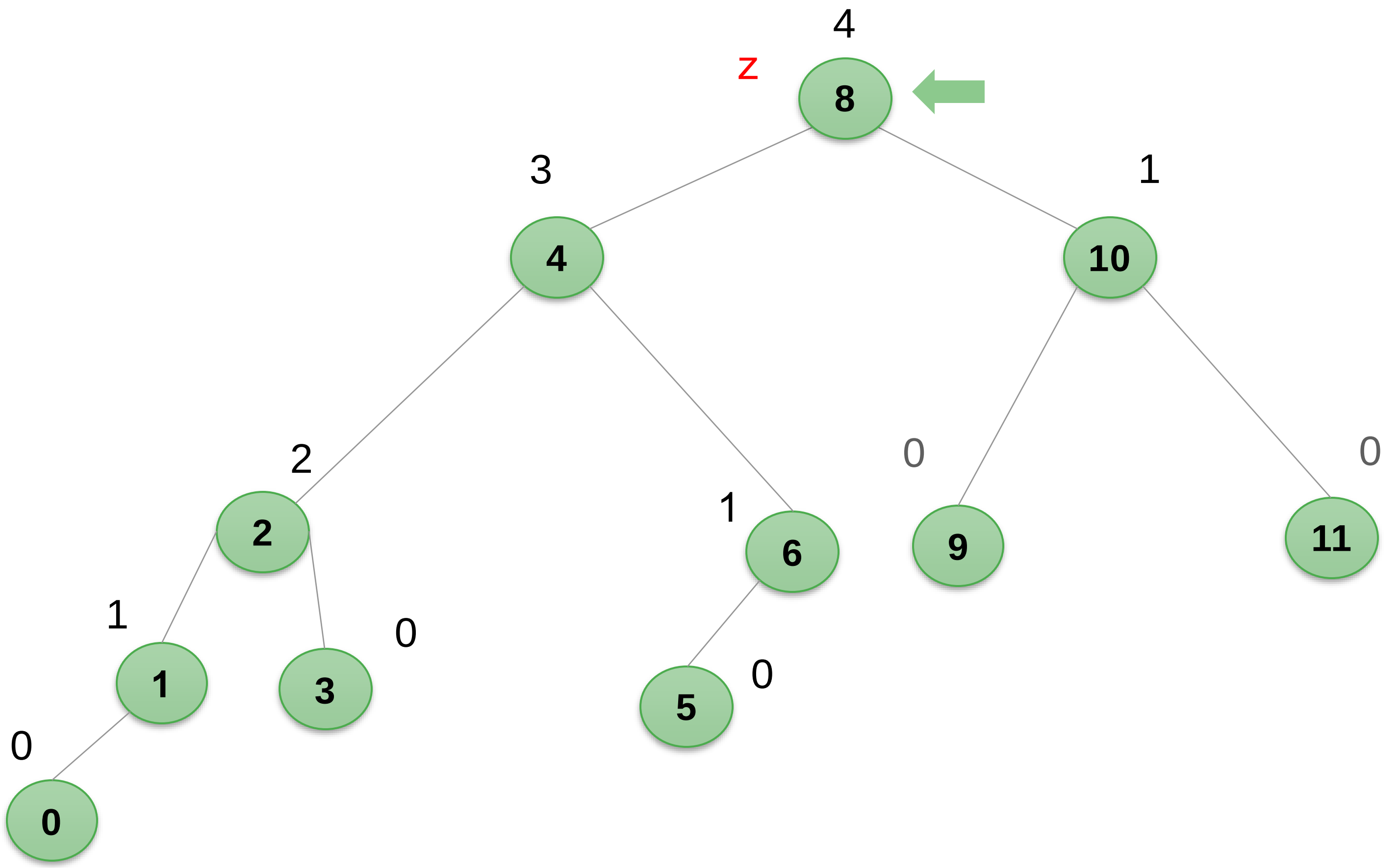
AVL TREES :: REMOVE



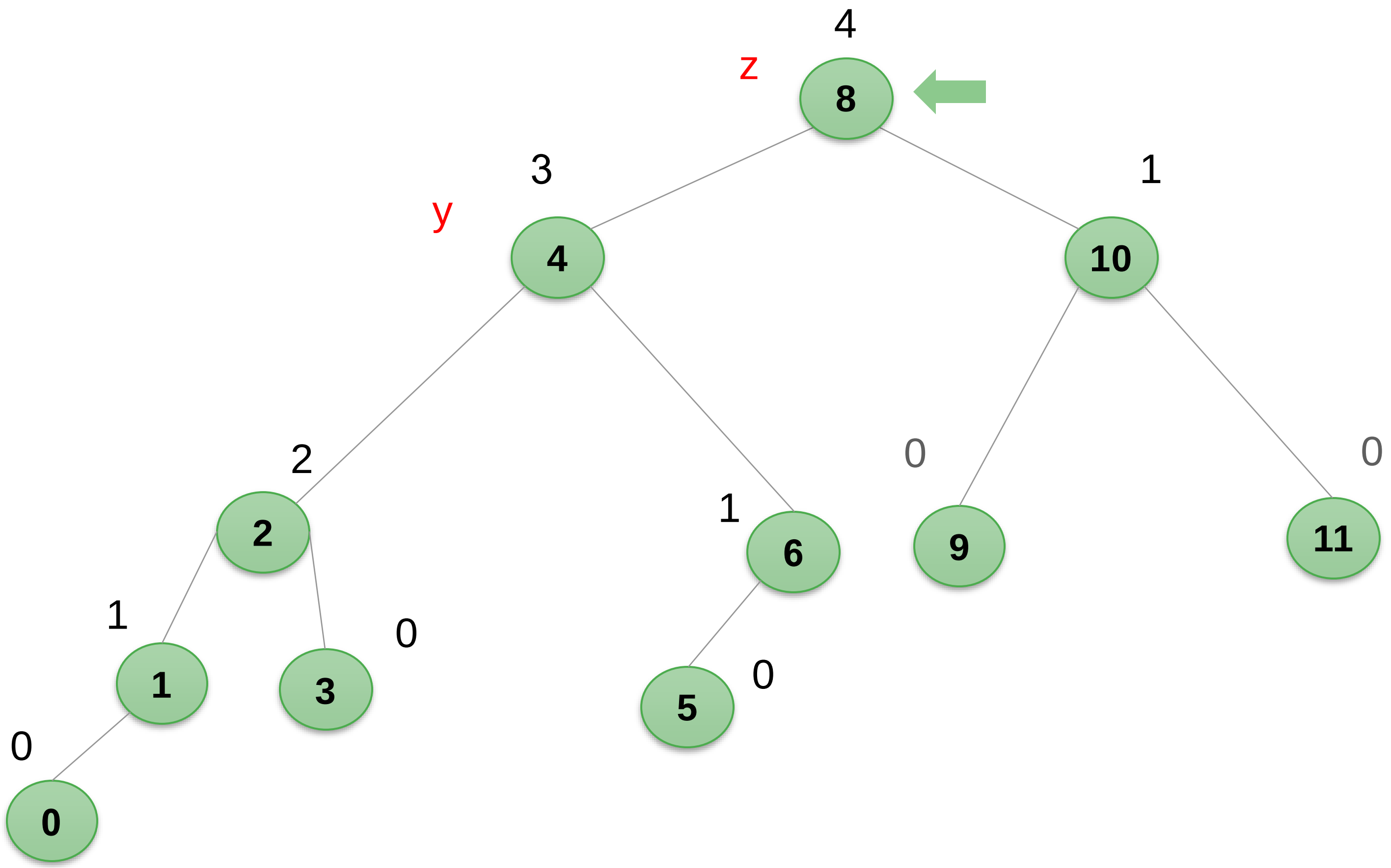
AVL TREES :: REMOVE



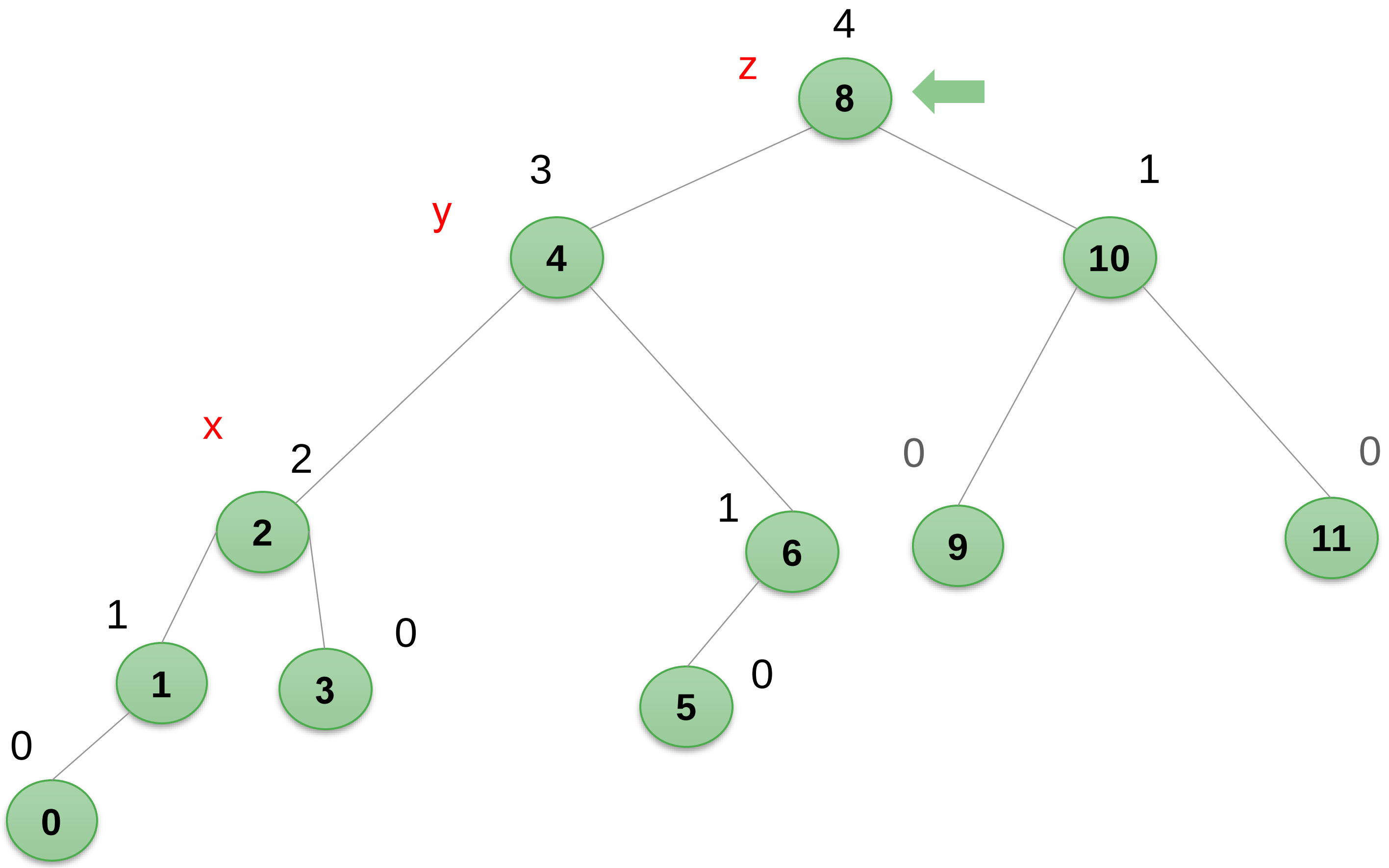
AVL TREES :: REMOVE



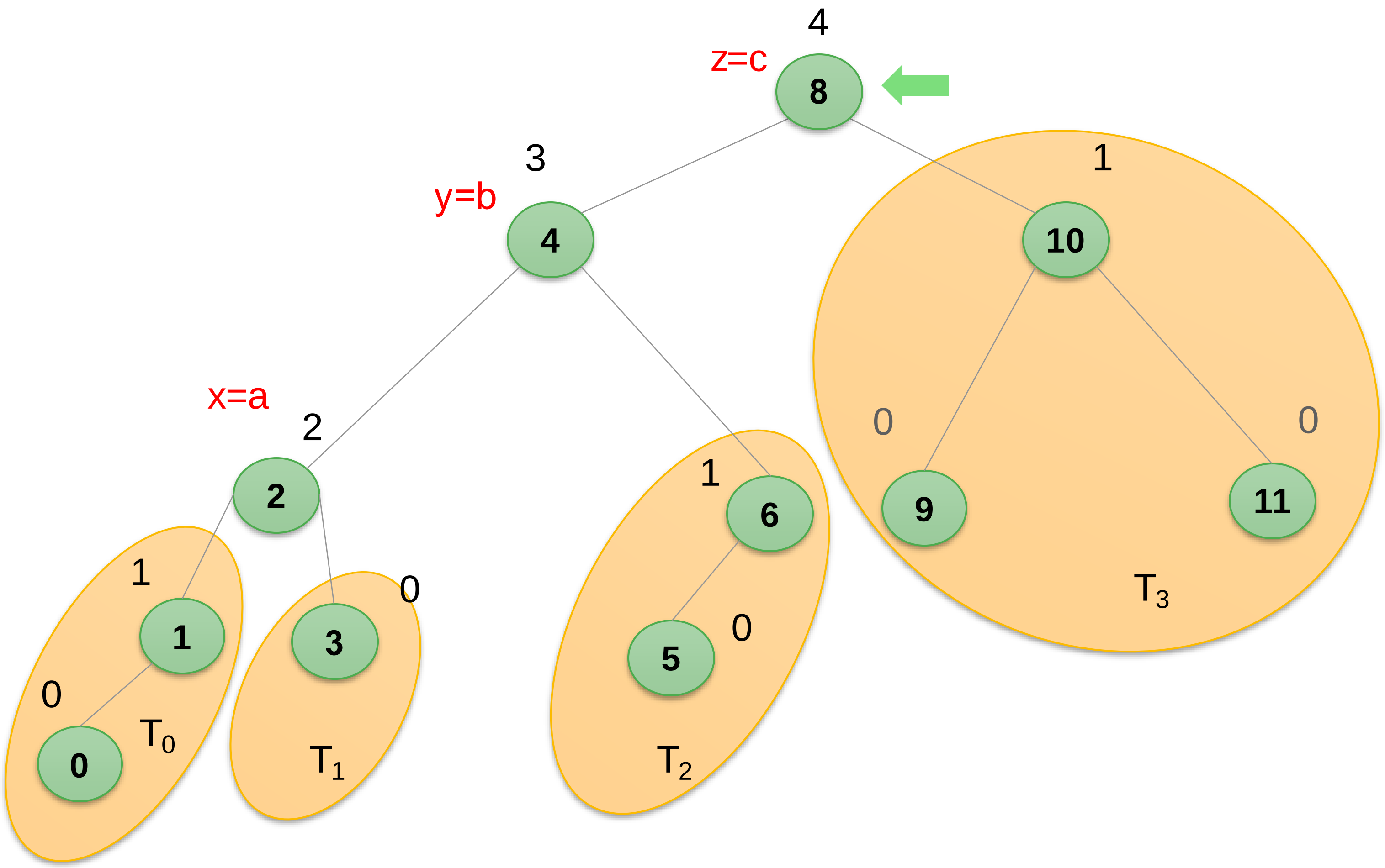
AVL TREES :: REMOVE



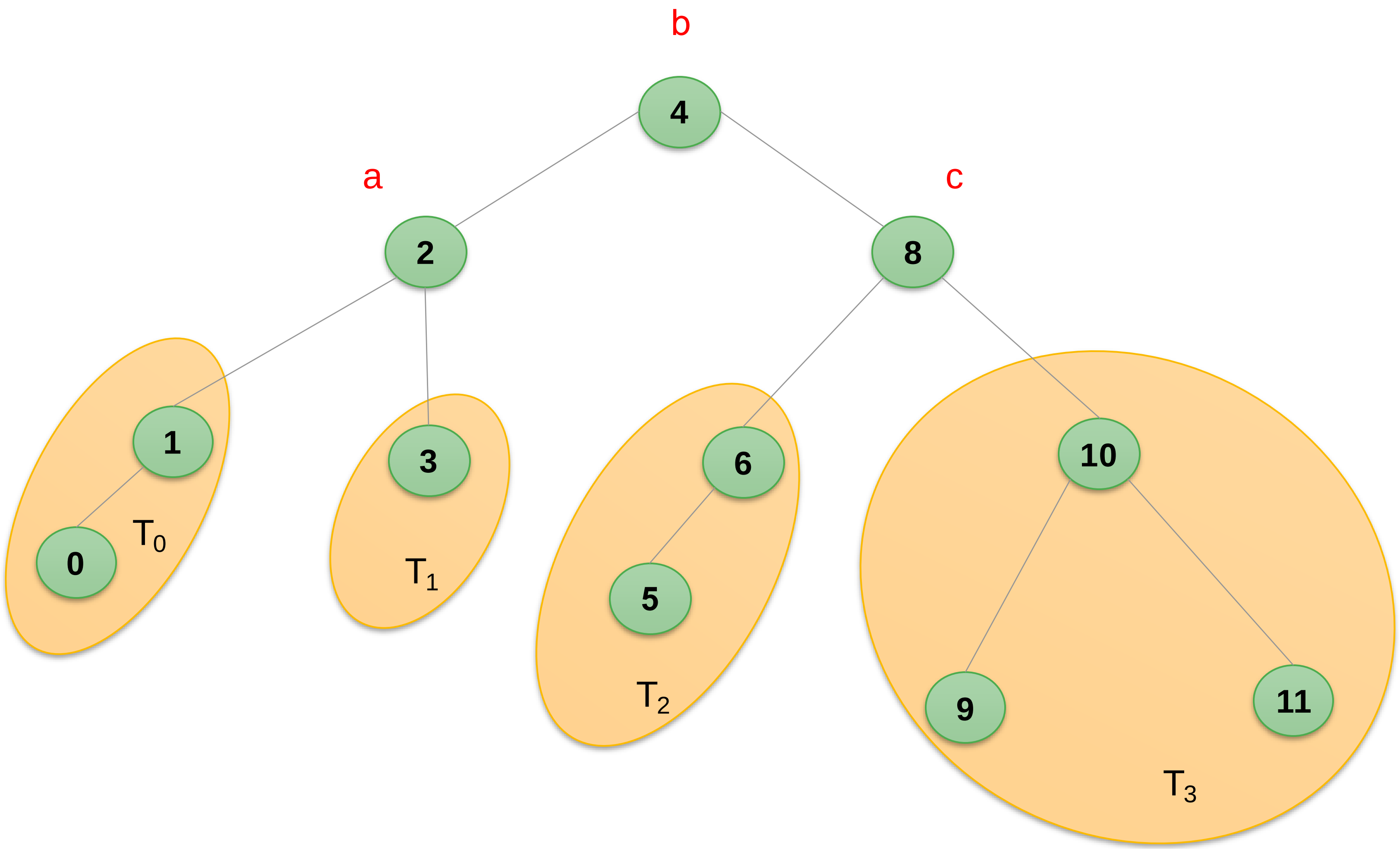
AVL TREES :: REMOVE



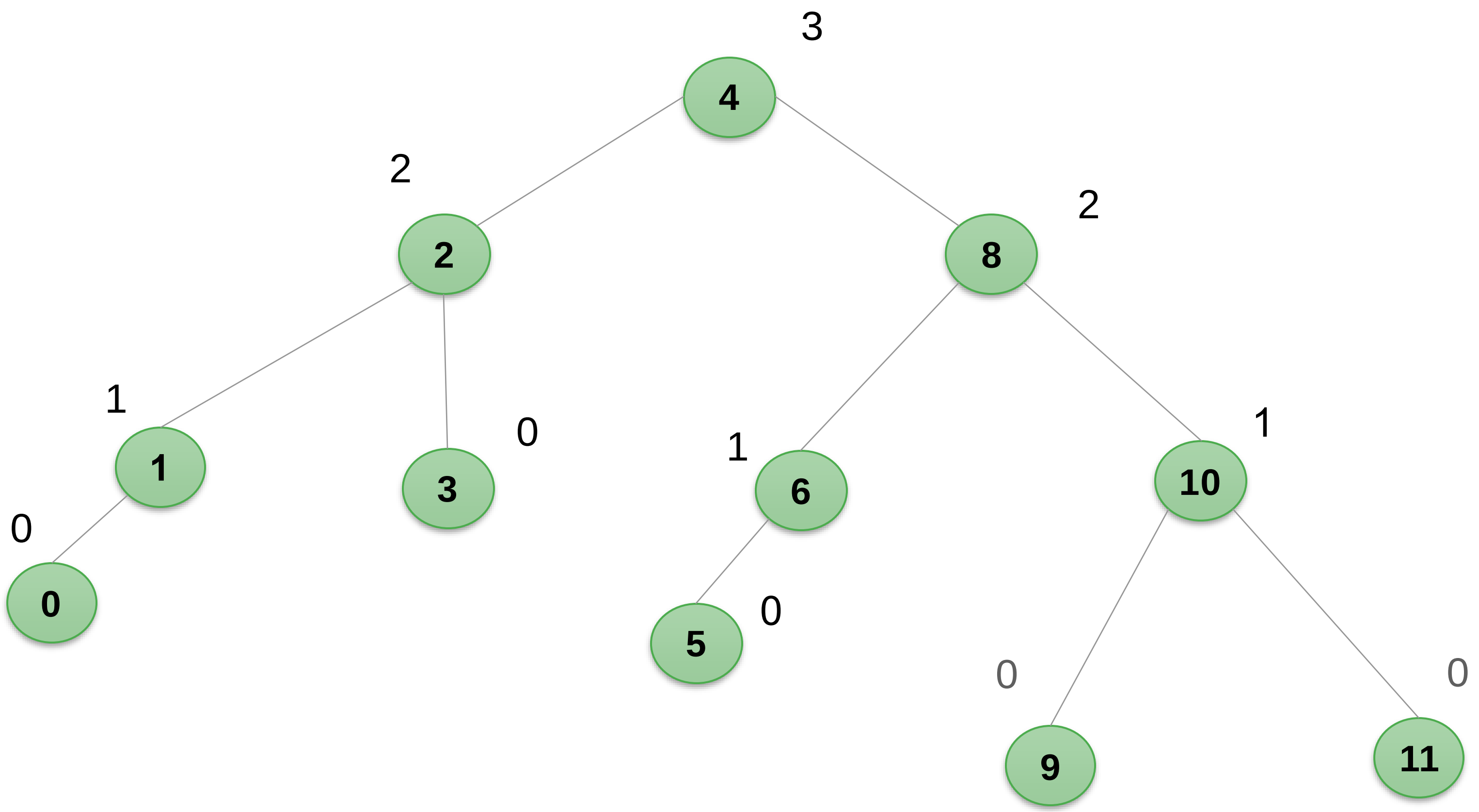
AVL TREES :: REMOVE



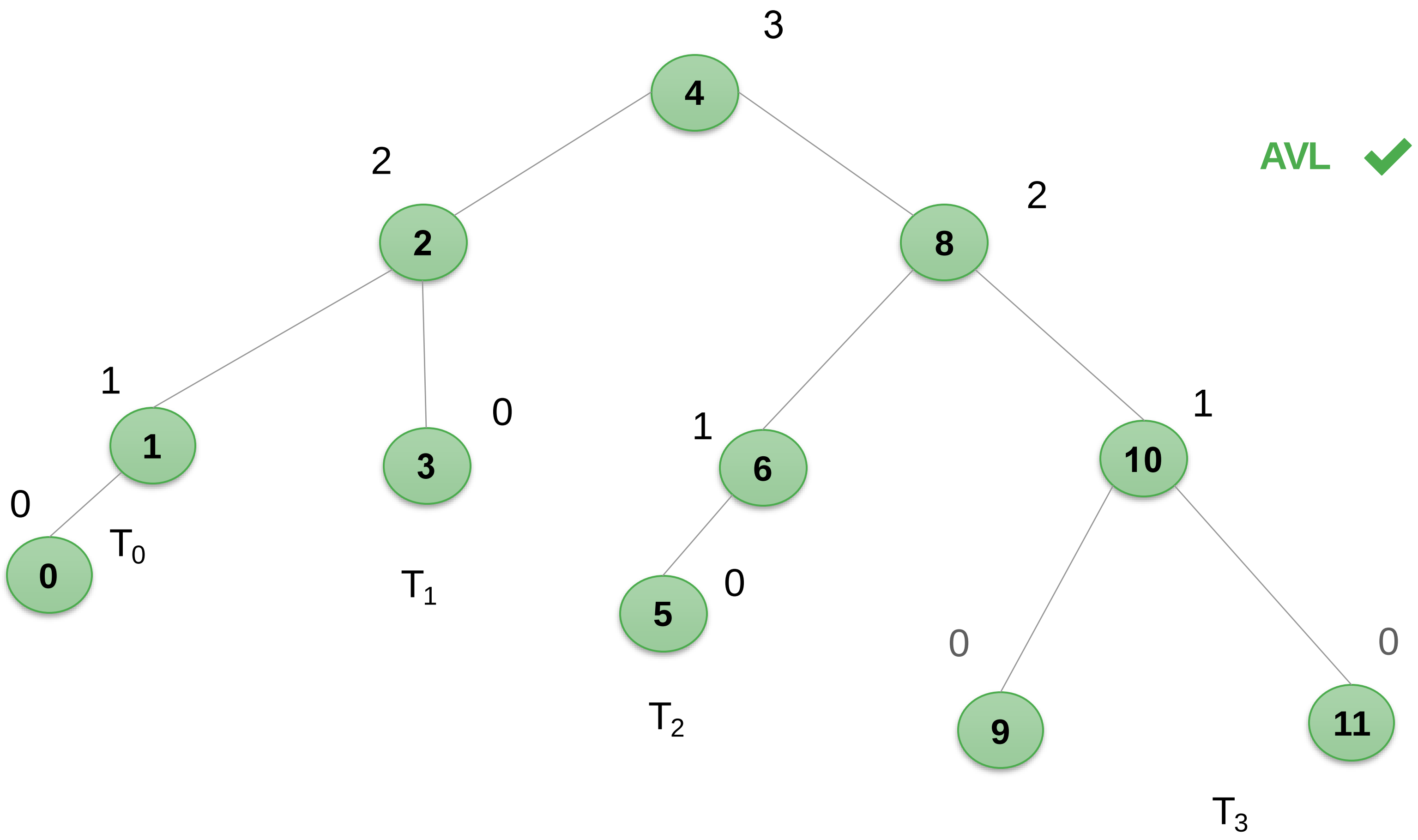
AVL TREES :: REMOVE



AVL TREES :: REMOVE



AVL TREES :: REMOVE



ASSIGNMENT 02

FAST SEARCHING / BALANCED TREES

Algorithms and Data Structures 2
Exercise – 2021W