JOHANNES KEPLER
UNIVERSITY LINZ

FAST SEARCHING / BALANCED TREES

Martin Schobesberger, Markus Weninger,
Markus Jäger, Florian Beck, Achref Rihani

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

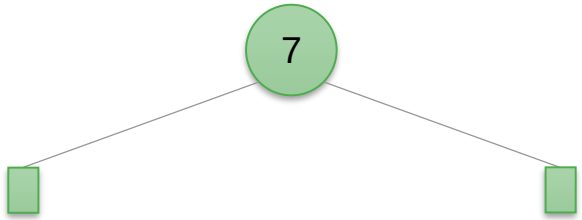
Algorithms and Data Structures 2
Exercise – 2023W

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

JOHANNES KEPLER
UNIVERSITY LINZ

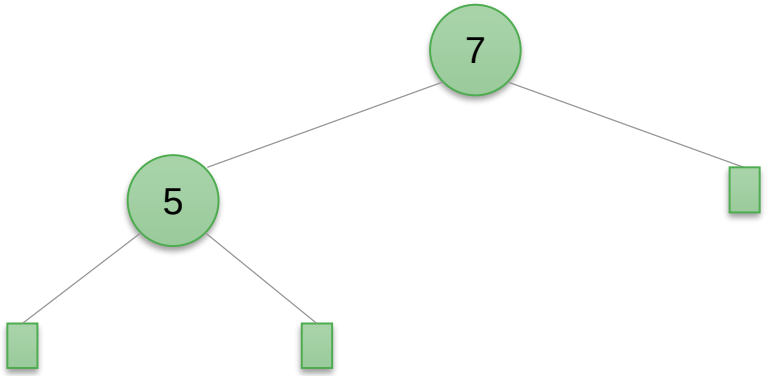
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



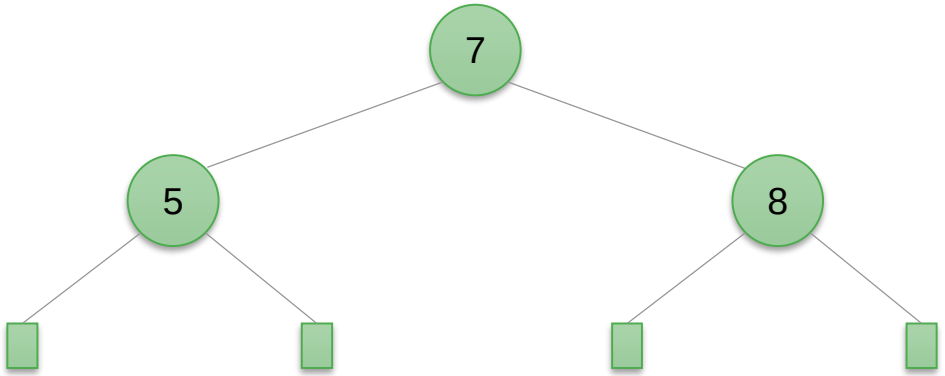
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



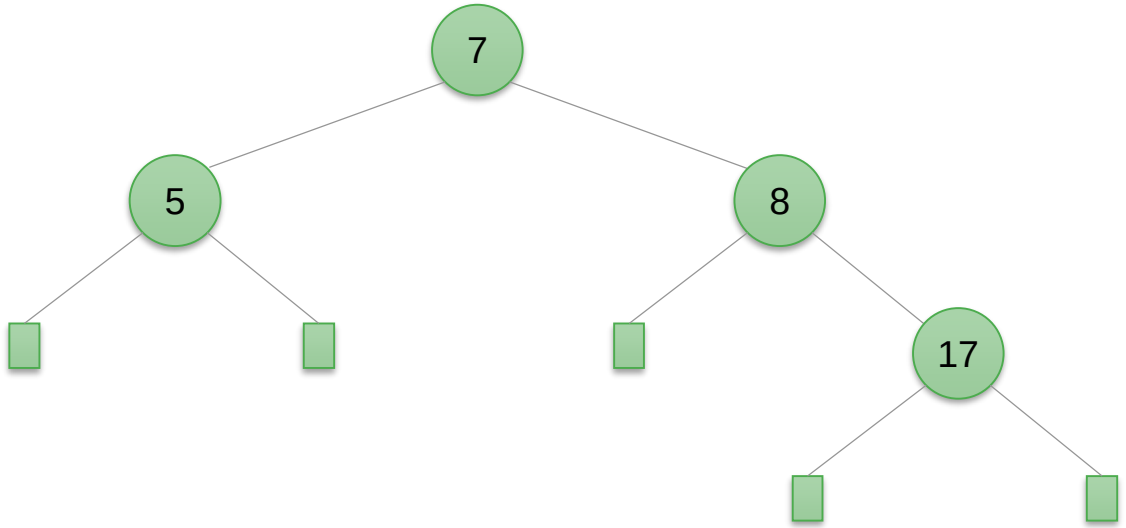
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



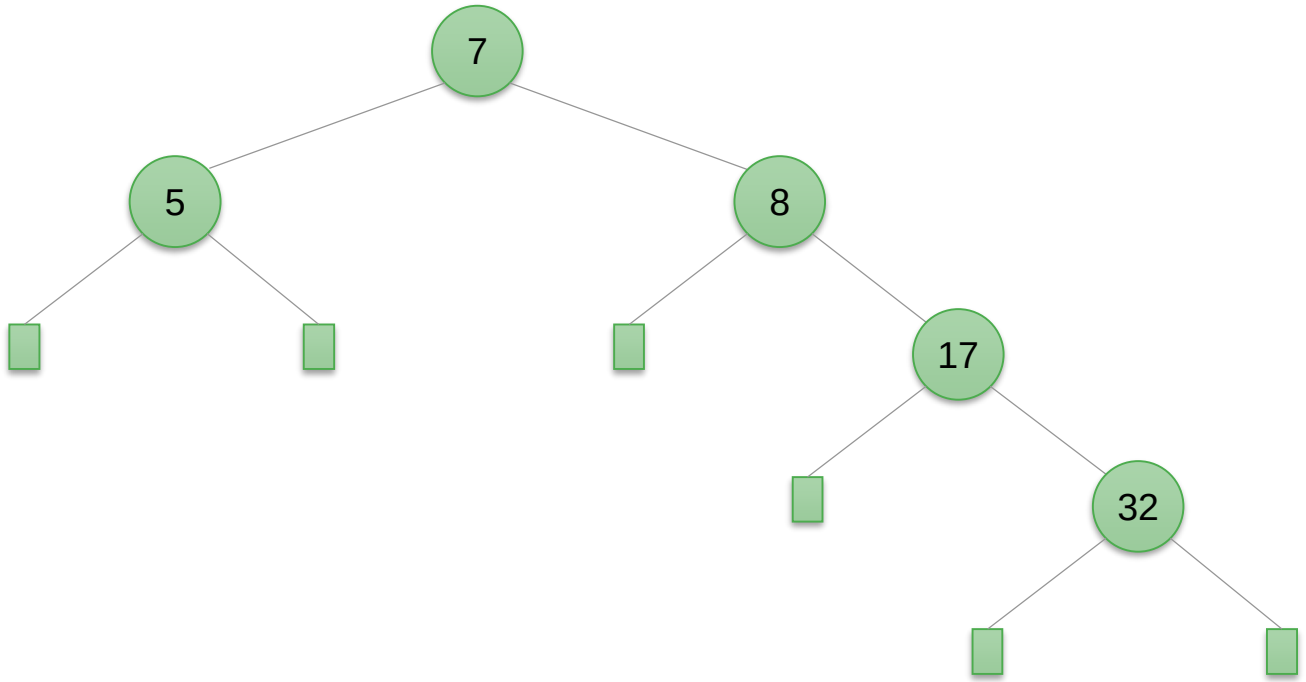
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



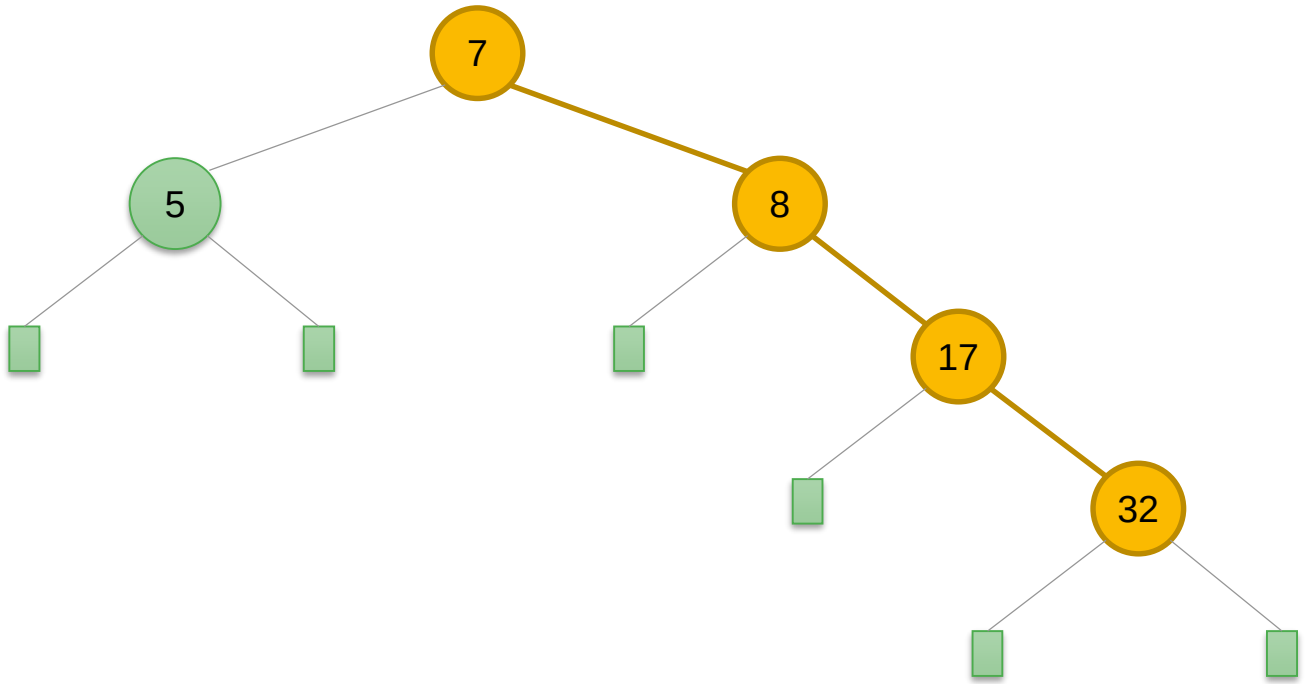
BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



BALANCED TREES :: DEGENERATION

7, 5, 8, 17, 32



List: Access O(n)

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

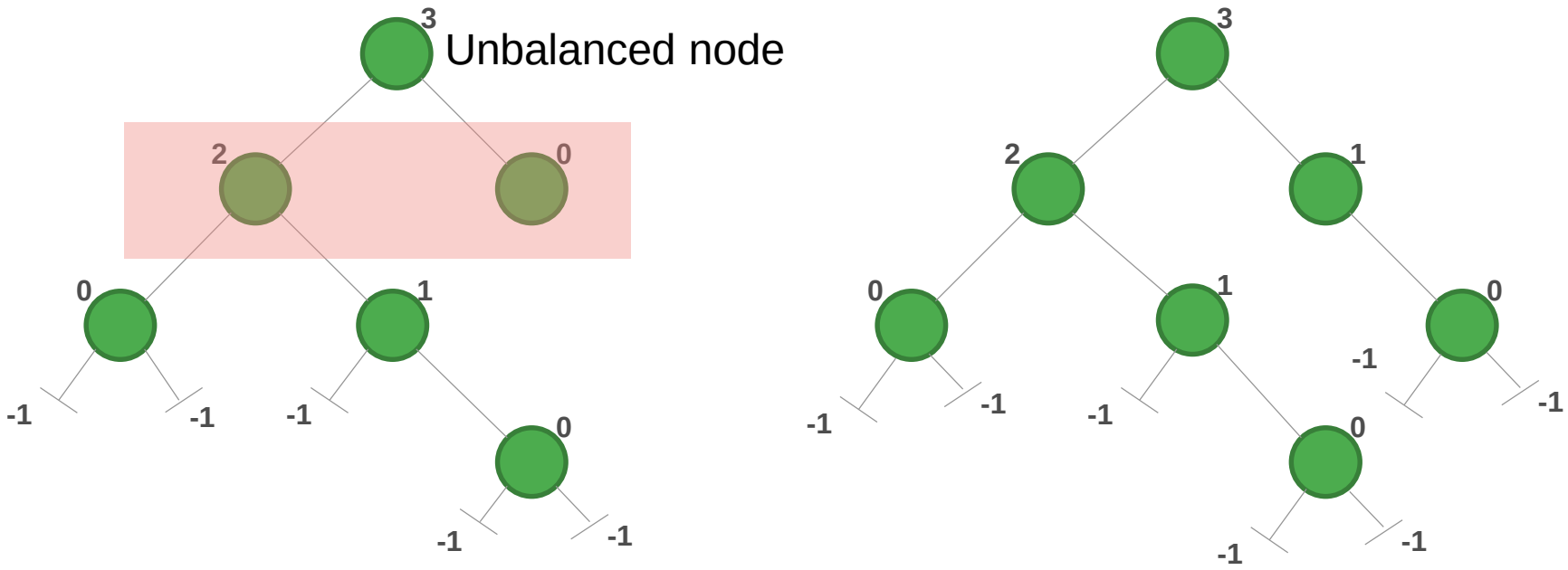
Binary search trees that guarantee the execution of search, insert and delete operations in **O(log(n))** even in worst case → height-balanced trees (e.g.: **AVL tree**)

AVL TREES

Properties

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

Examples



AVL TREES :: INSERT

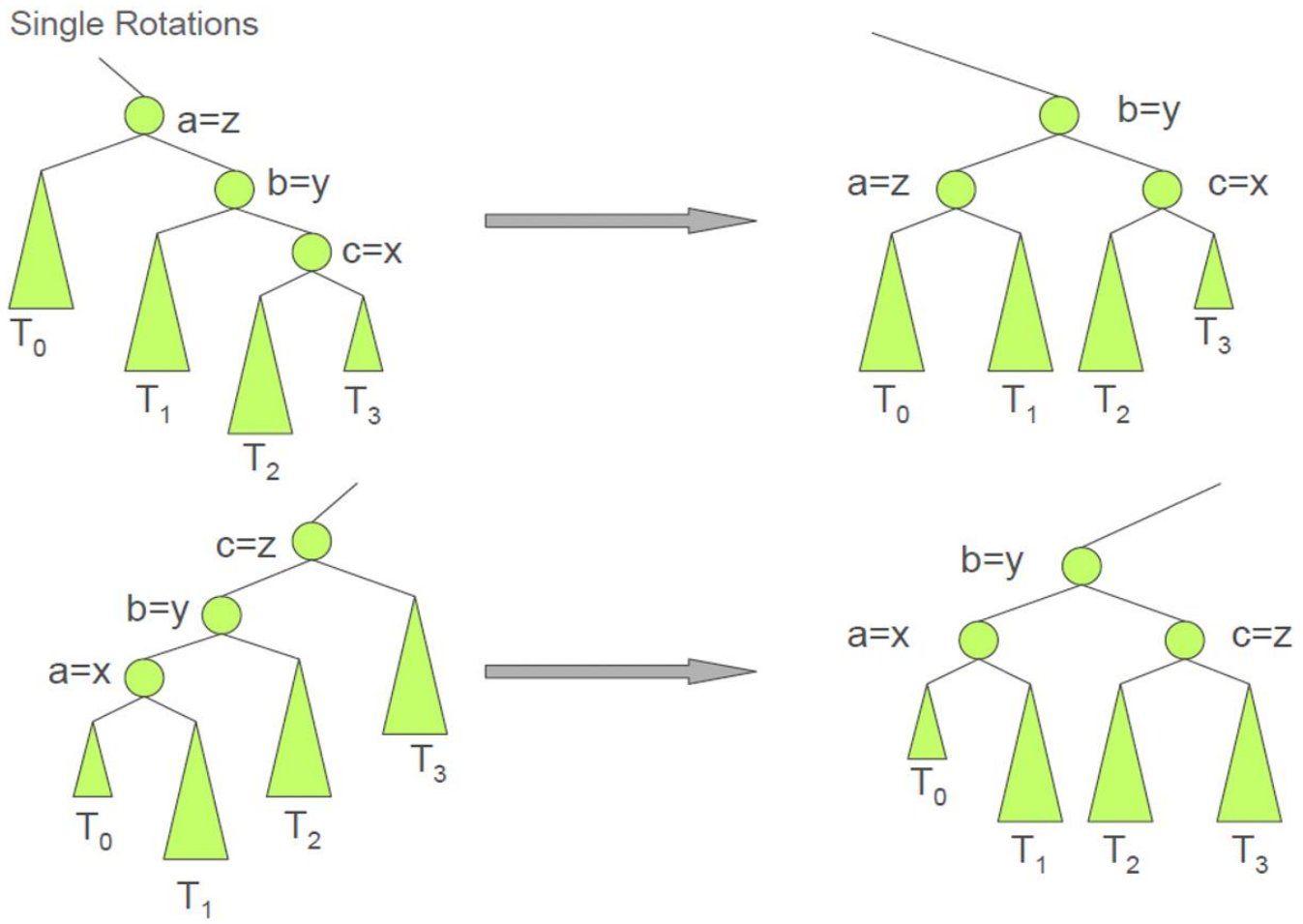
Insert is in general the same as for the binary search tree but may cause the AVL tree to become unbalanced → **restructuring required!**

Restructuring

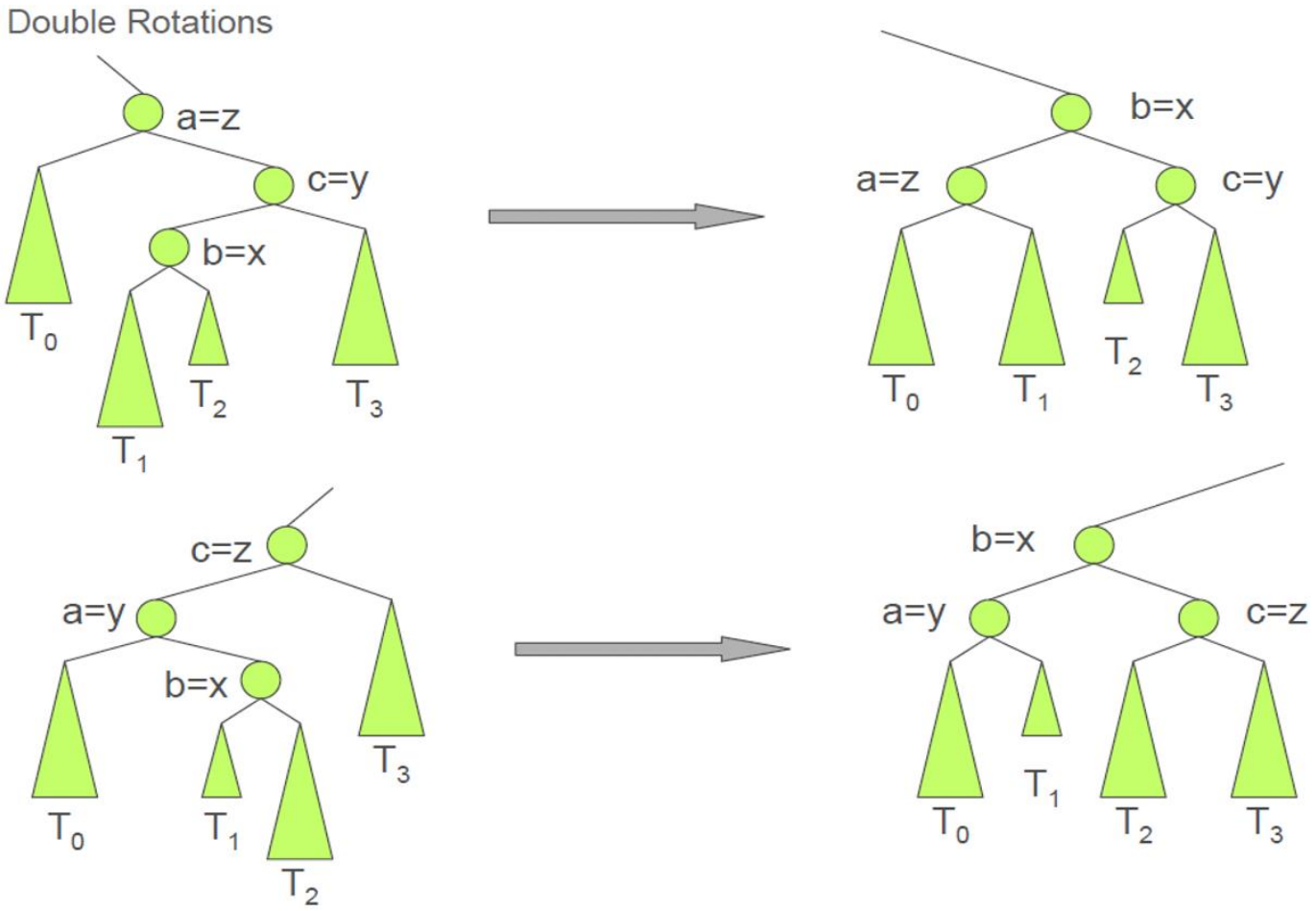
1. Go up from the new node in the tree until the first node **x** is found, whose grandparent **z** is an unbalanced node
2. Define **y** as child of **z** (= the node we passed on the way to z)
3. Define **x** as child of **y**
4. Rename **x**,**y**,**z** in **a**,**b**,**c** (according to Inorder traversal!)
5. Replace **z** (old subroot of unsorted part-tree) by **b** (new subroot of sorted part-tree)
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…**

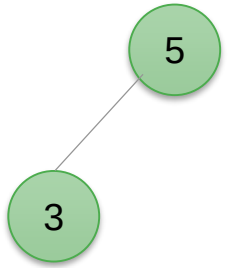
AVL TREES :: ROTATIONS :: 4 CASES



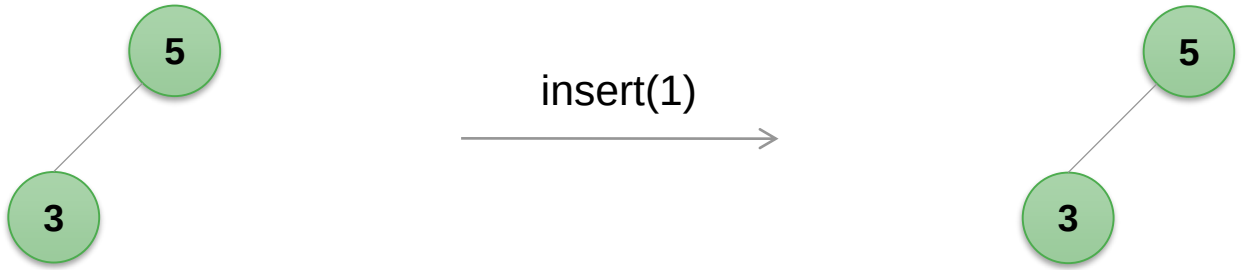
AVL TREES :: ROTATIONS :: 4 CASES



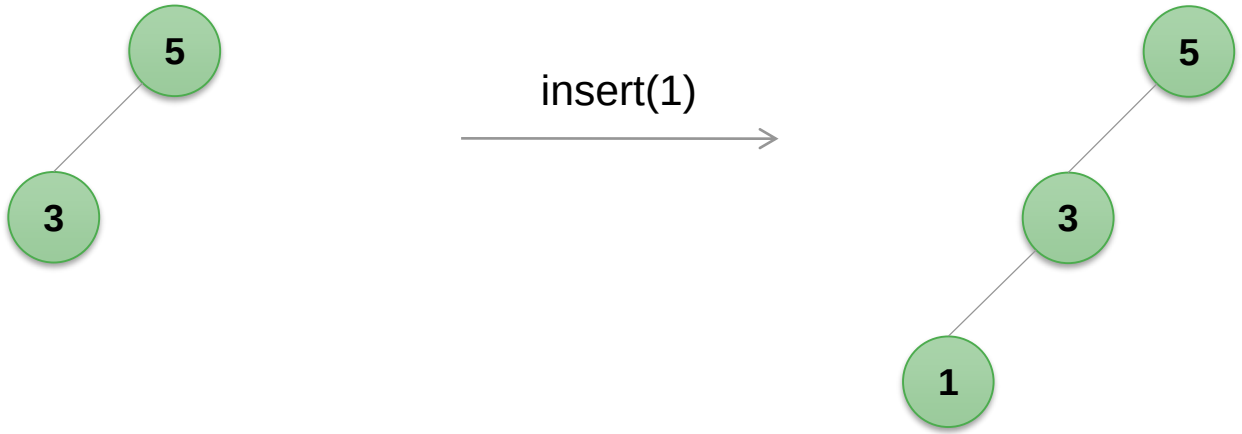
AVL TREES :: ROTATIONS :: EXAMPLE



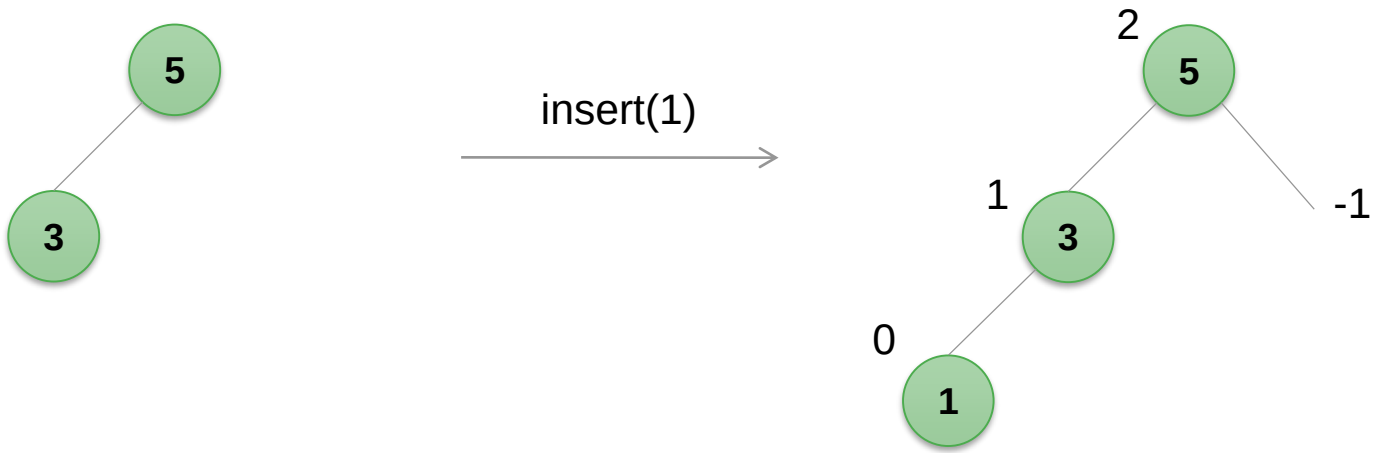
AVL TREES :: ROTATIONS :: EXAMPLE



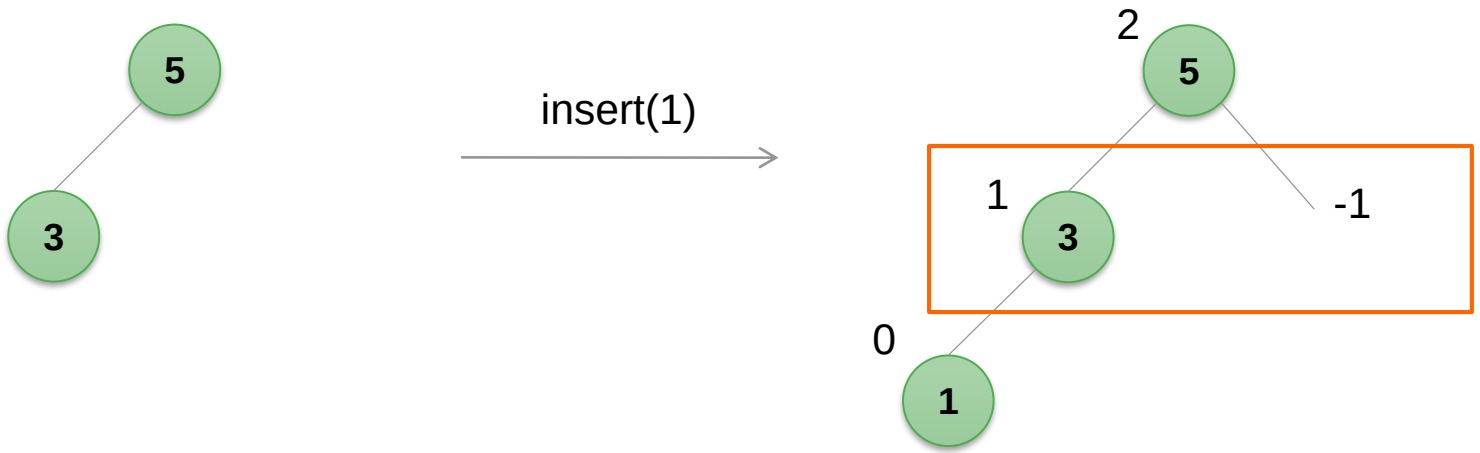
AVL TREES :: ROTATIONS :: EXAMPLE



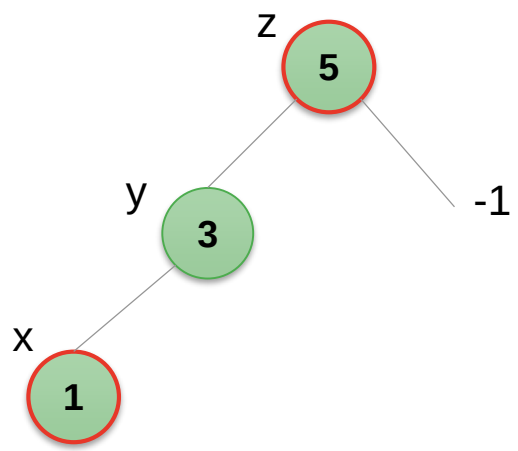
AVL TREES :: ROTATIONS :: EXAMPLE



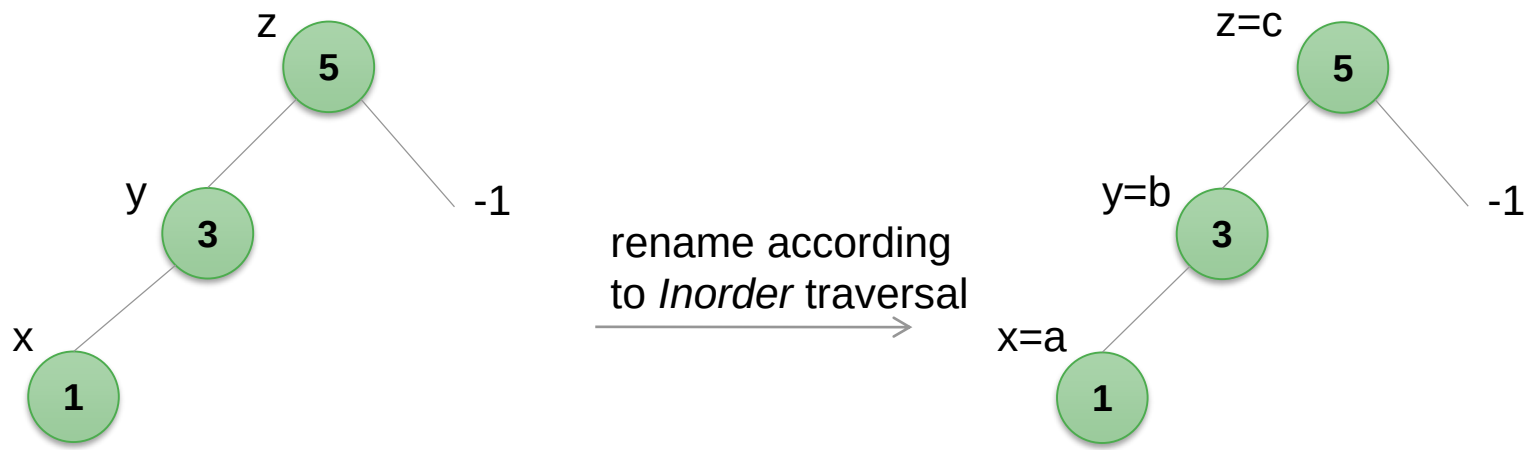
AVL TREES :: ROTATIONS :: EXAMPLE



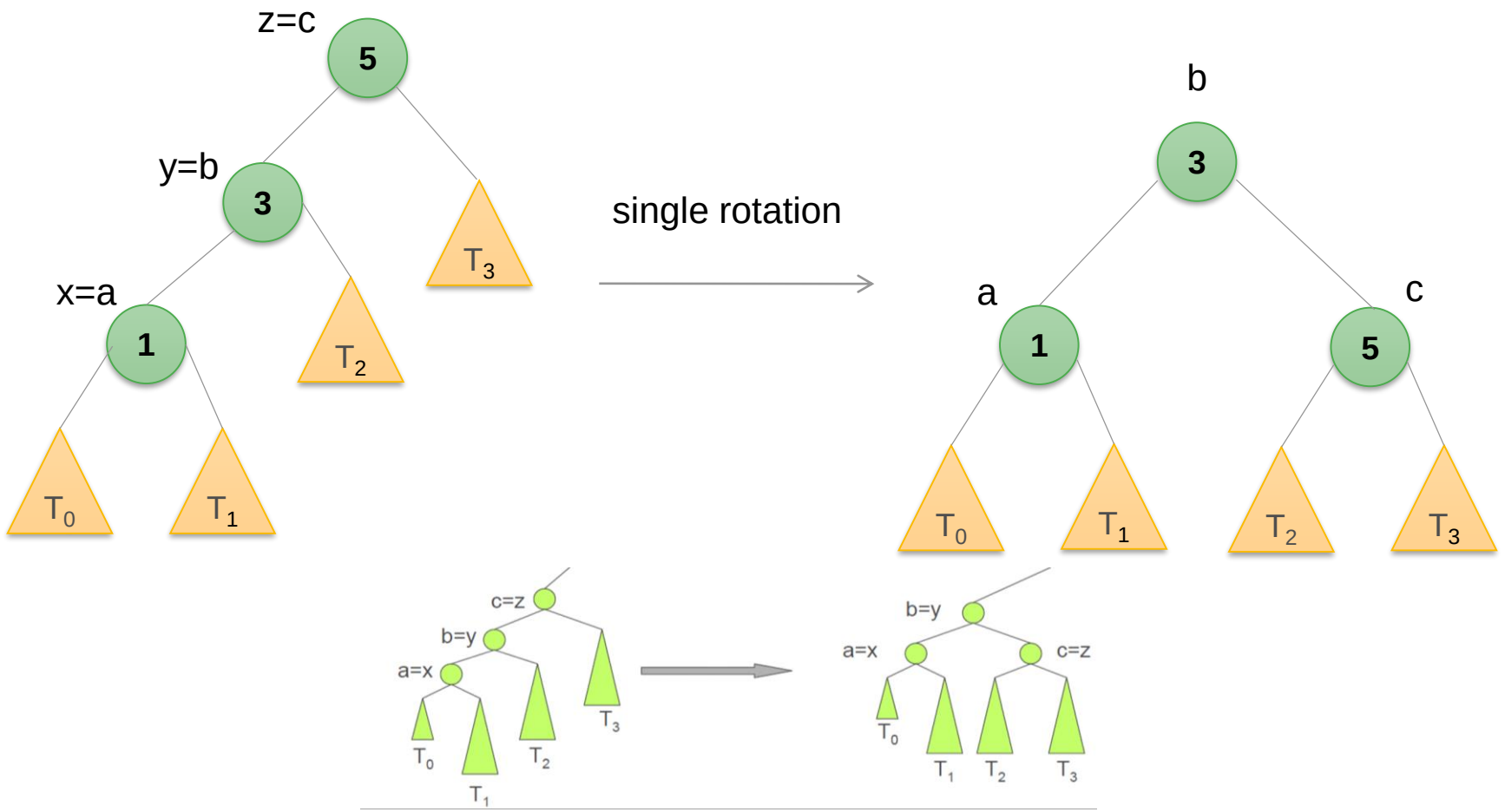
AVL TREES :: ROTATIONS :: EXAMPLE



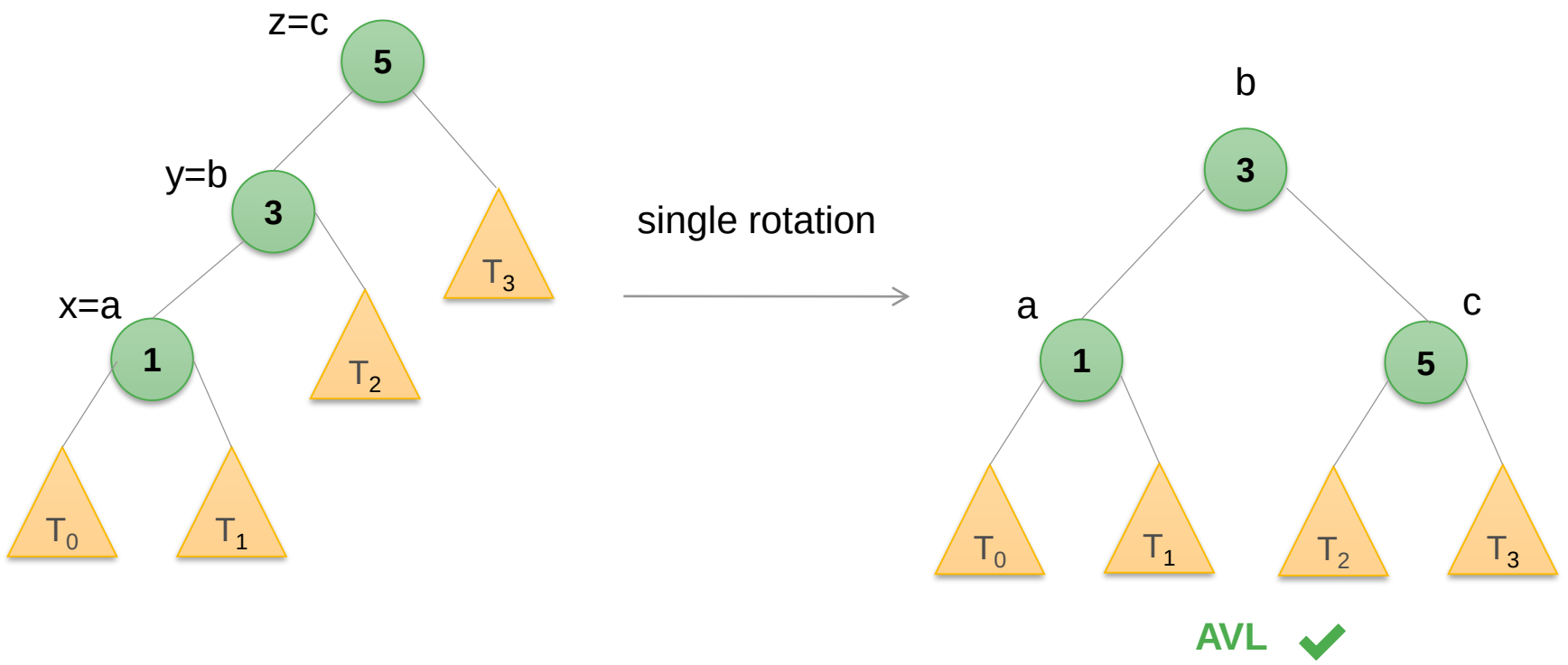
AVL TREES :: ROTATIONS :: EXAMPLE



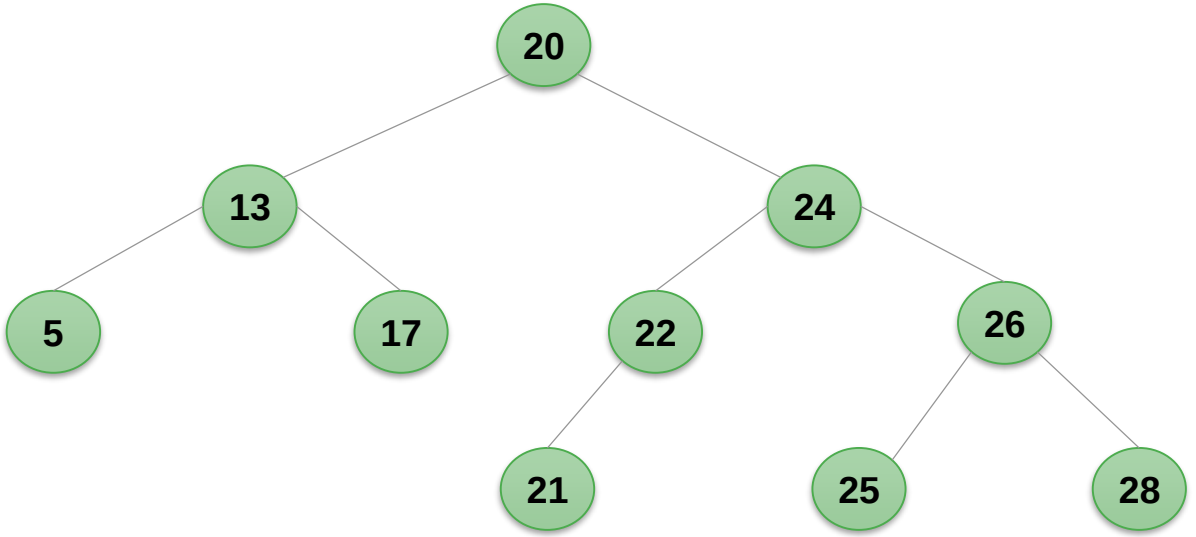
AVL TREES :: ROTATIONS :: EXAMPLE



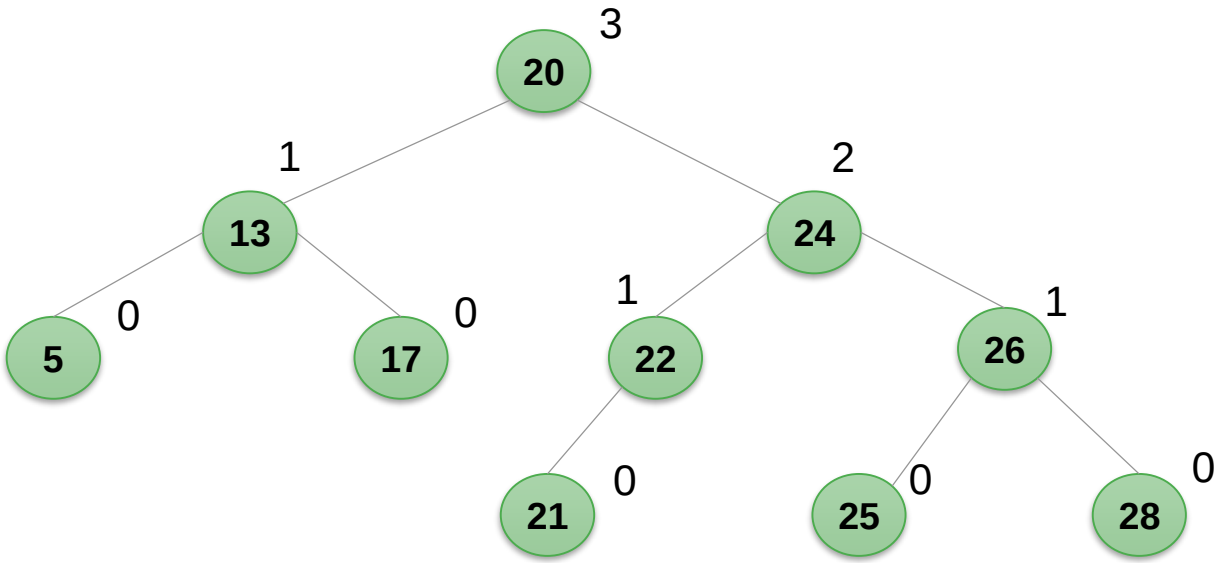
AVL TREES :: ROTATIONS :: EXAMPLE



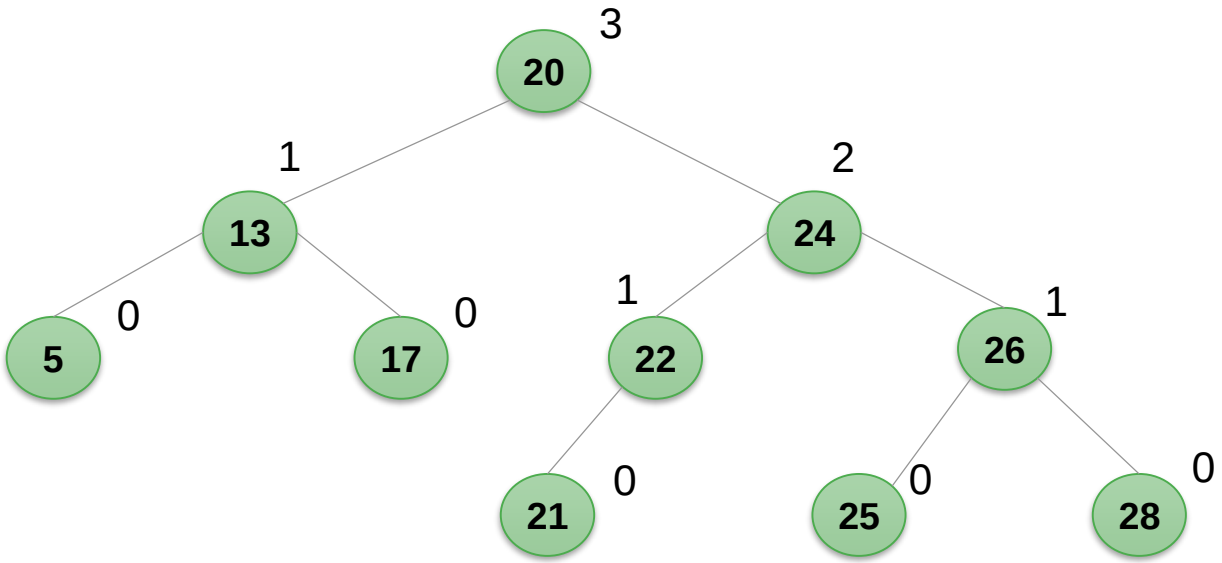
AVL TREES :: SINGLE ROTATION EXAMPLE



AVL TREES :: SINGLE ROTATION EXAMPLE

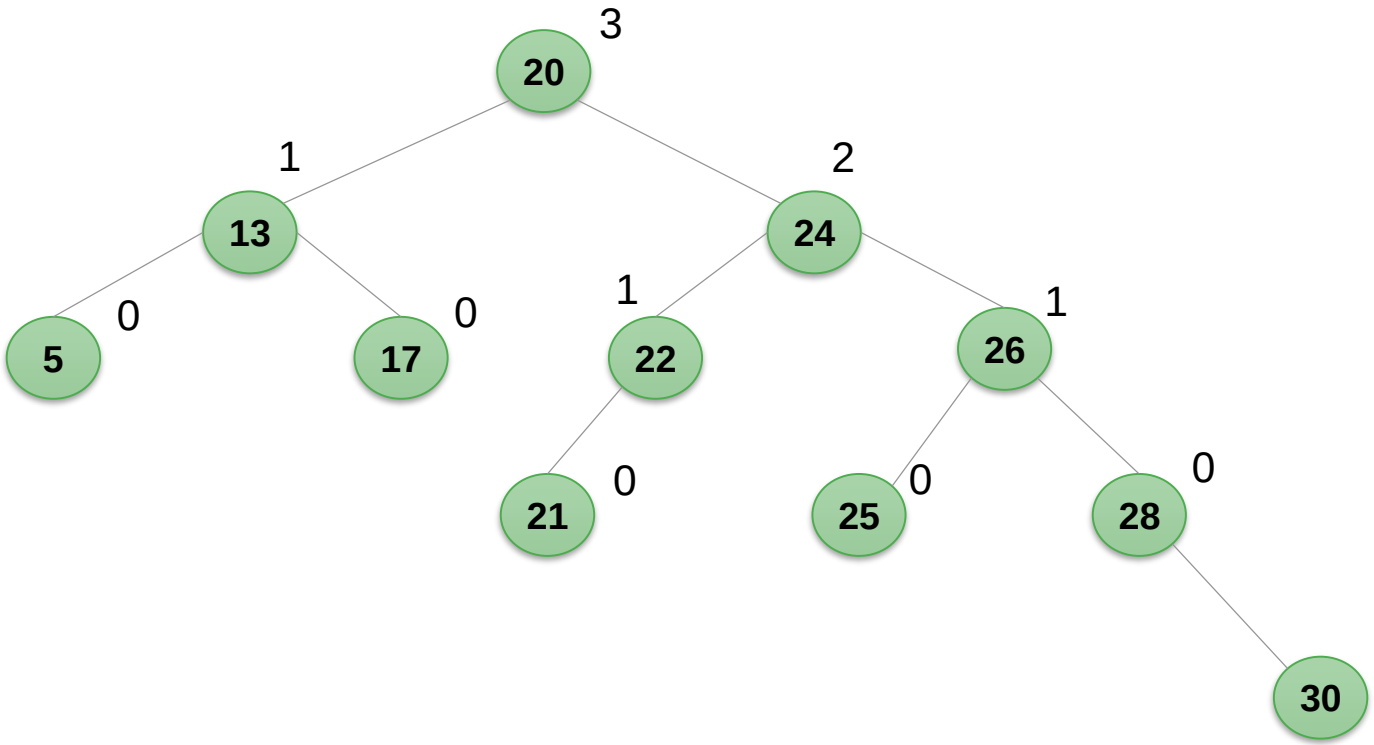


AVL TREES :: SINGLE ROTATION EXAMPLE



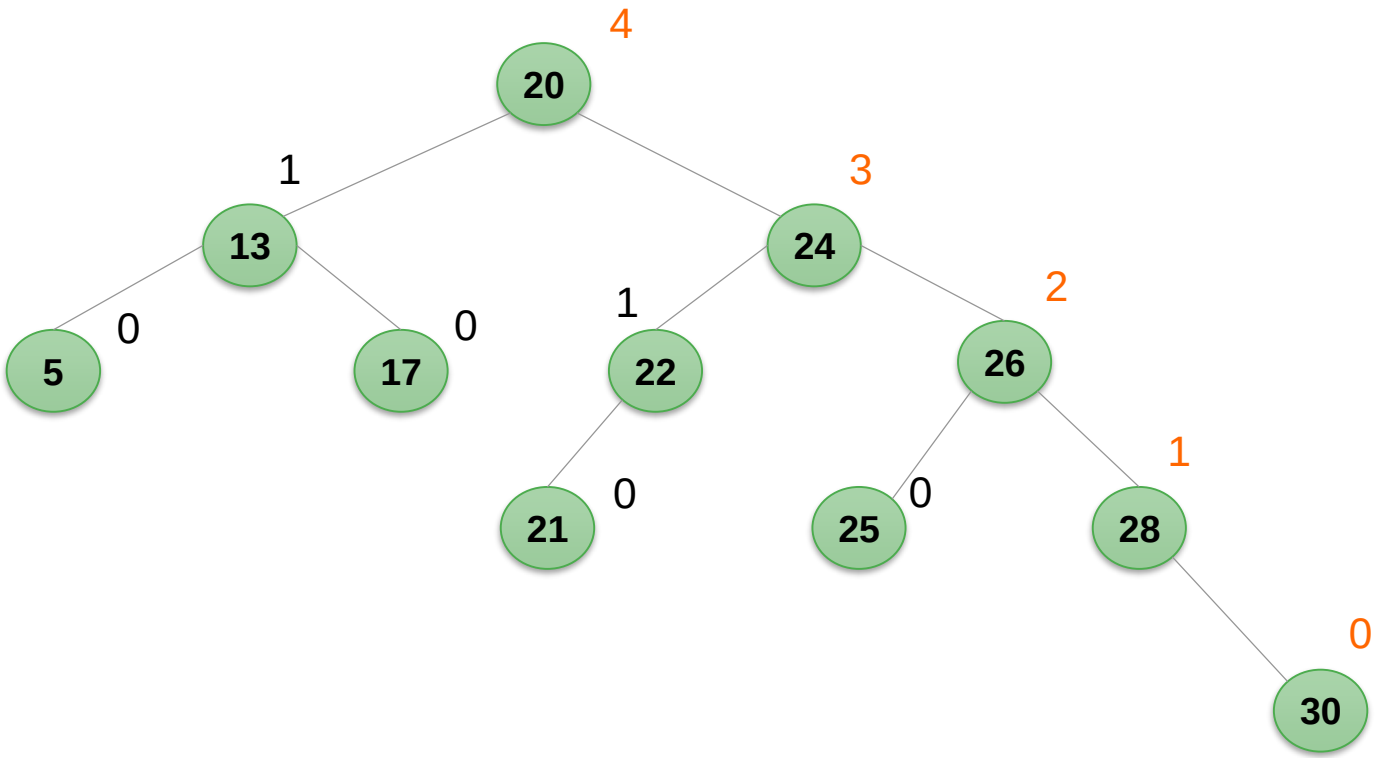
insert ‘30’

AVL TREES :: SINGLE ROTATION EXAMPLE



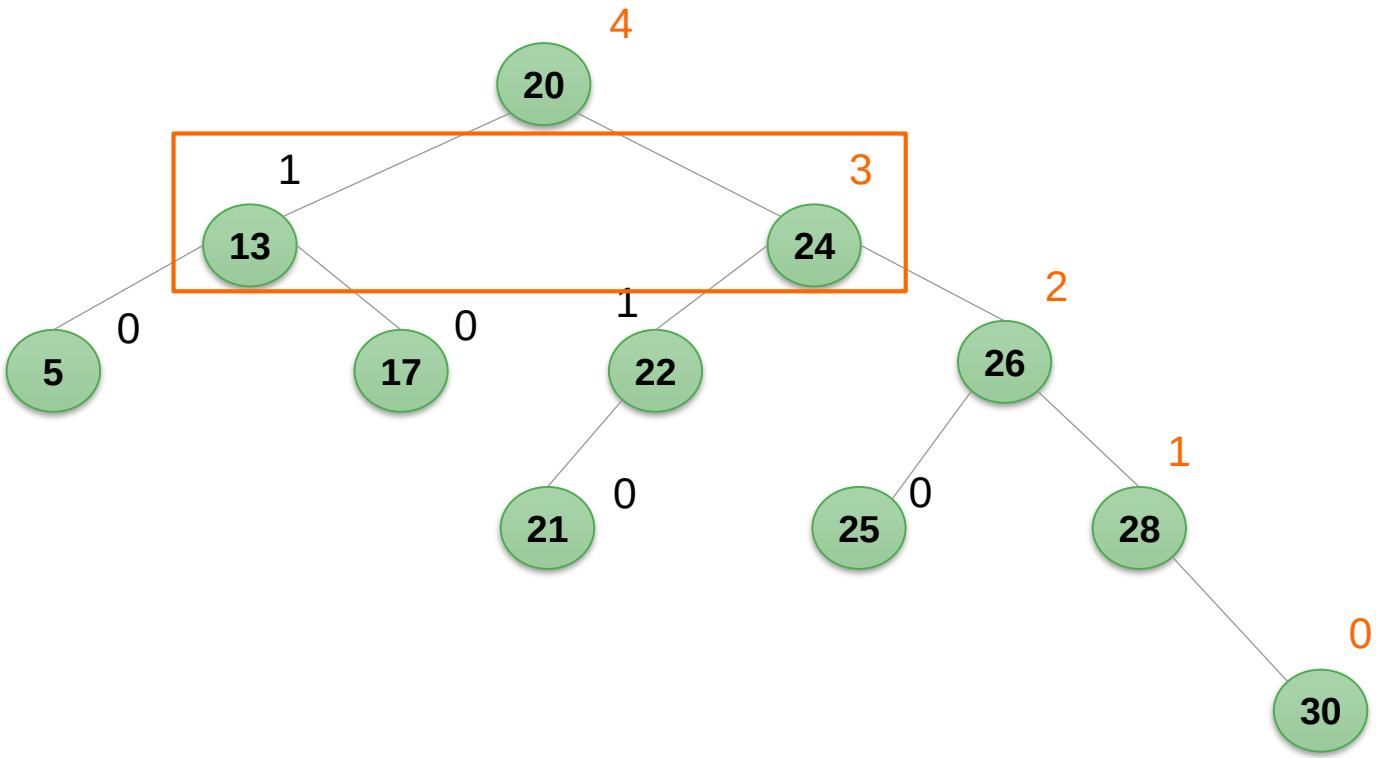
insert ‘30’

AVL TREES :: SINGLE ROTATION EXAMPLE



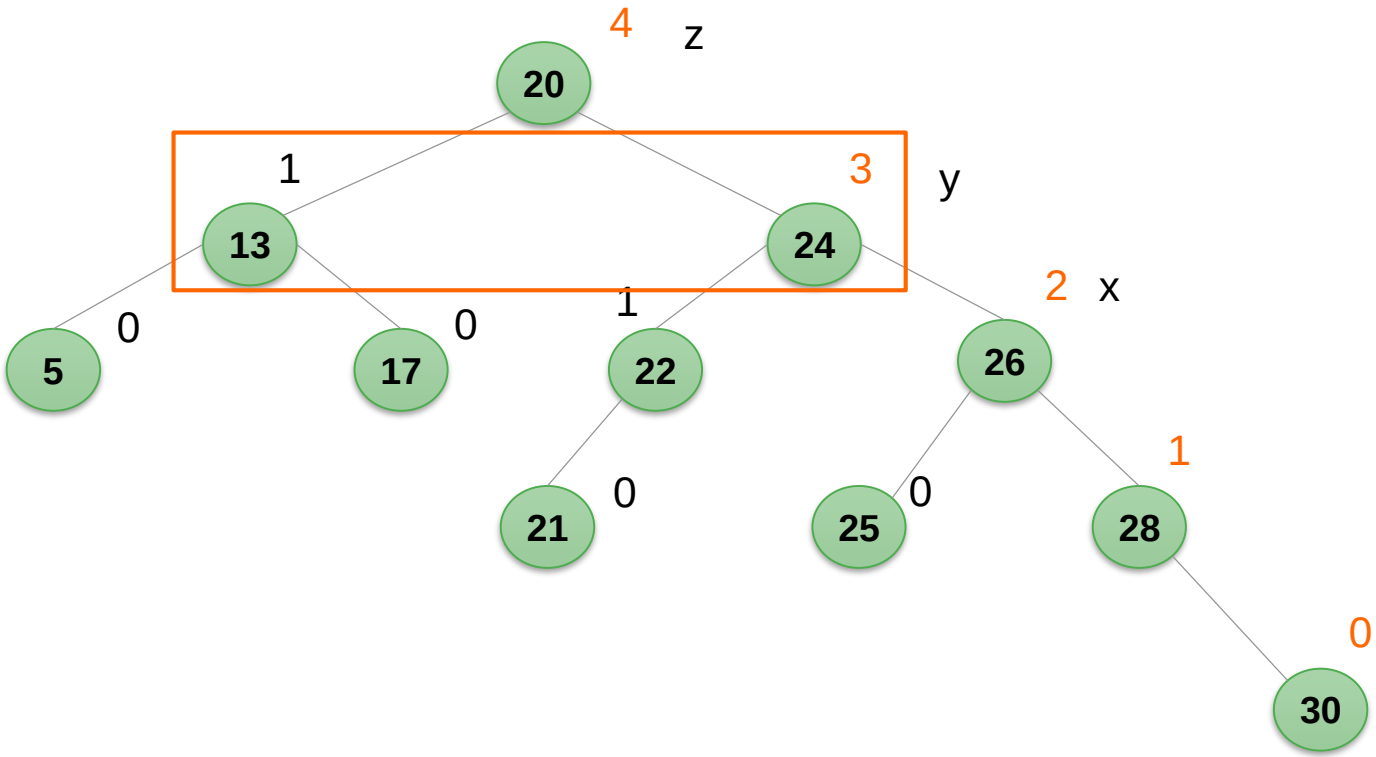
insert ‘30’

AVL TREES :: SINGLE ROTATION EXAMPLE



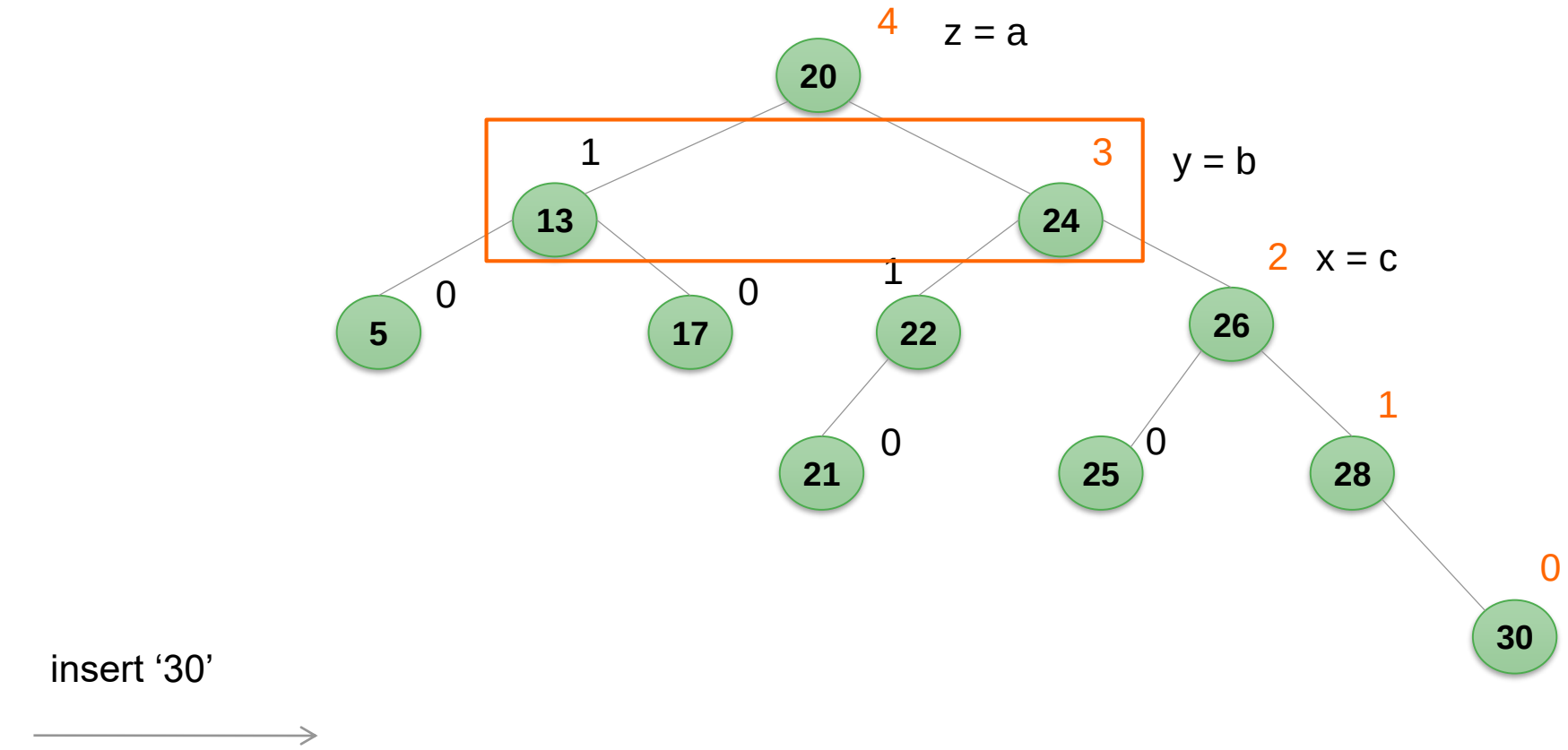
insert ‘30’

AVL TREES :: SINGLE ROTATION EXAMPLE

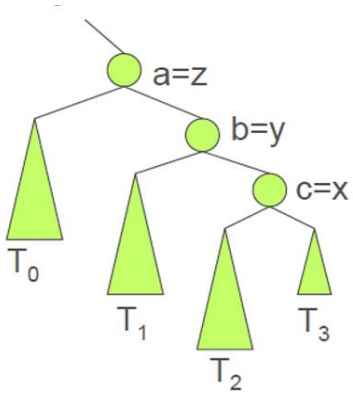


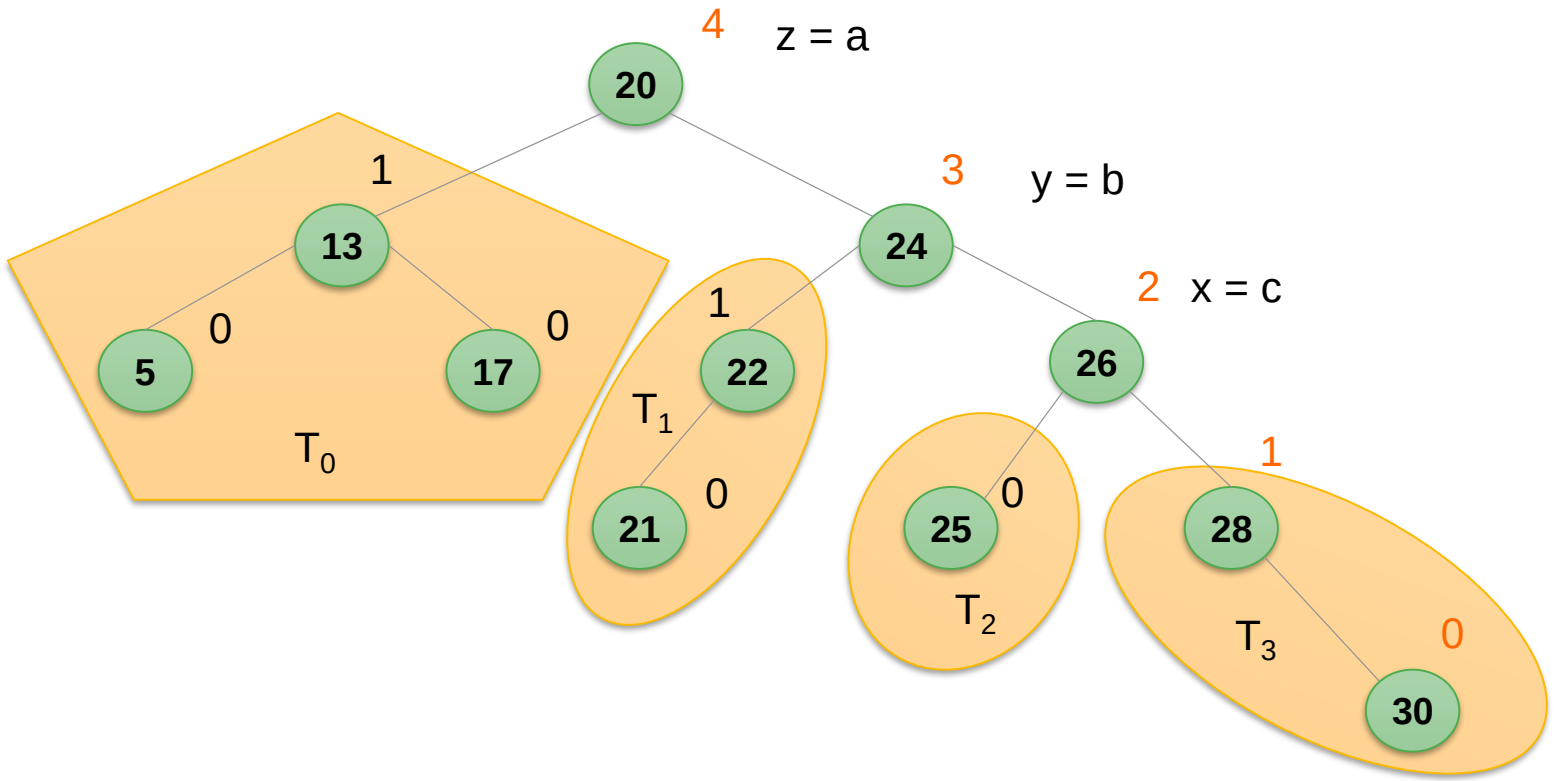
insert ‘30’

AVL TREES :: SINGLE ROTATION EXAMPLE

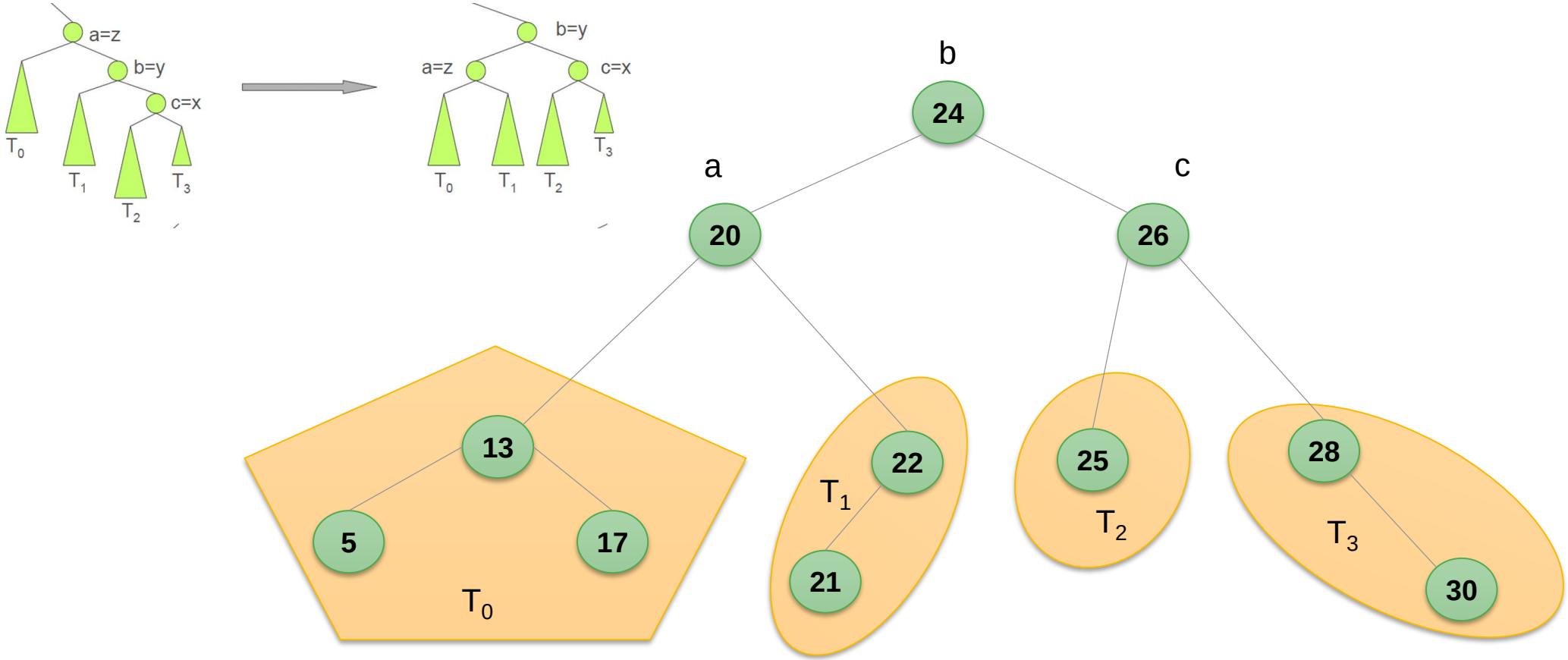


AVL TREES :: SINGLE ROTATION EXAMPLE

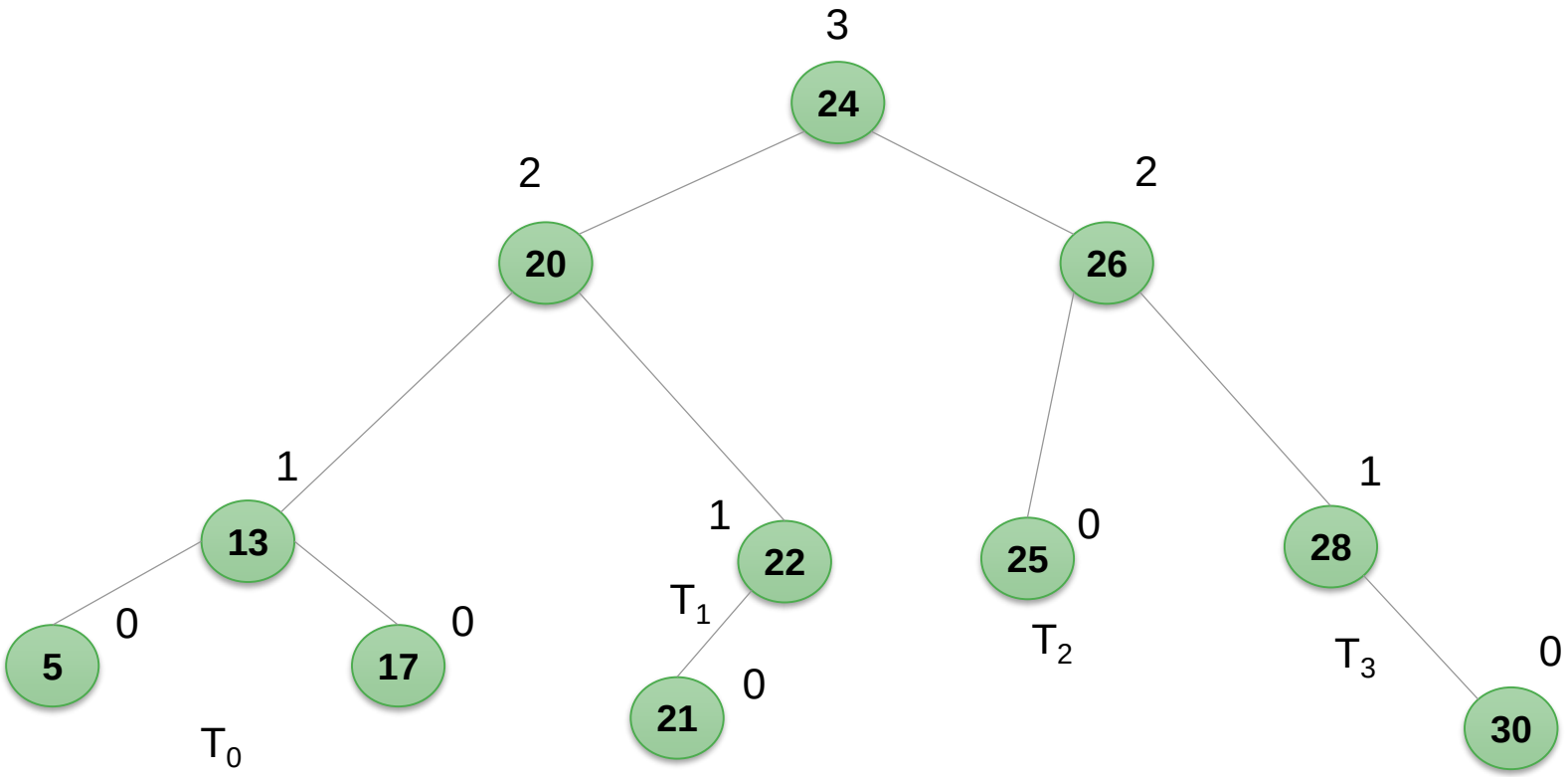




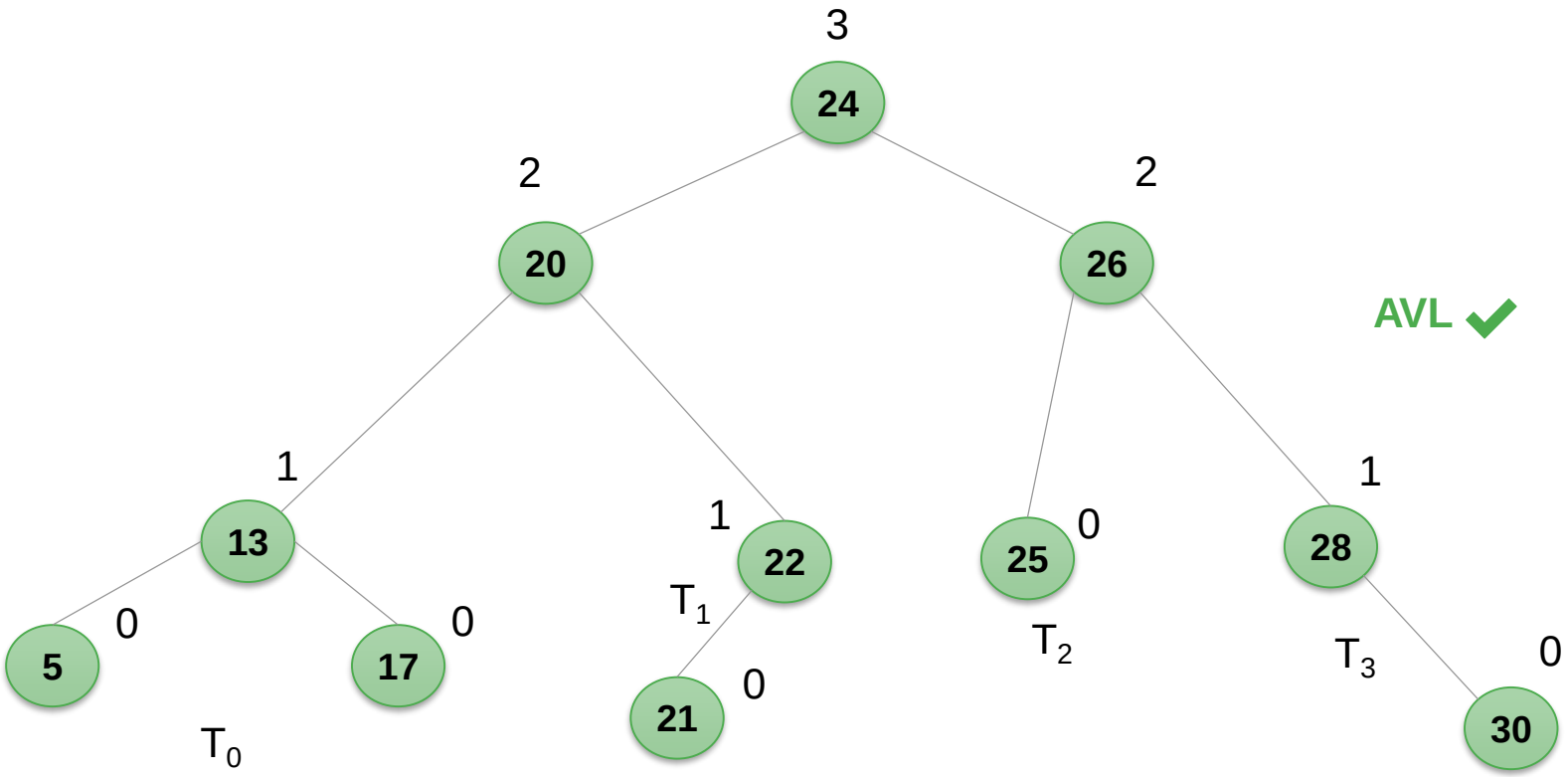
AVL TREES :: SINGLE ROTATION EXAMPLE



AVL TREES :: SINGLE ROTATION EXAMPLE



AVL TREES :: SINGLE ROTATION EXAMPLE



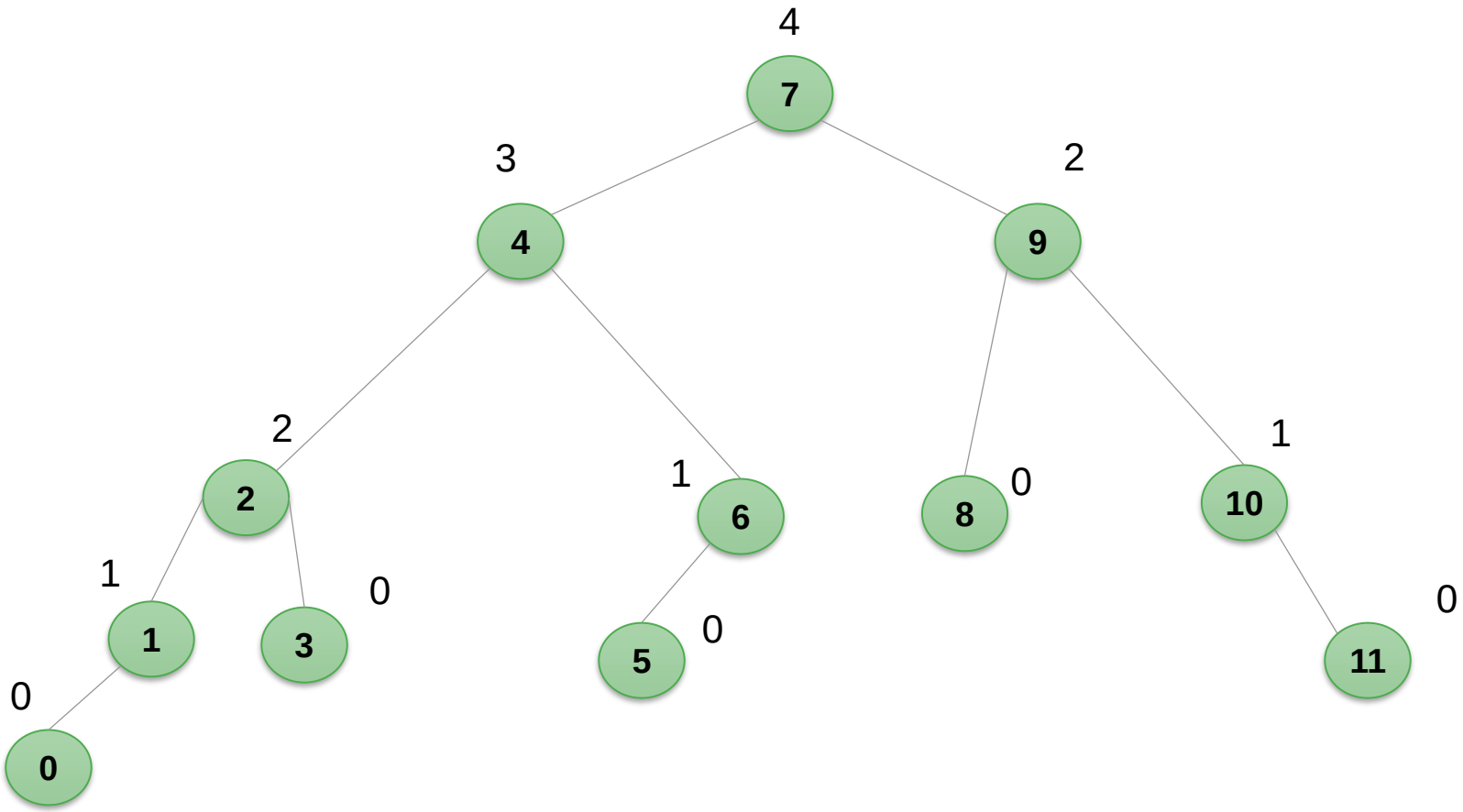
AVL TREES :: REMOVE

- Remove as in binary search tree
- Check the balance starting from the **parent** node of the moved **Inorder** successor towards the root
- **Restructure** if necessary until the tree is balanced

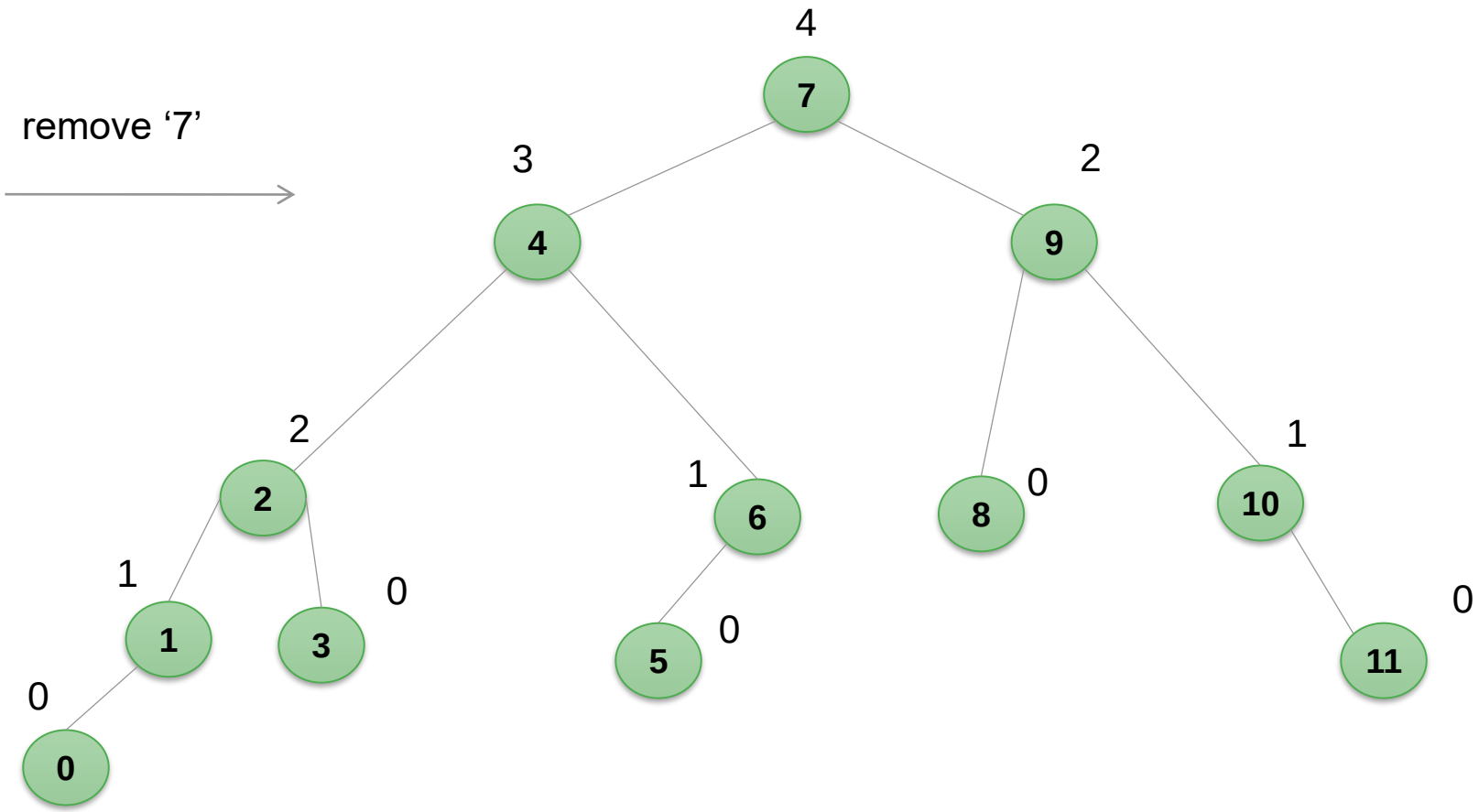
Procedure

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

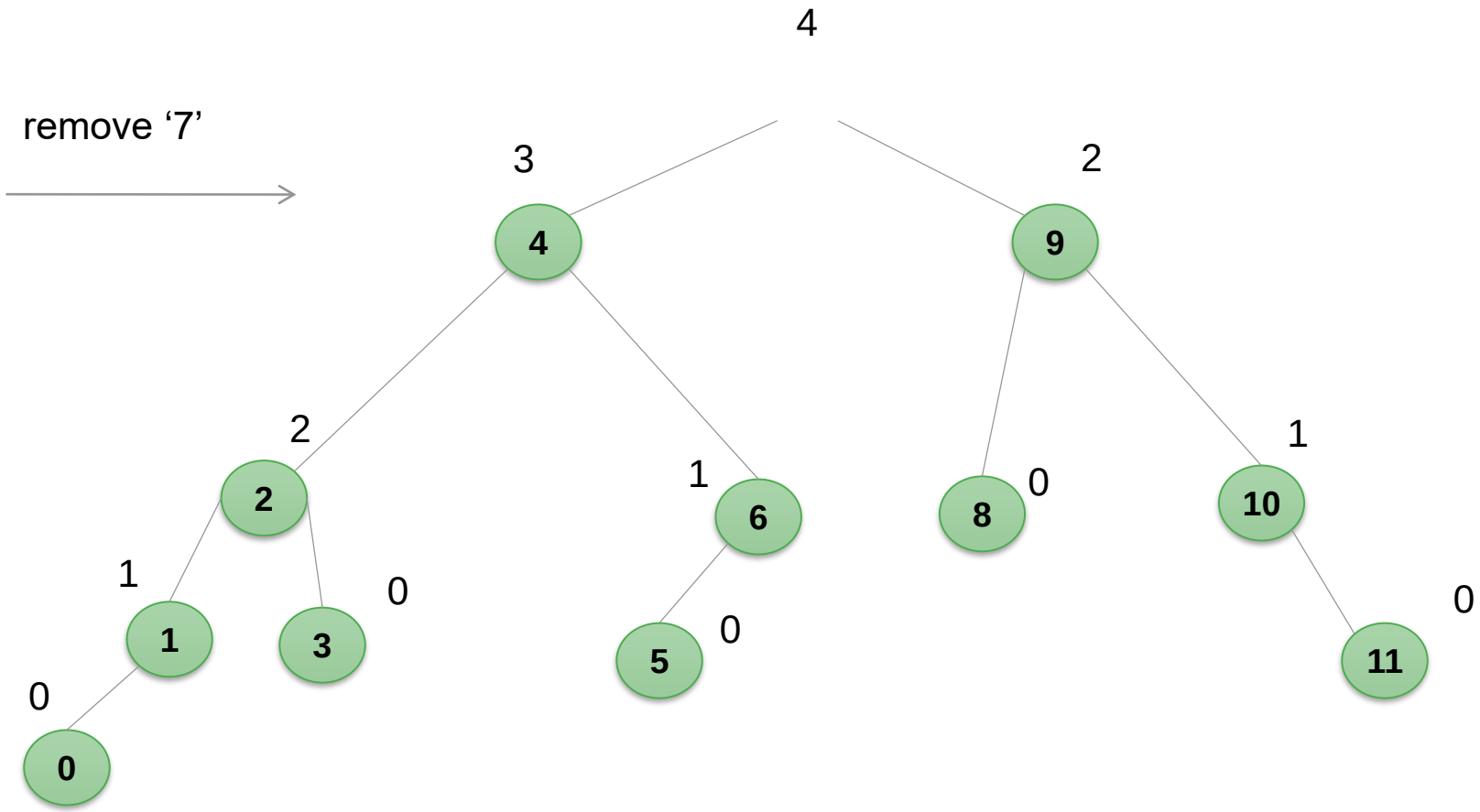
AVL TREES :: REMOVE



AVL TREES :: REMOVE

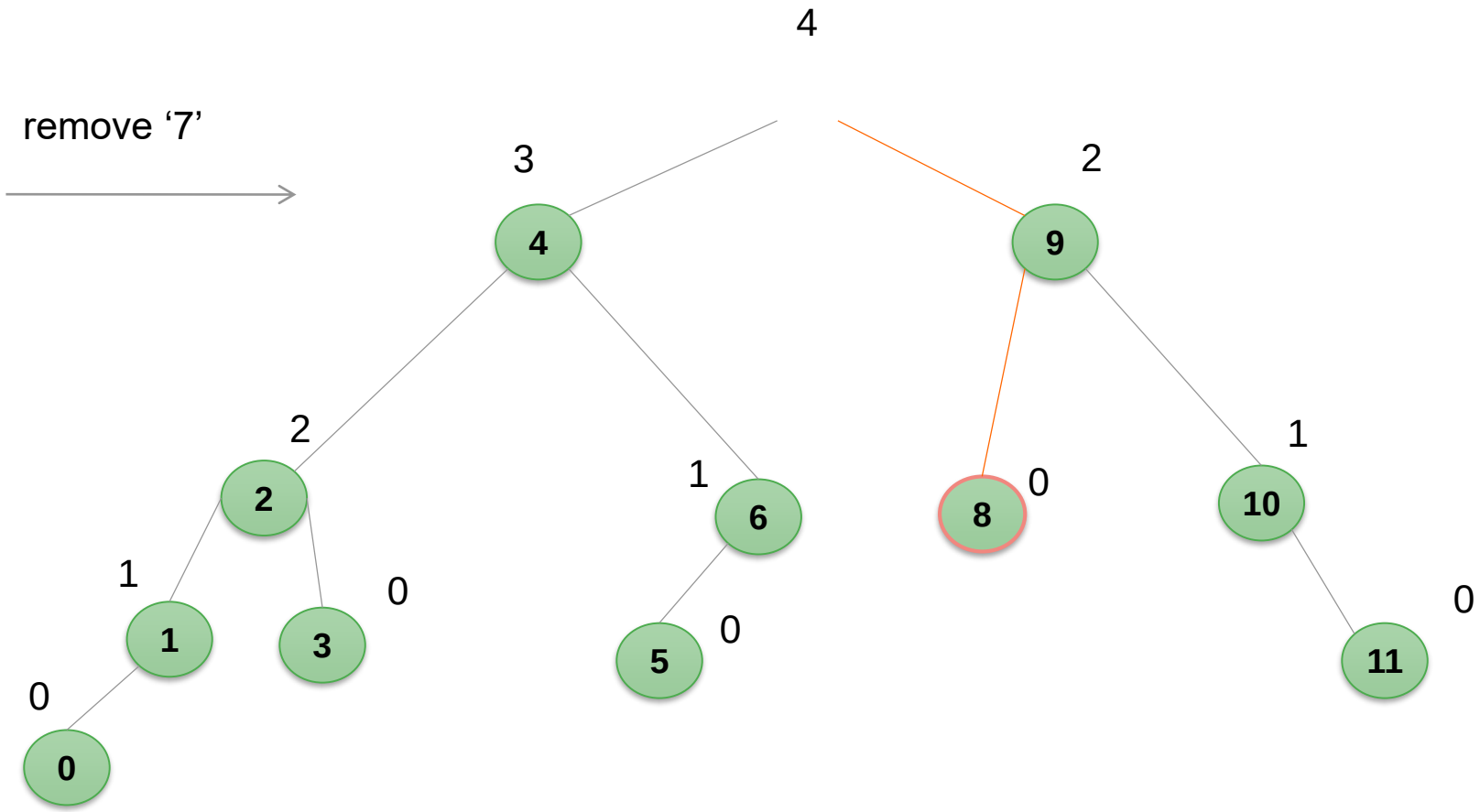


AVL TREES :: REMOVE

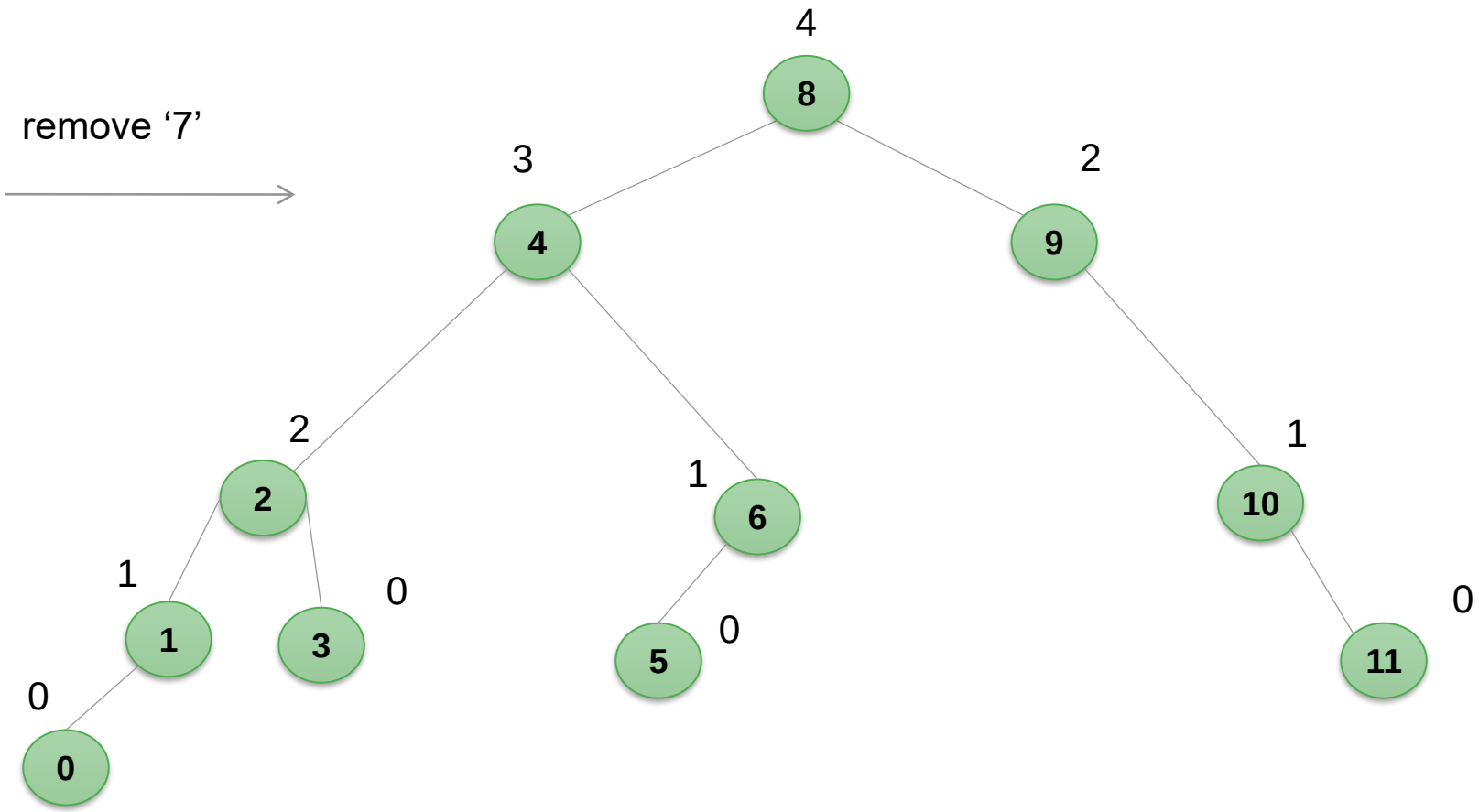


AVL TREES :: REMOVE

BST remove:
‘8’ is inorder successor of removed node ‘7’

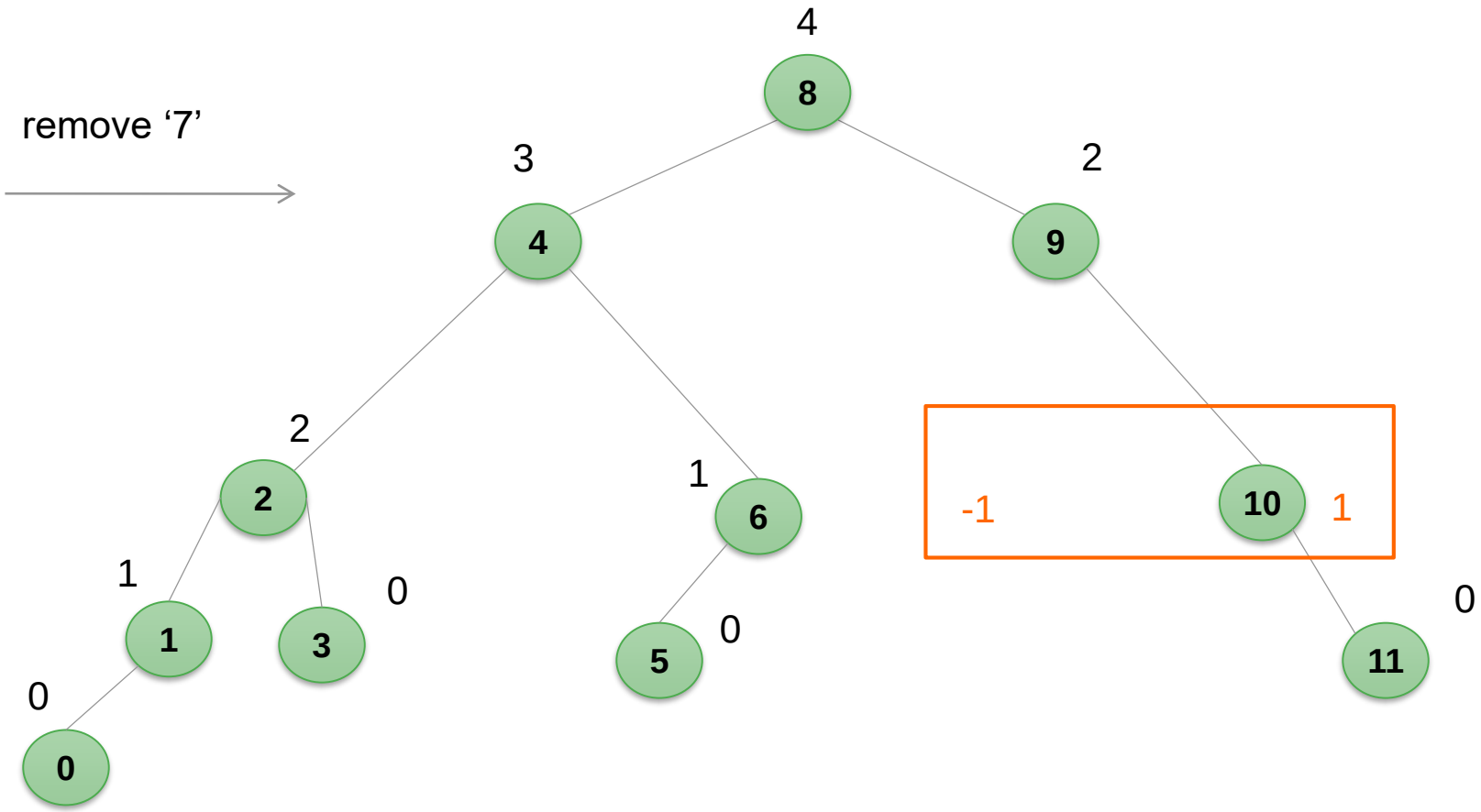


AVL TREES :: REMOVE



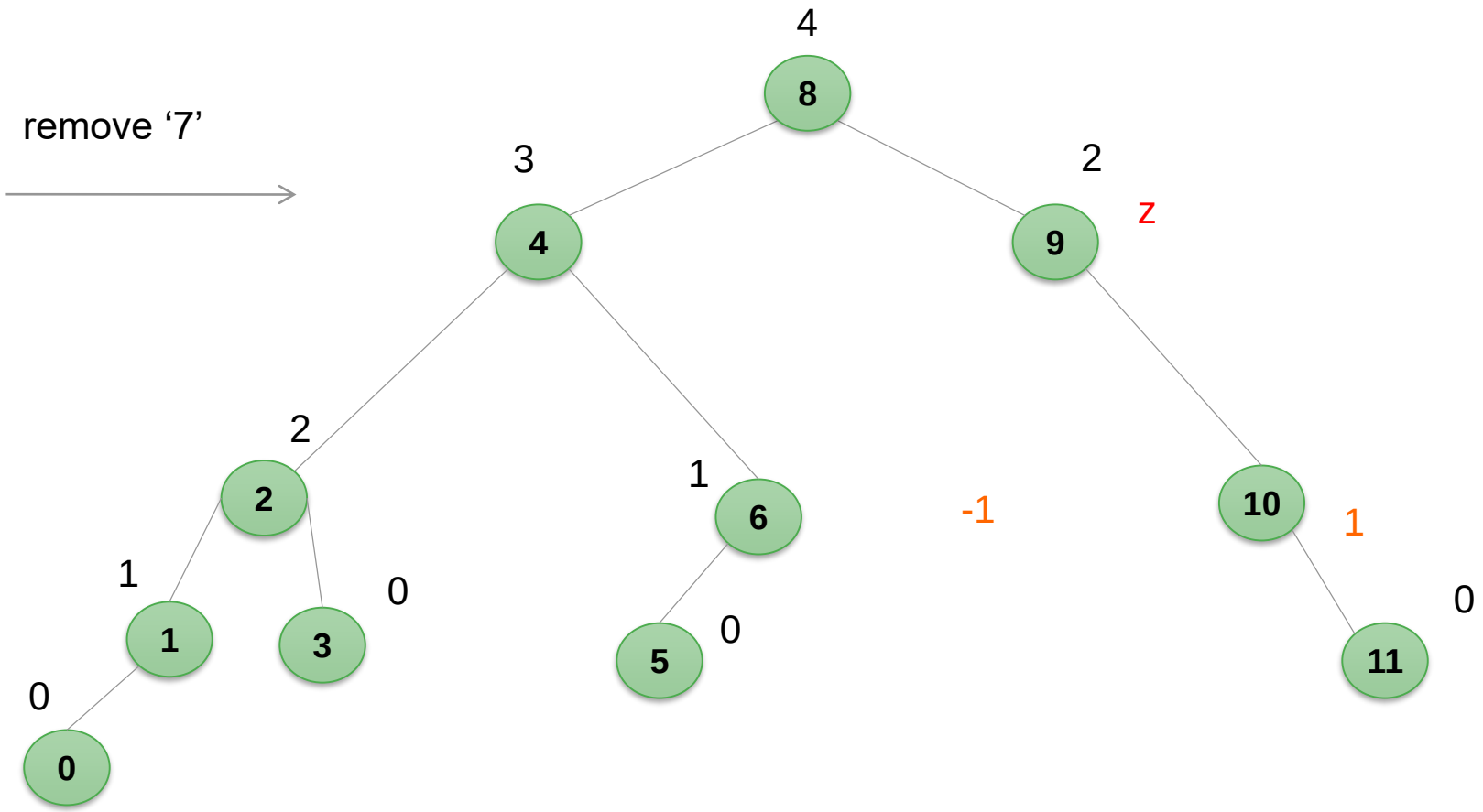
AVL TREES :: REMOVE

‘8’ was inorder successor of removed node ‘7’ and its parent node is ‘9’



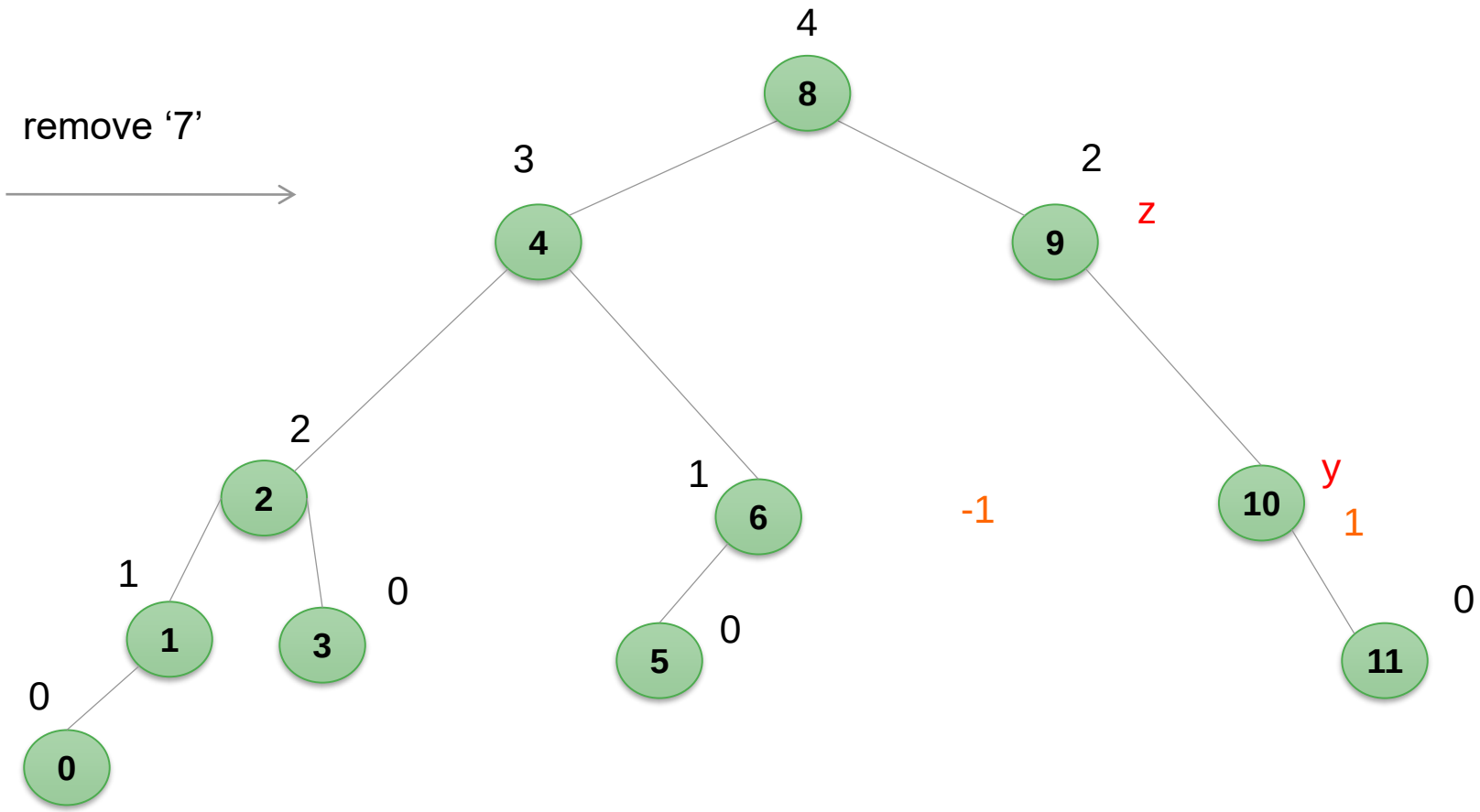
AVL TREES :: REMOVE

First unbalanced node = z



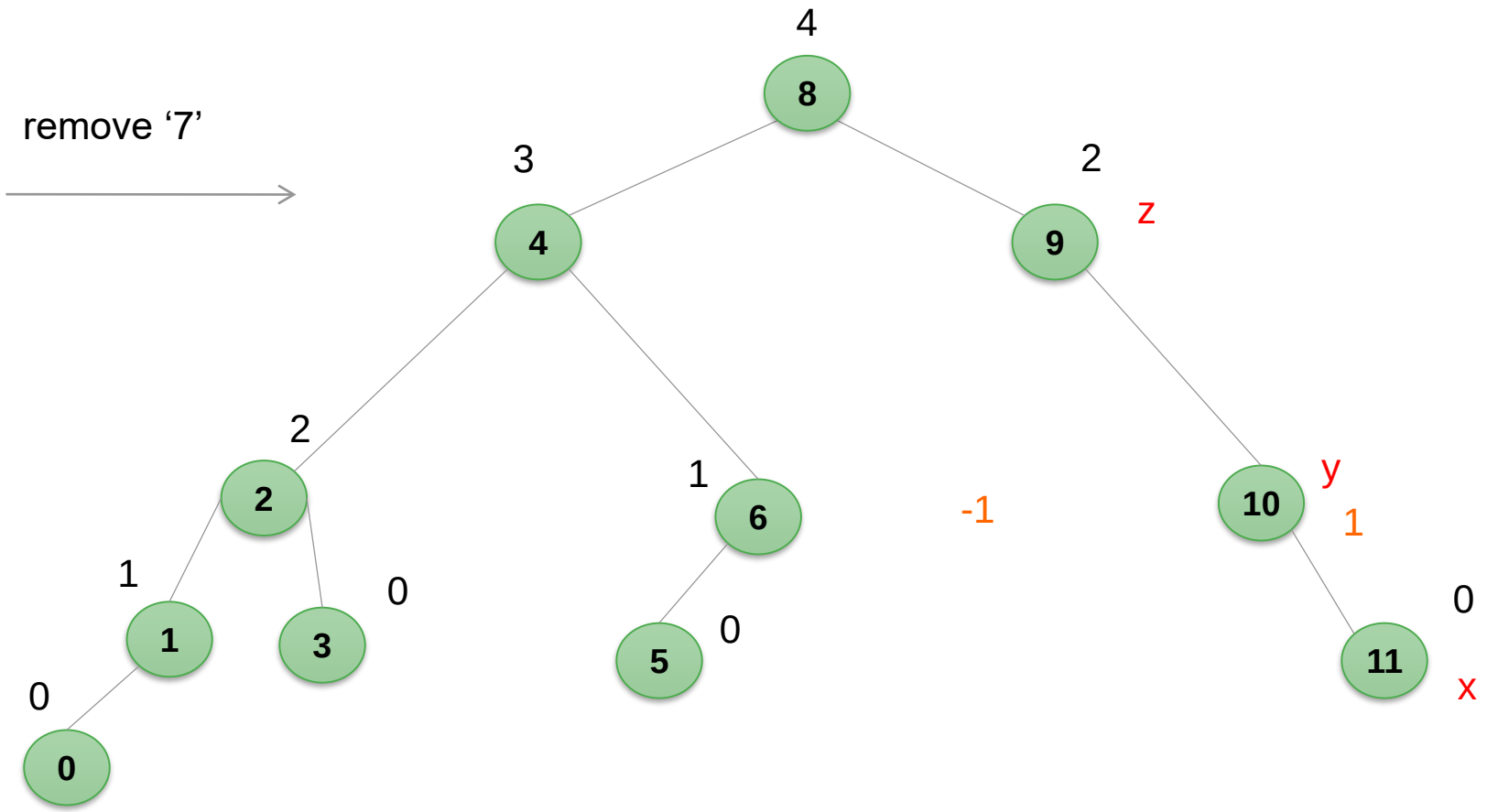
AVL TREES :: REMOVE

Child of z with greatest height = y



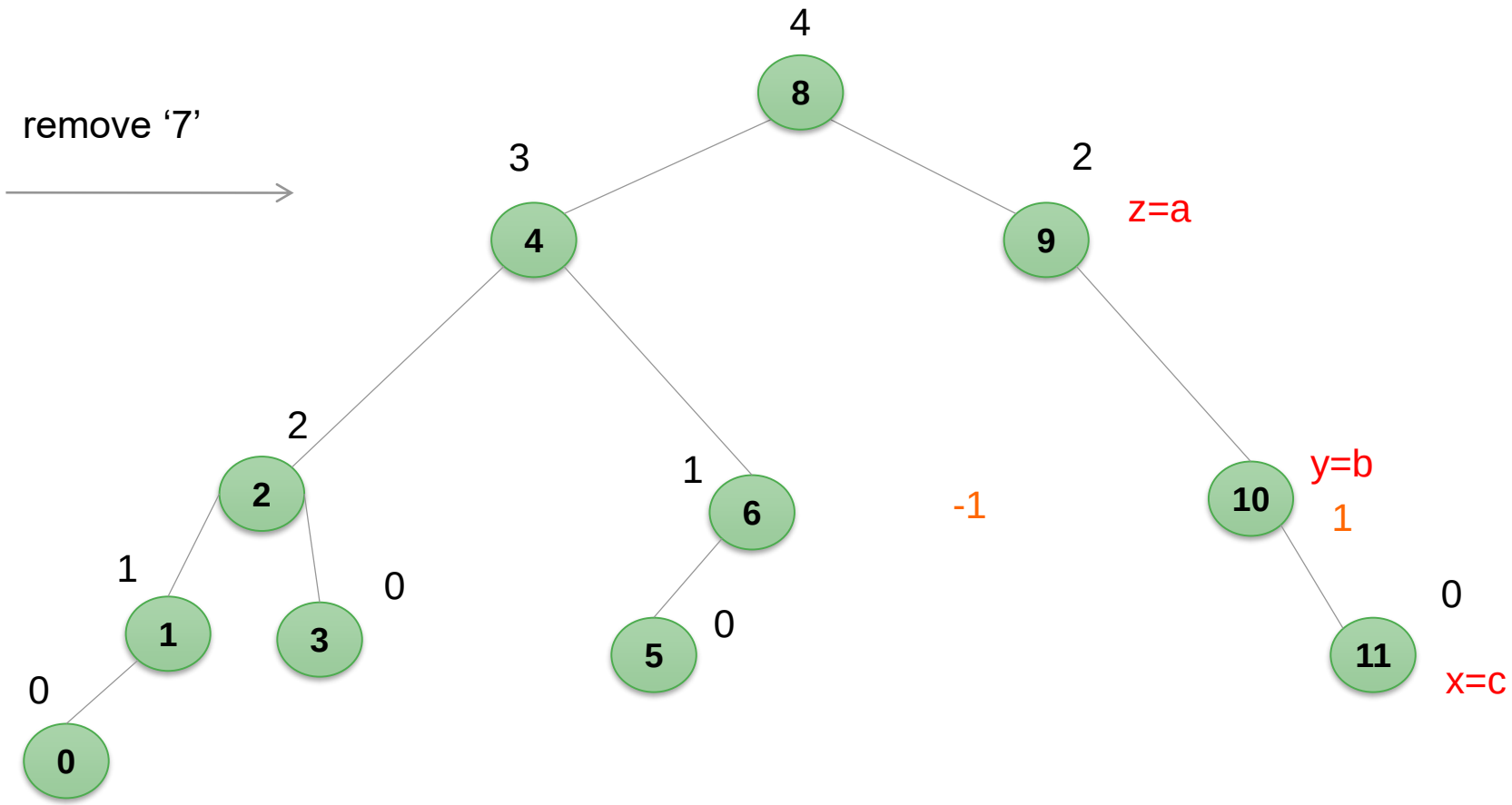
AVL TREES :: REMOVE

Child of y with greatest height = x



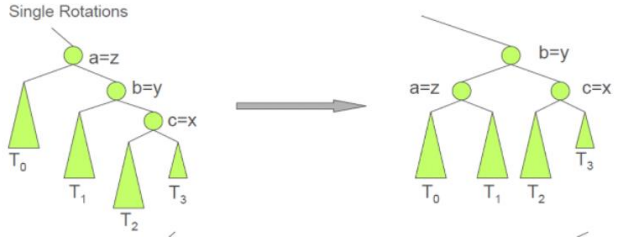
AVL TREES :: REMOVE

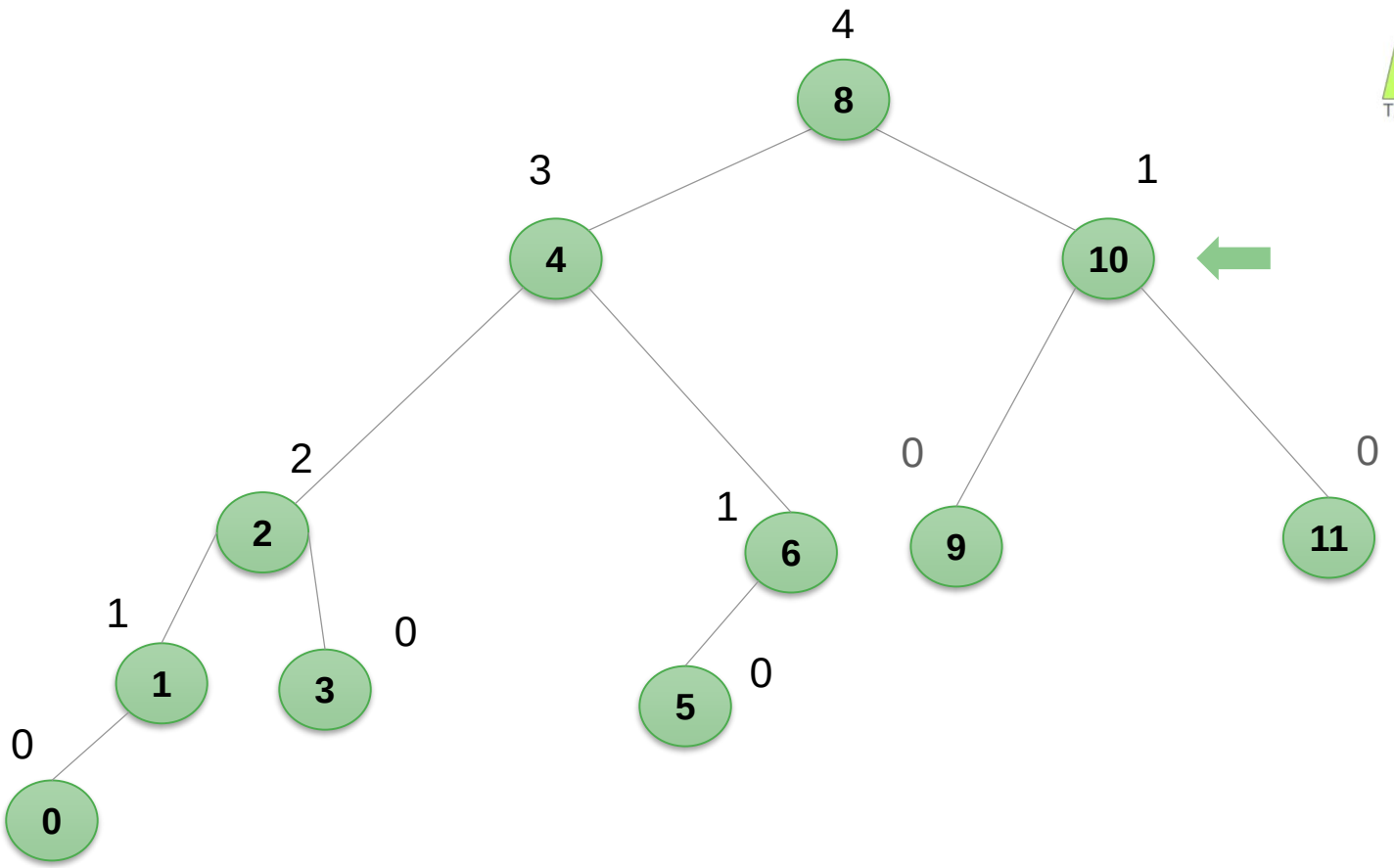
Rename to a, b, c according to inorder traversal



AVL TREES :: REMOVE

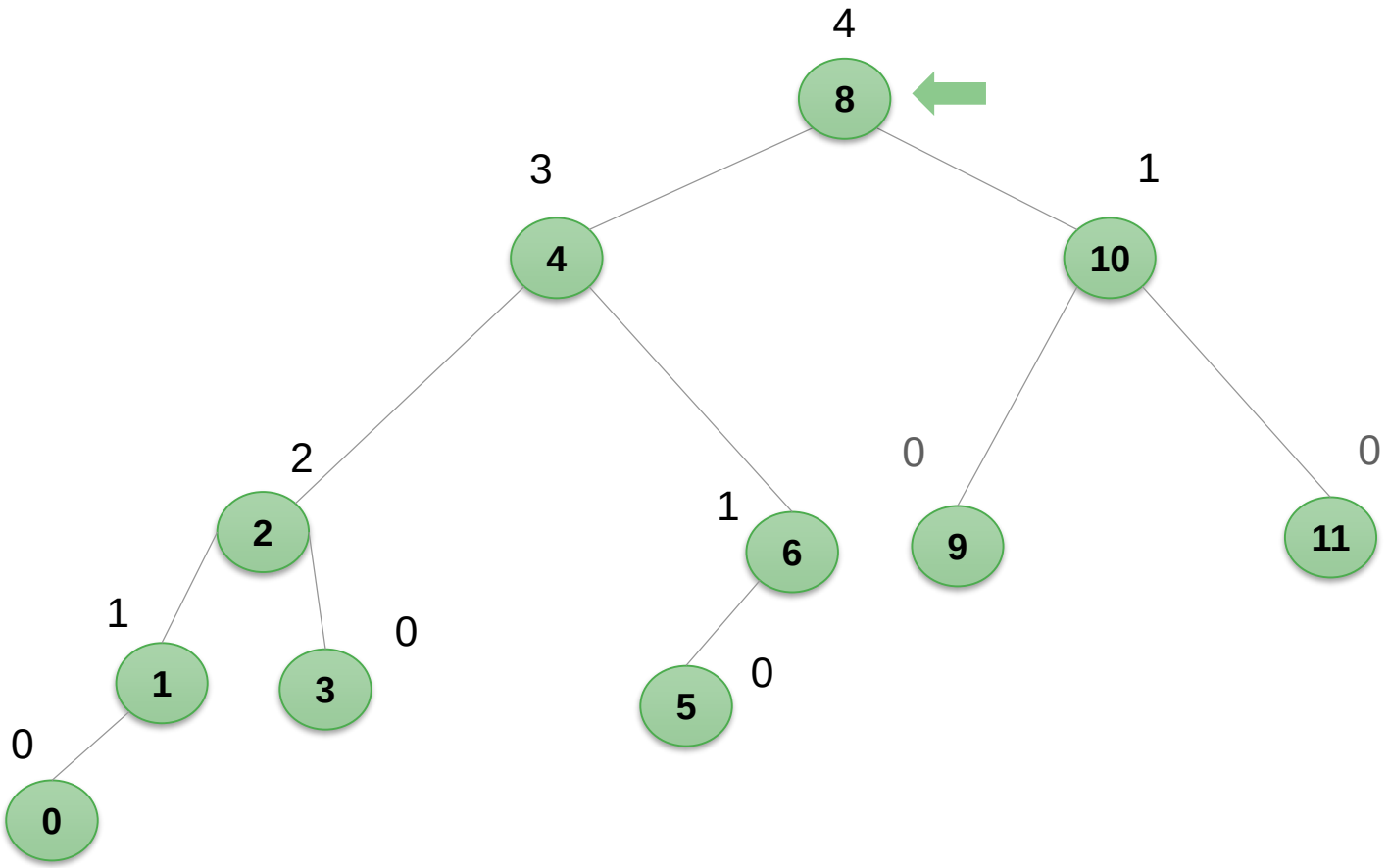
Rotate according to 4 cases



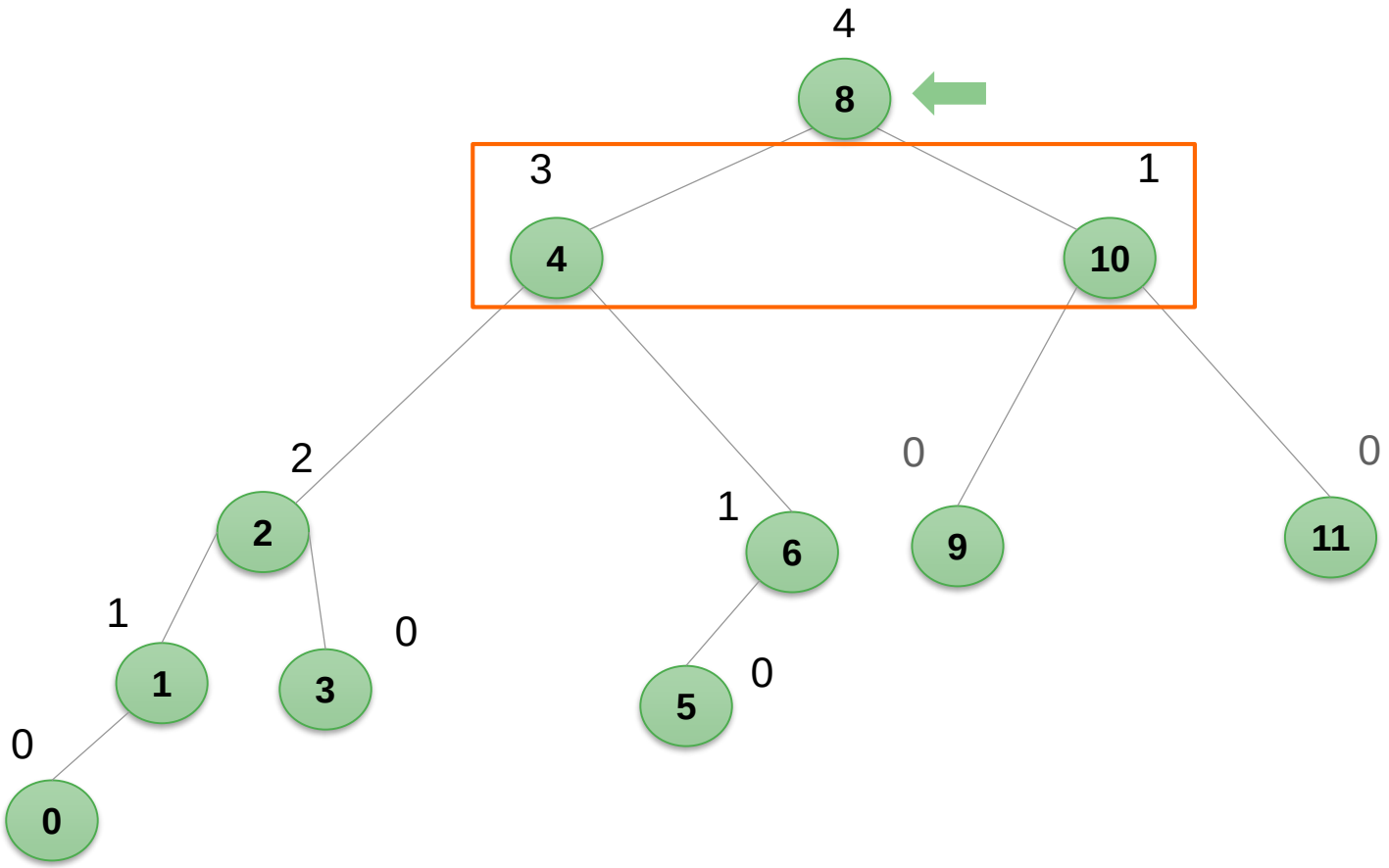


AVL TREES :: REMOVE

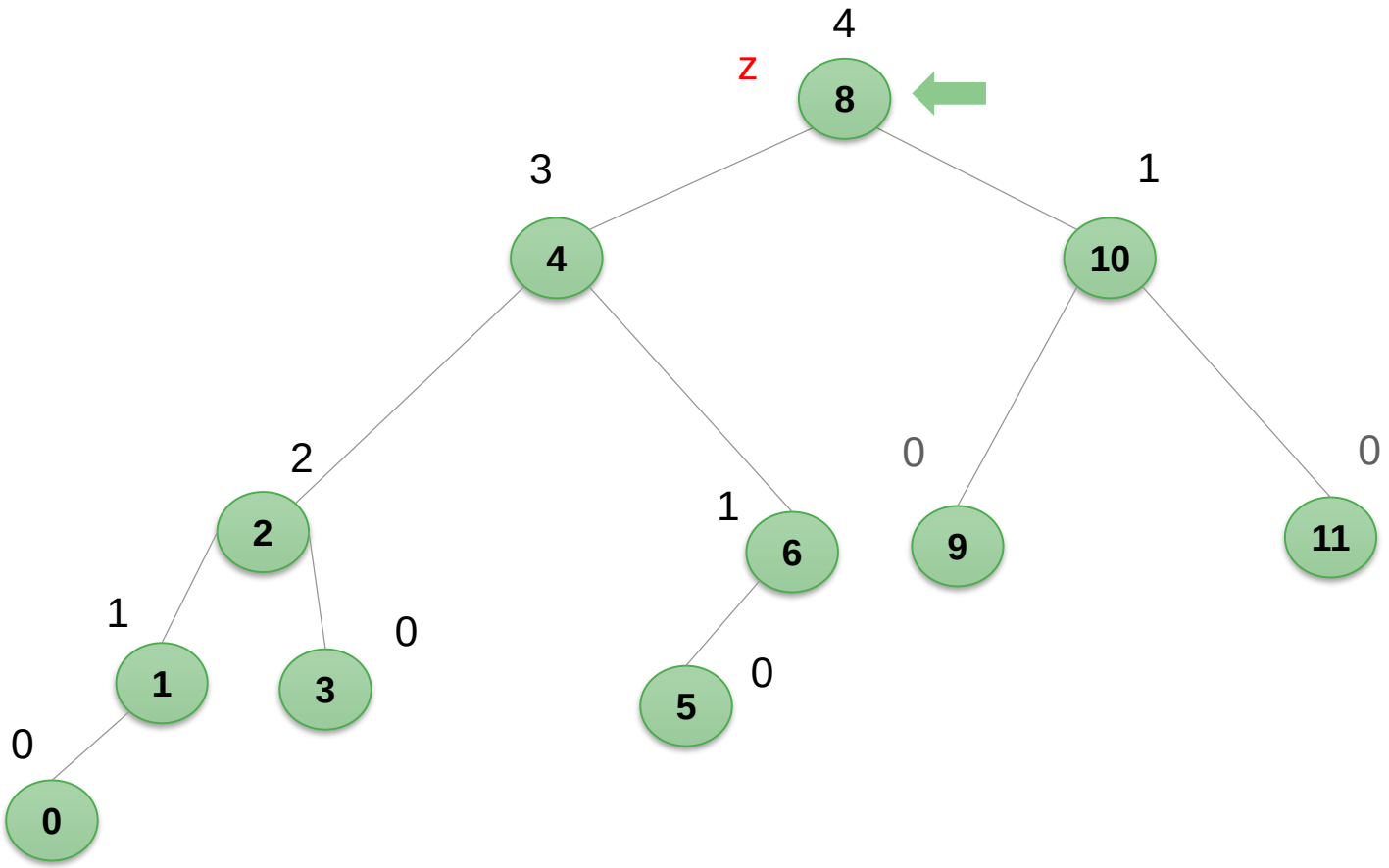
Continue balance check towards root



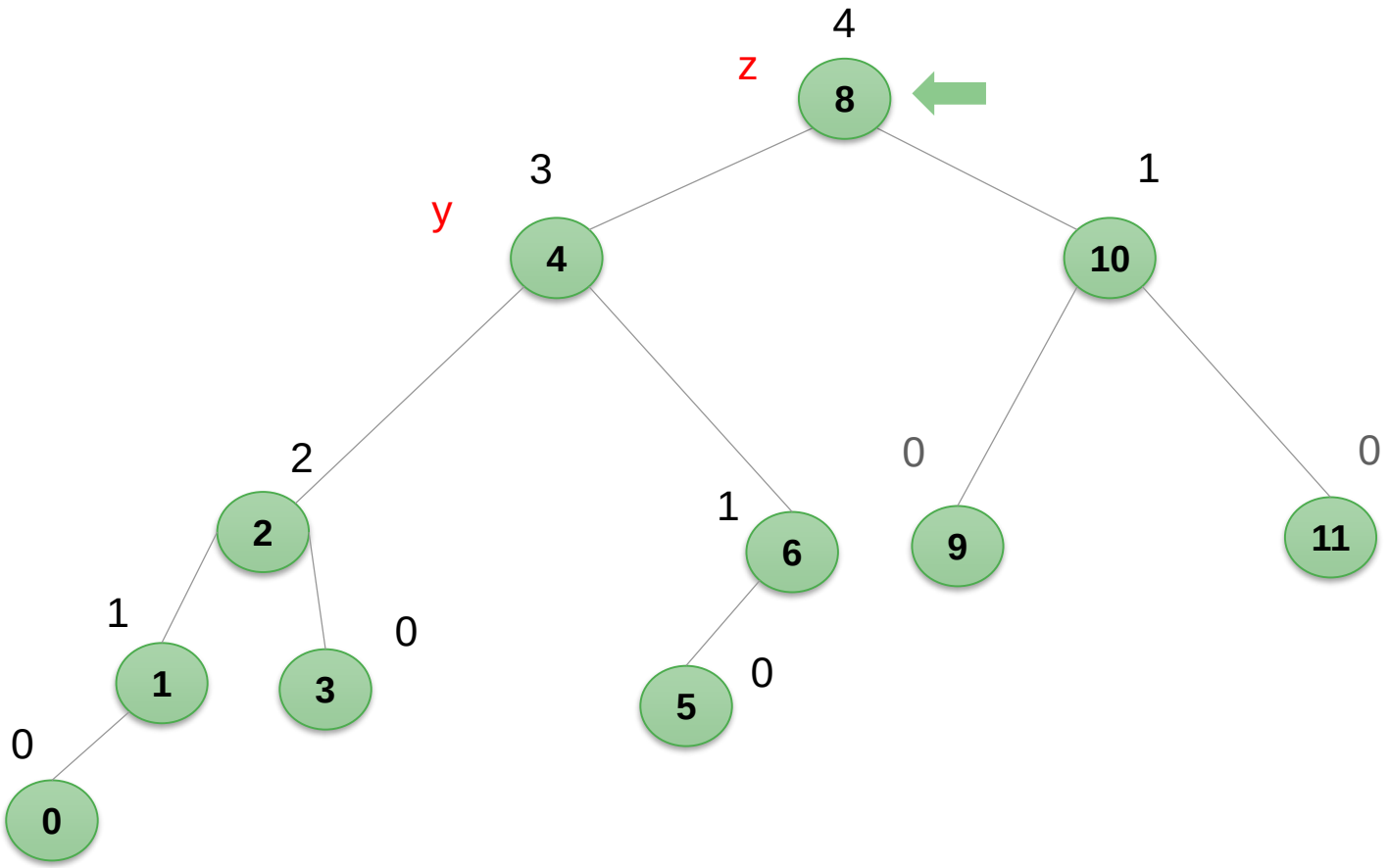
AVL TREES :: REMOVE



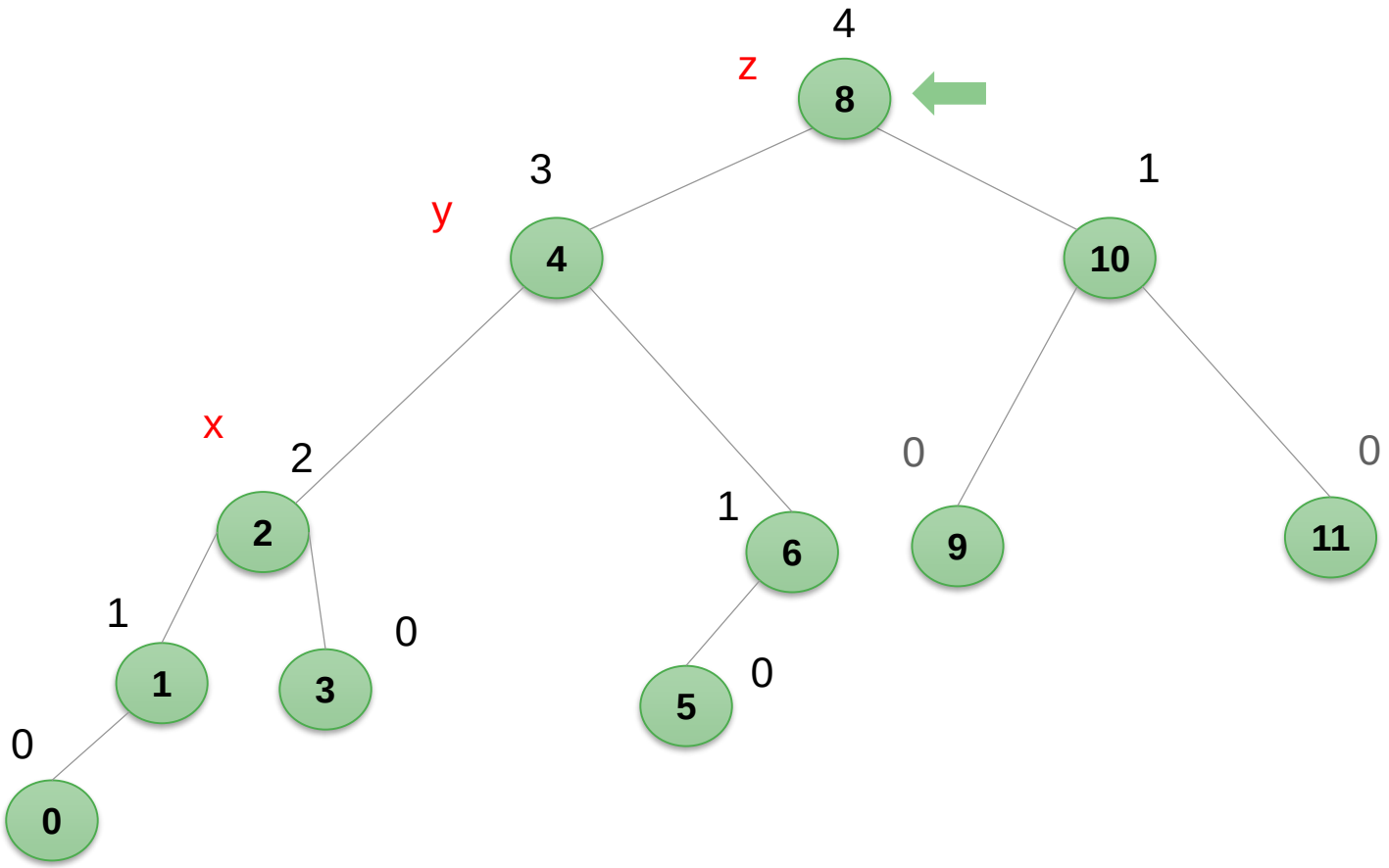
AVL TREES :: REMOVE



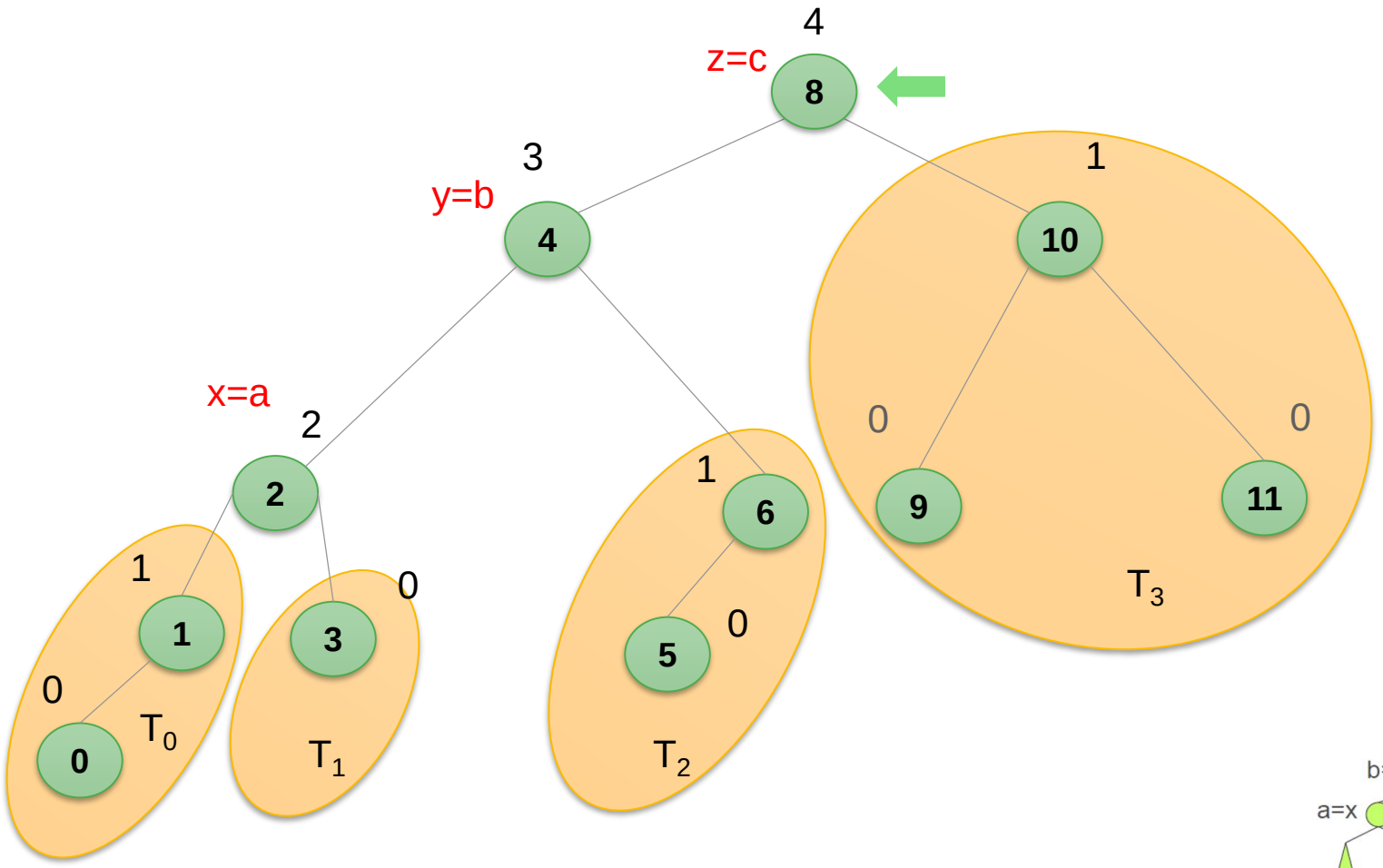
AVL TREES :: REMOVE

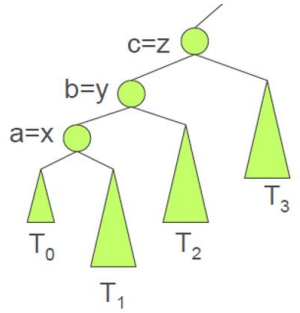


AVL TREES :: REMOVE

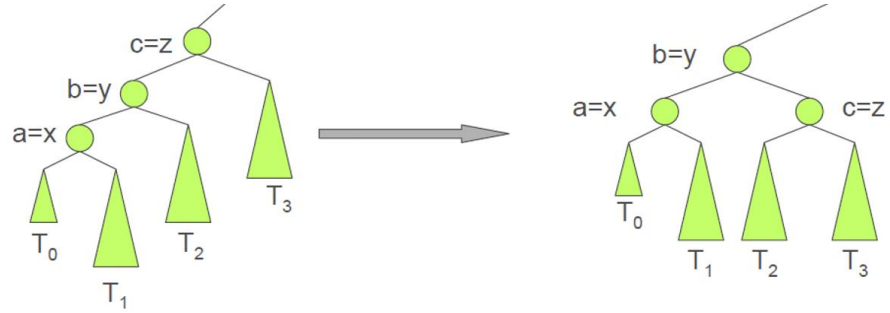


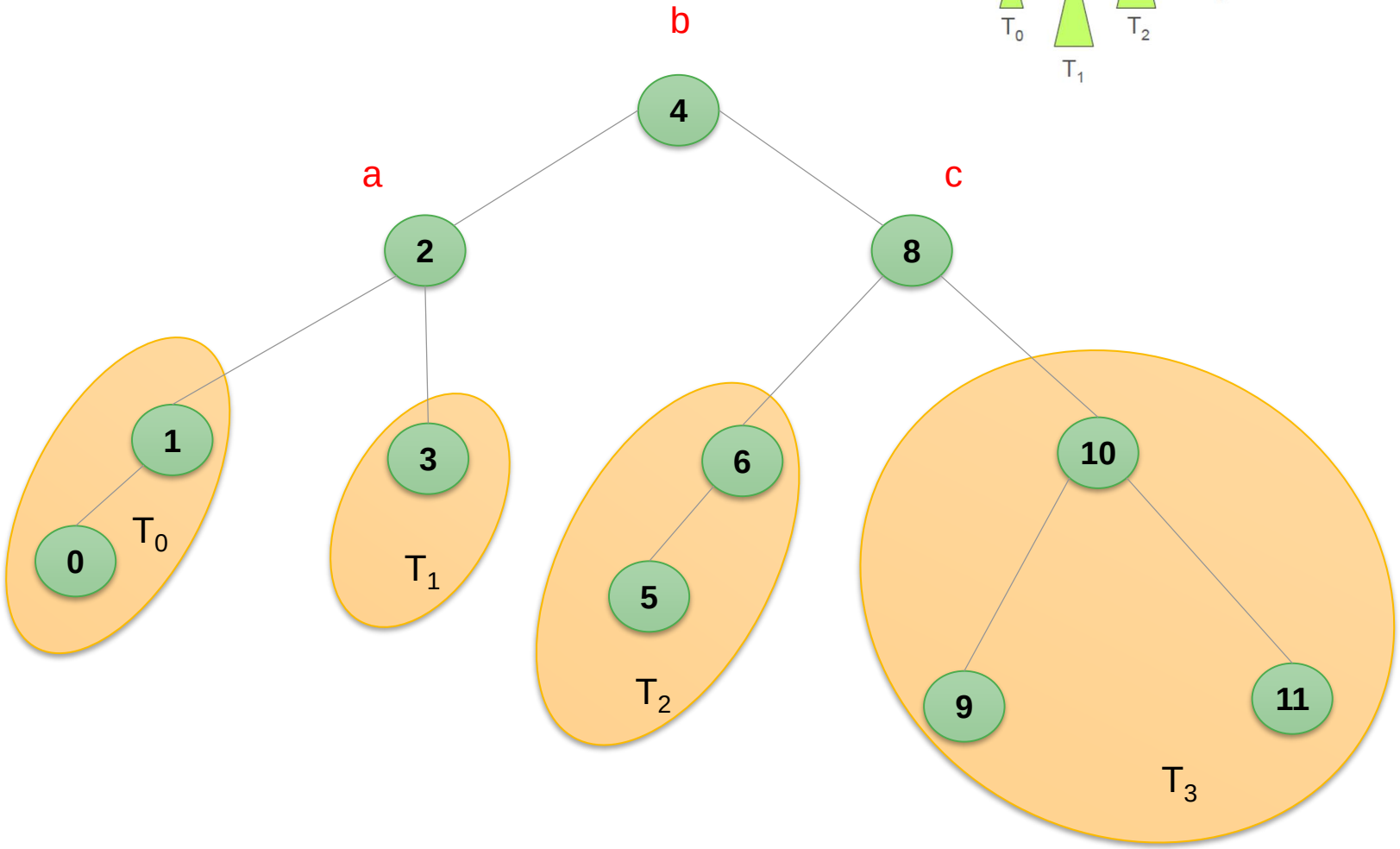
AVL TREES :: REMOVE



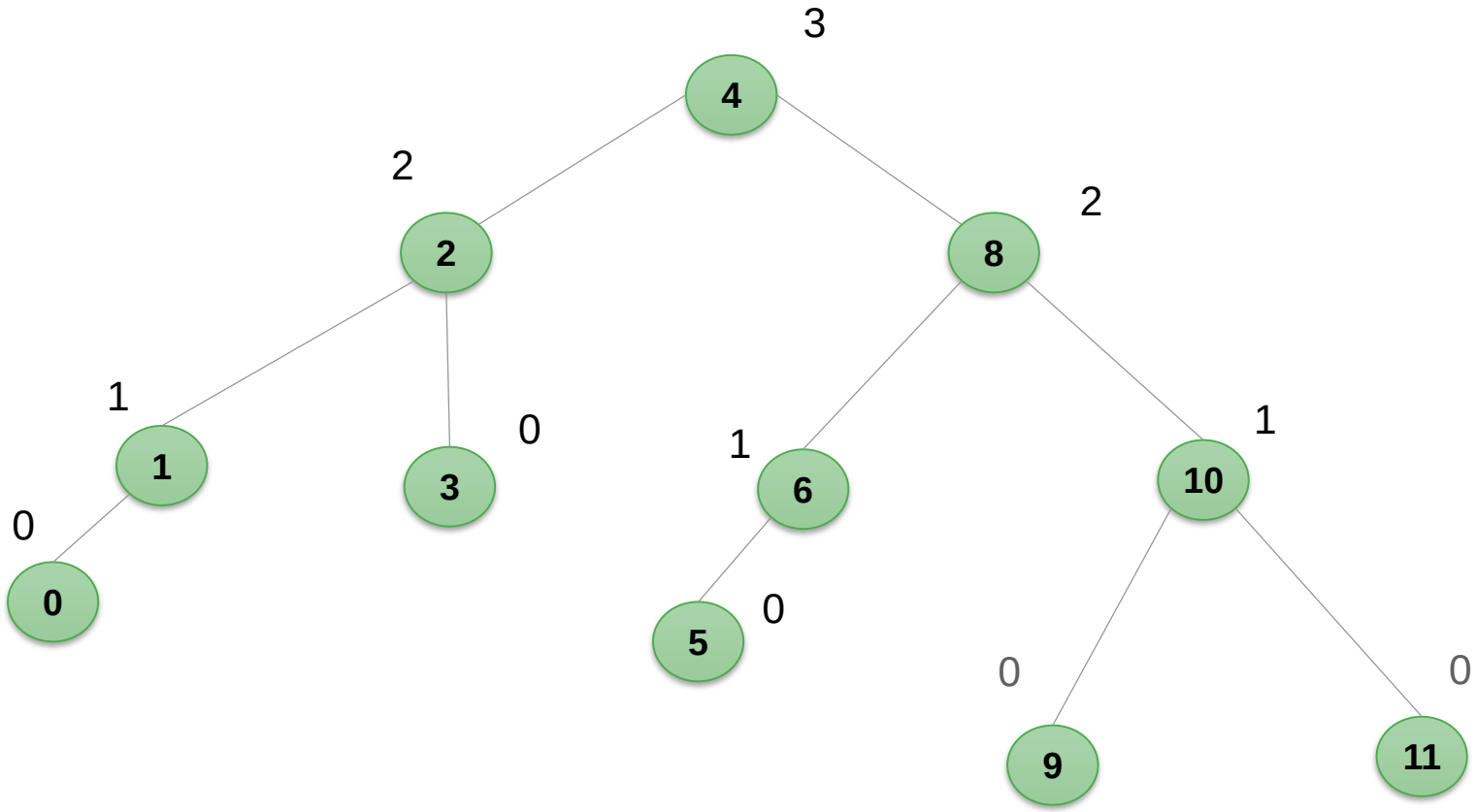


AVL TREES :: REMOVE

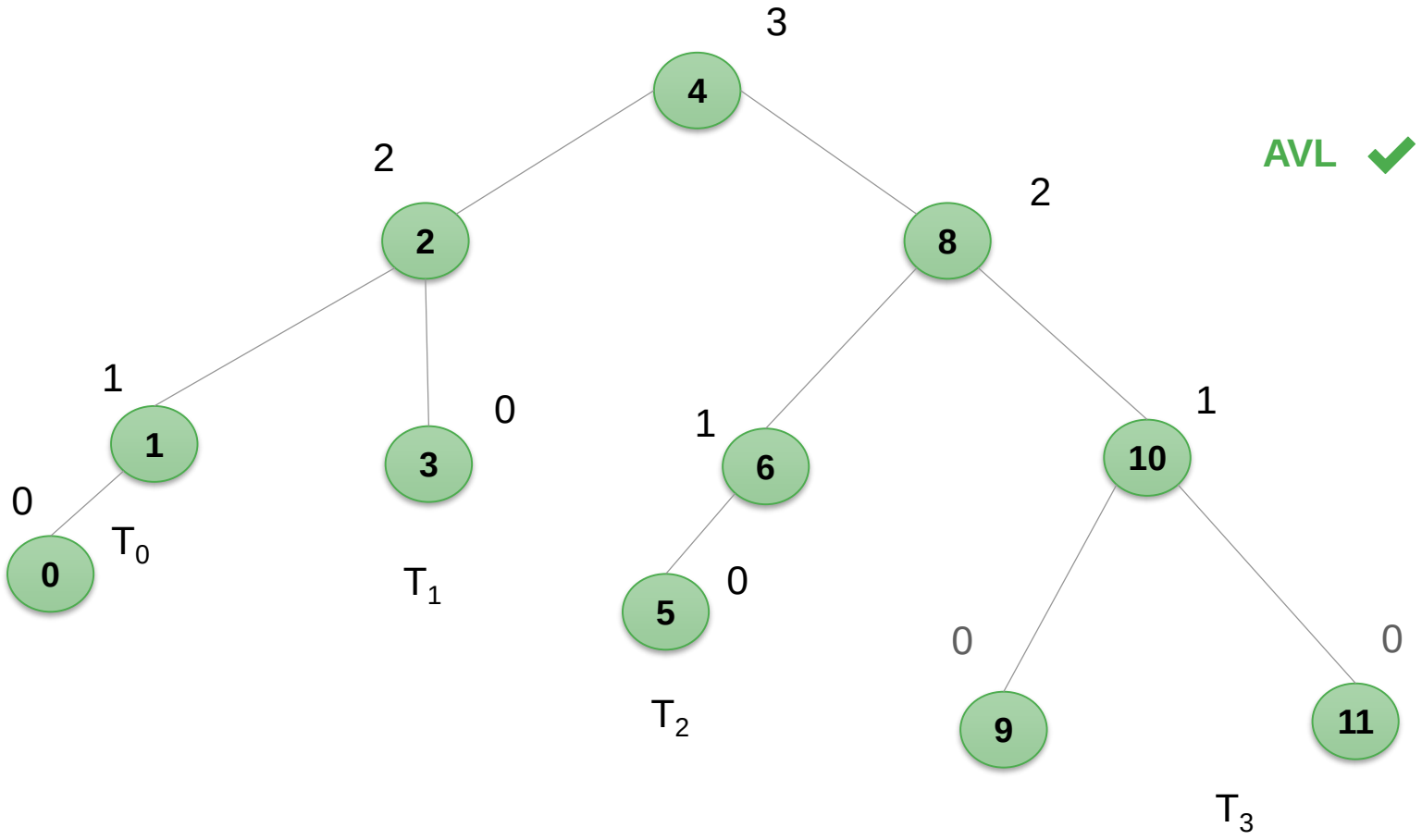




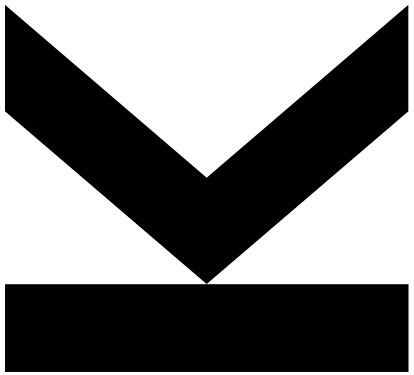
AVL TREES :: REMOVE



AVL TREES :: REMOVE

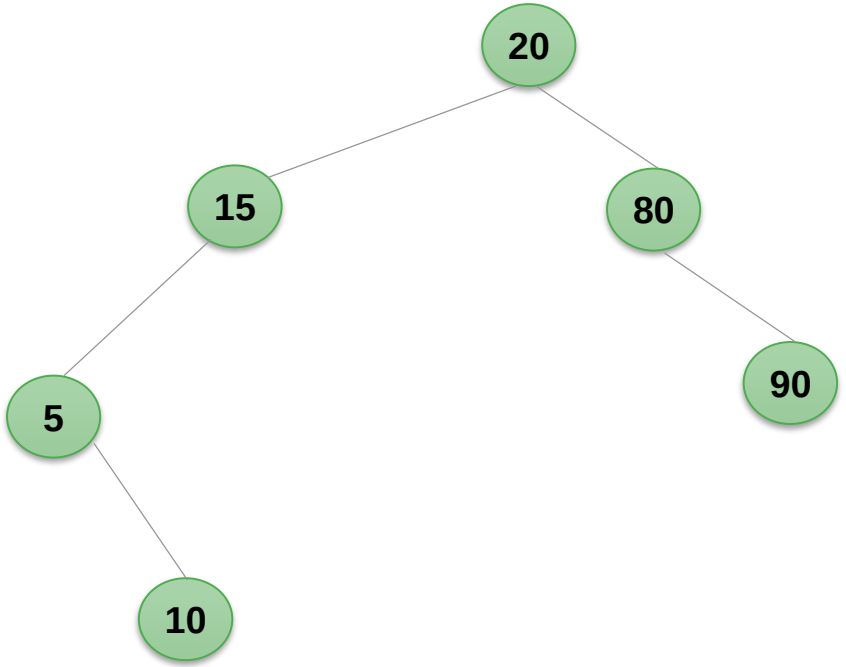


ASSIGNMENT 01



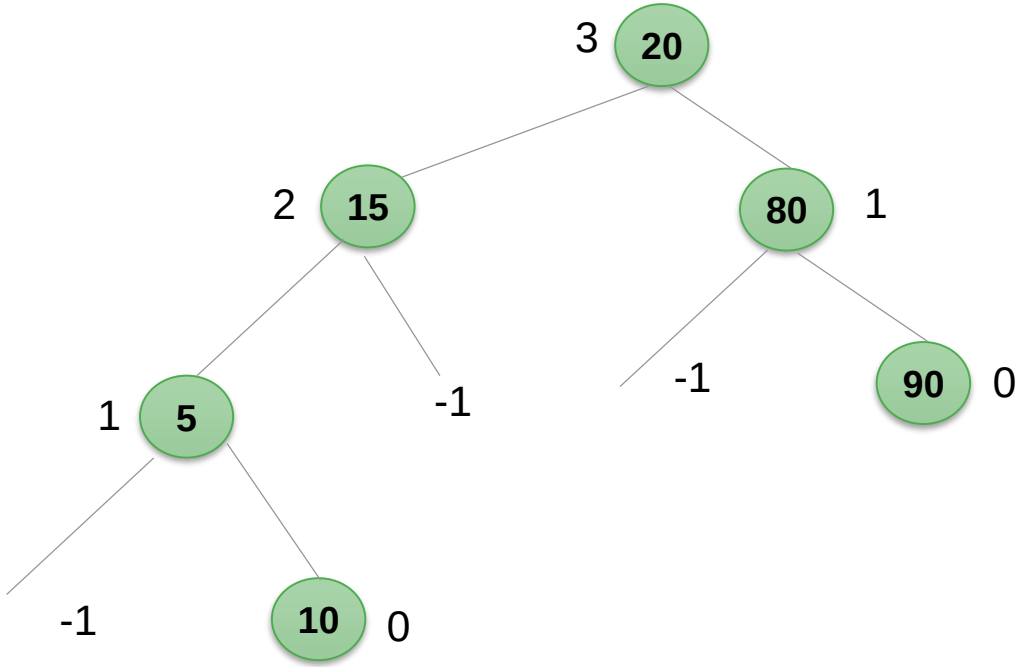
AVL TREES :: DISCUSSION

valid AVL tree?



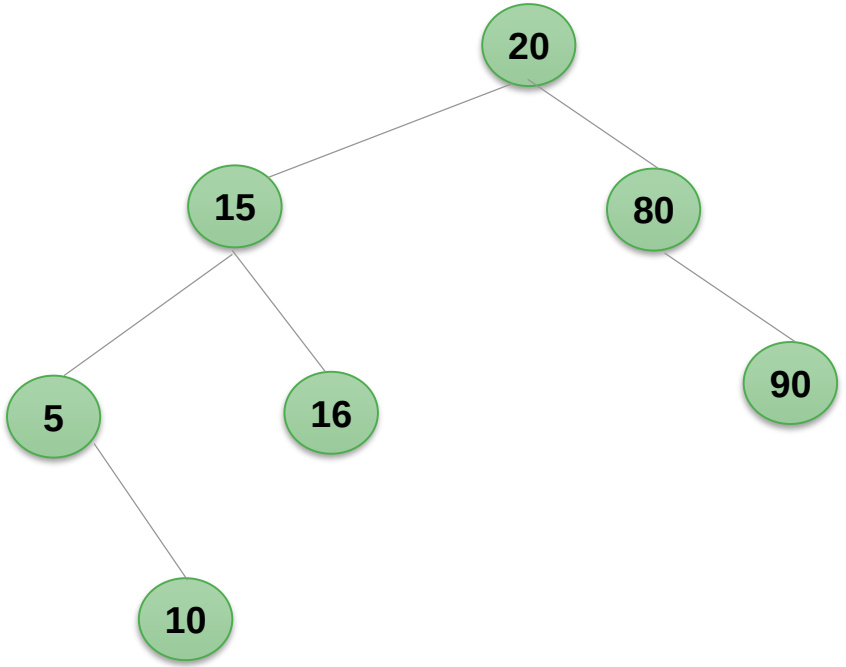
AVL TREES :: DISCUSSION

valid AVL tree?



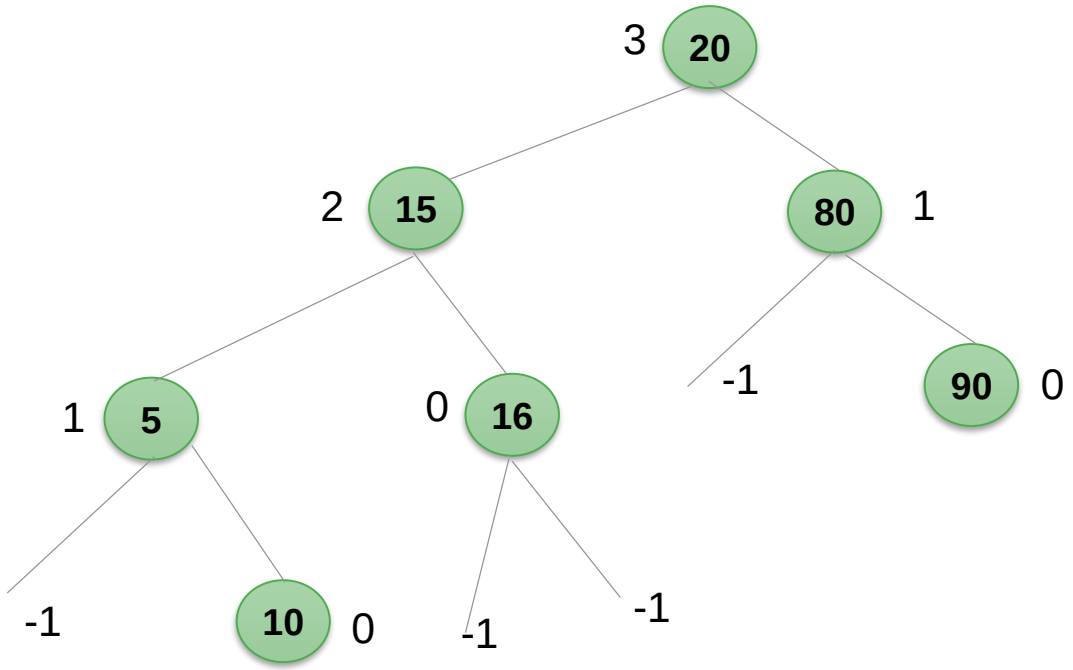
AVL TREES :: DISCUSSION

valid AVL tree?



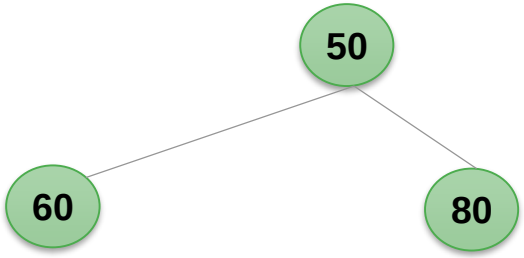
AVL TREES :: DISCUSSION

valid AVL tree?



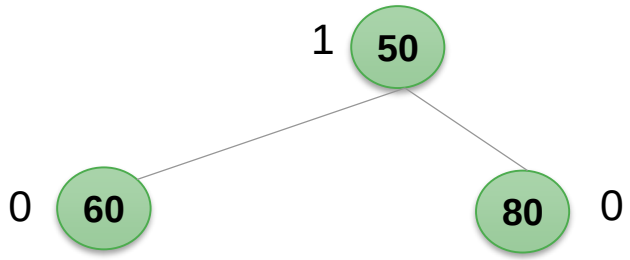
AVL TREES :: DISCUSSION

valid AVL tree?



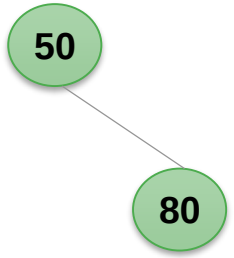
AVL TREES :: DISCUSSION

valid AVL tree?



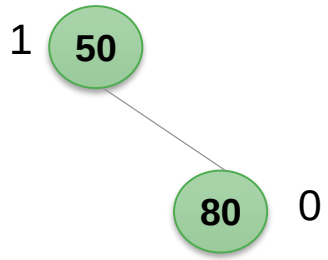
AVL TREES :: DISCUSSION

valid AVL tree?



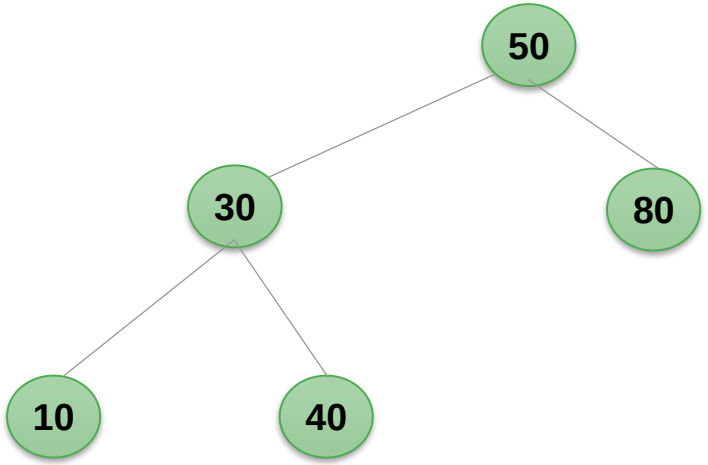
AVL TREES :: DISCUSSION

valid AVL tree?



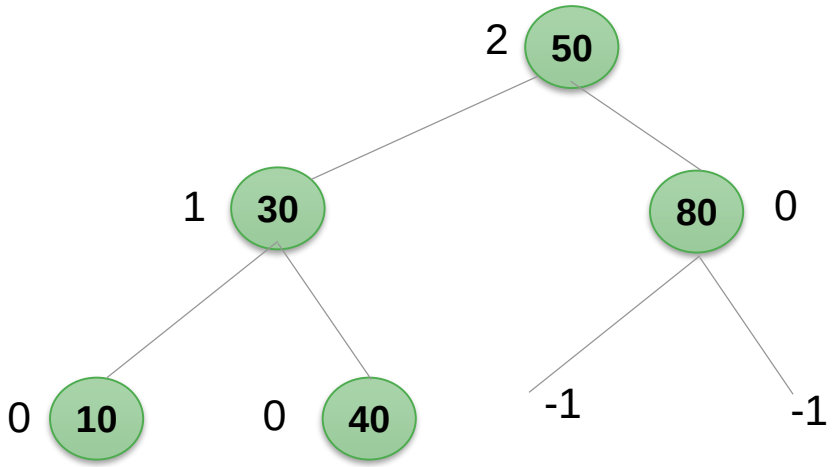
AVL TREES :: DISCUSSION

valid AVL tree?



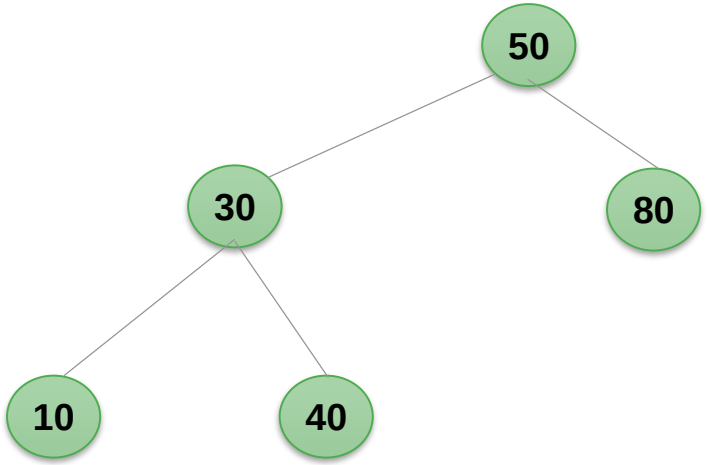
AVL TREES :: DISCUSSION

valid AVL tree?



AVL TREES :: DISCUSSION

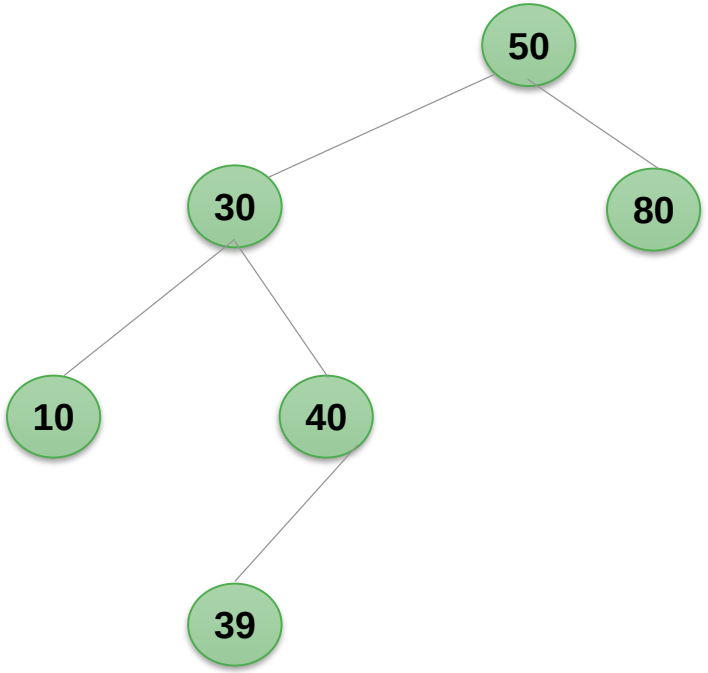
insert(39)



AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

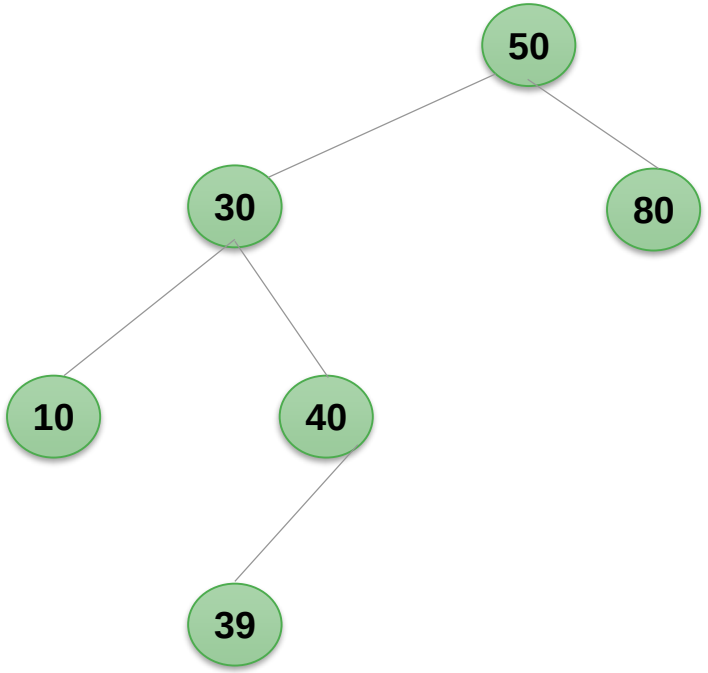


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> update heights

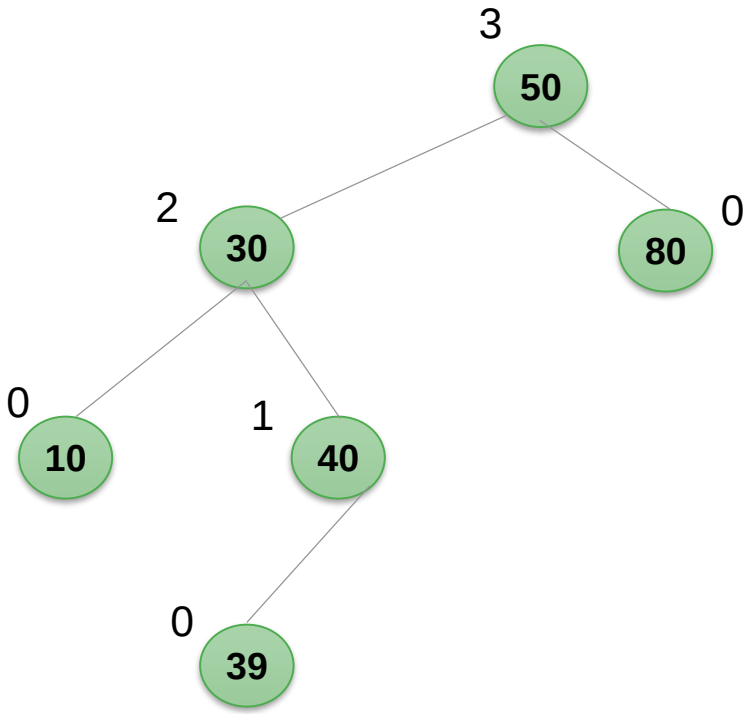


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> update heights

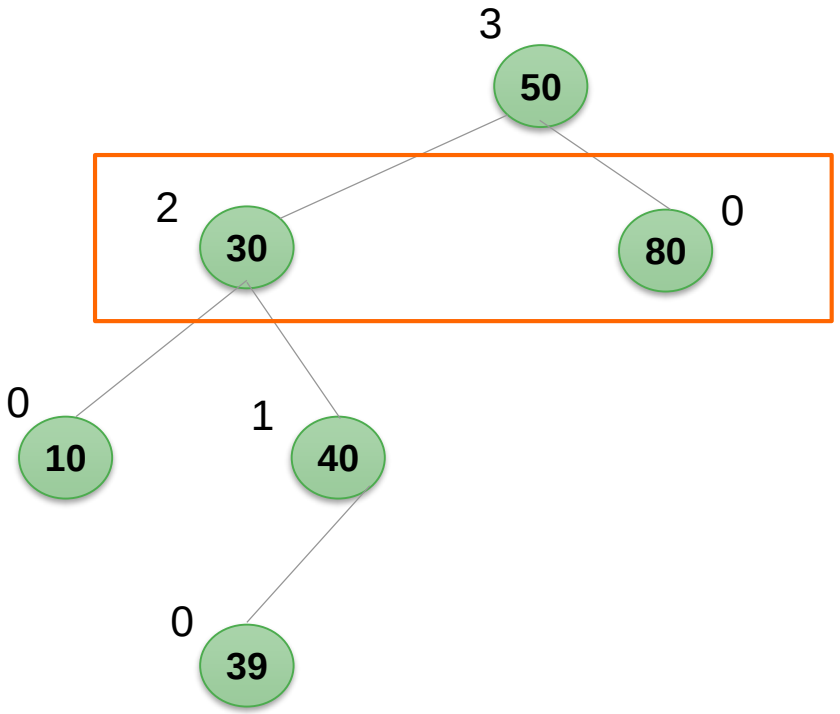


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> update heights

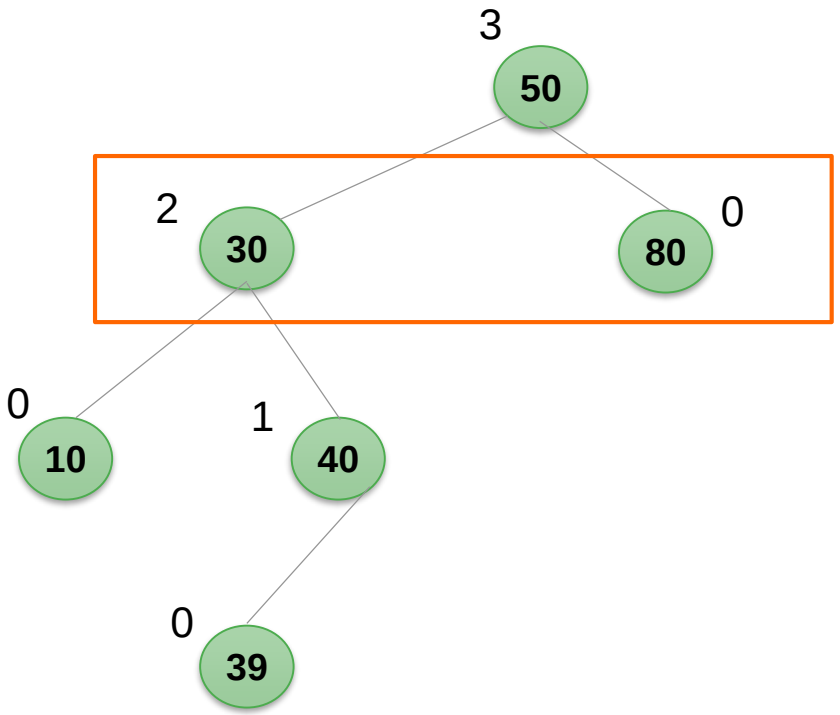


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance

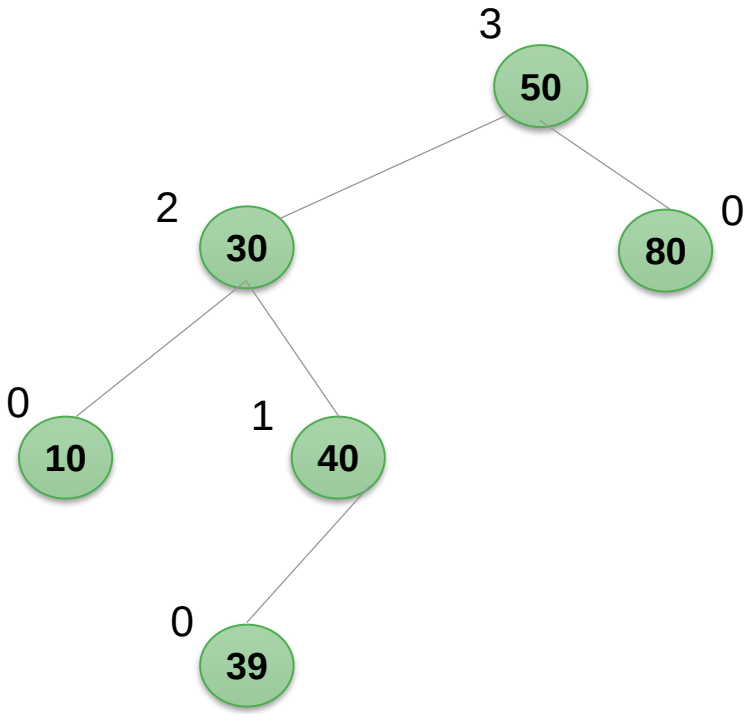


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



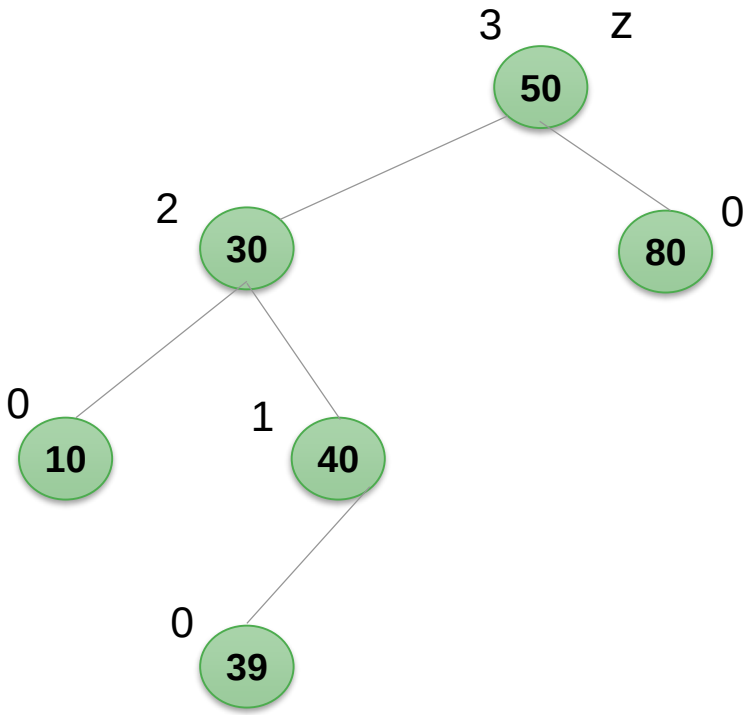
Go up from the new node in the tree until the first node **x** is found, whose grandparent **z** is an unbalanced node

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



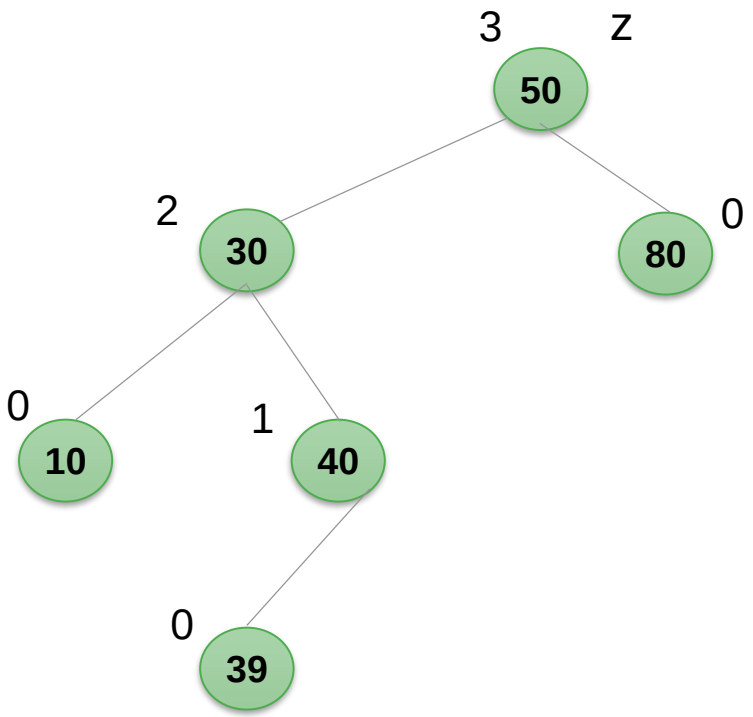
Go up from the new node in the tree until the first node **x** is found, whose grandparent **z** is an unbalanced node

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



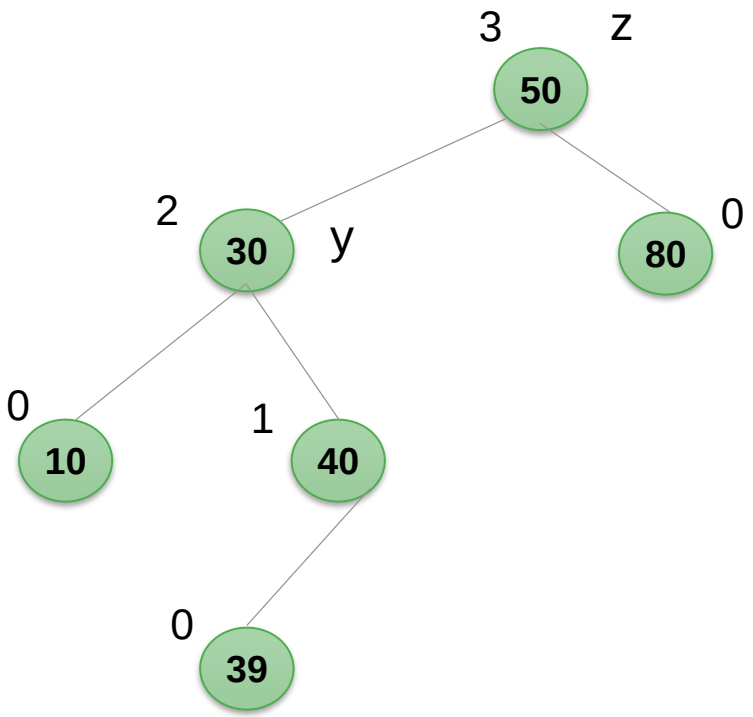
Define **y** as child of **z** (= the node we passed on the way to z);

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



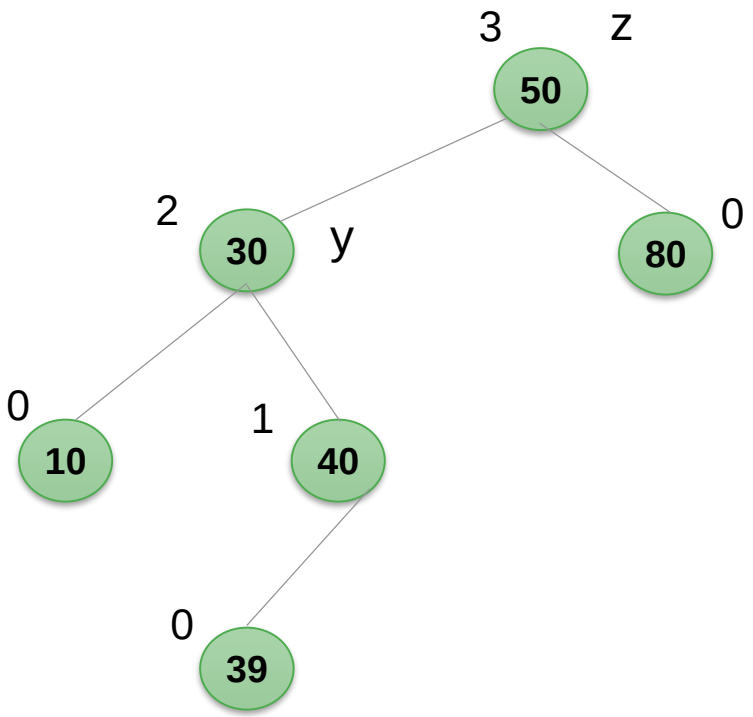
Define **y** as child of **z** (= the node we passed on the way to z);

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



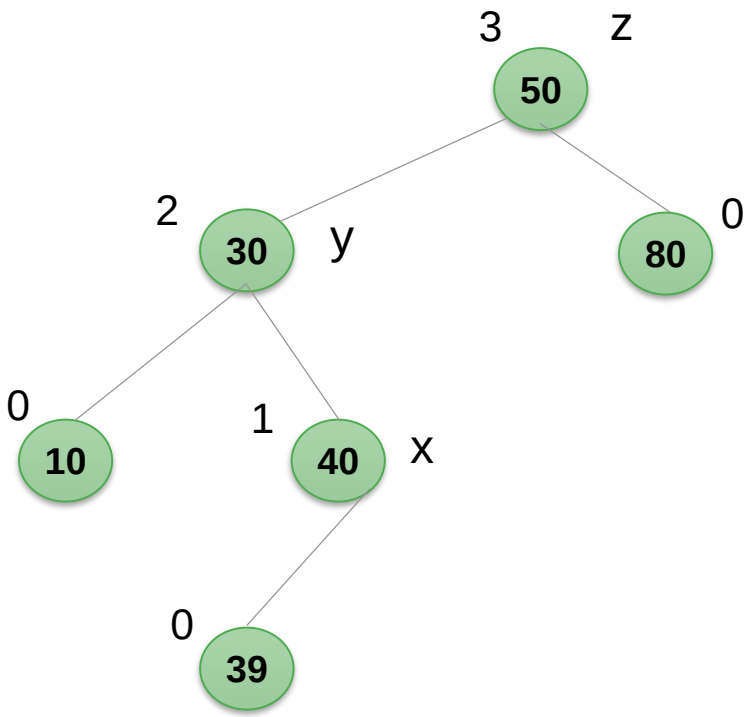
Define **x** as child of **y**

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



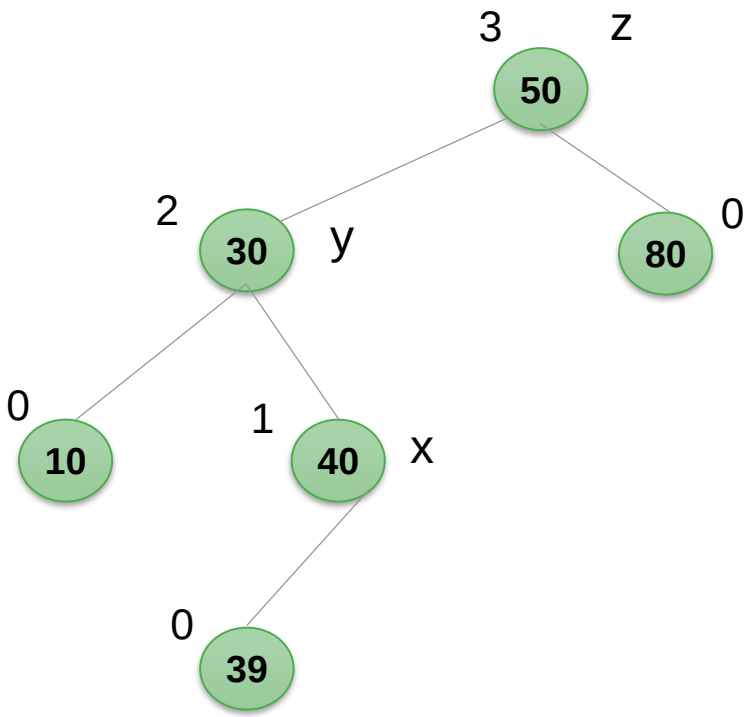
Define **x** as child of **y**

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



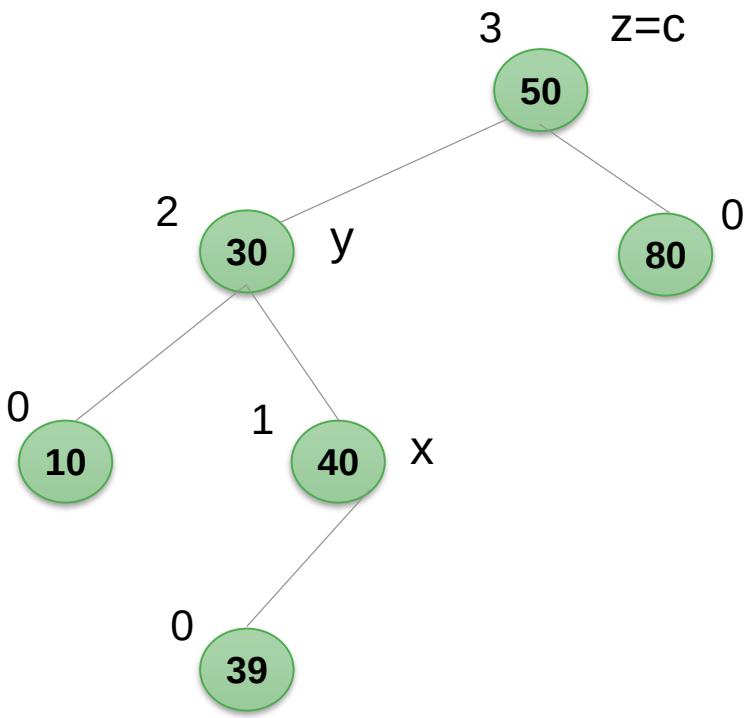
Rename **x**,**y**,**z** in **a**,**b**,**c** (according to Inorder traversal!)

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



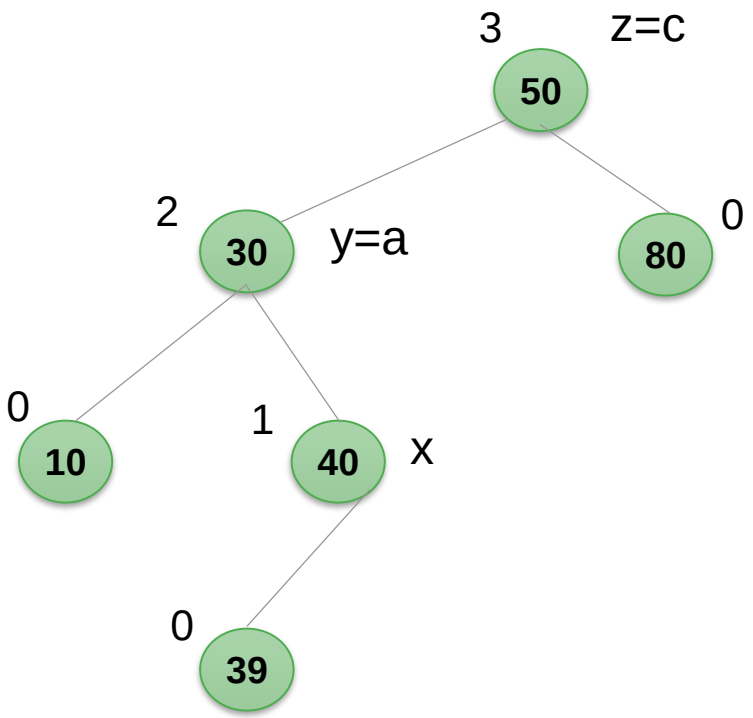
Rename **x**,**y**,**z** in **a**,**b**,**c** (according to Inorder traversal!)

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



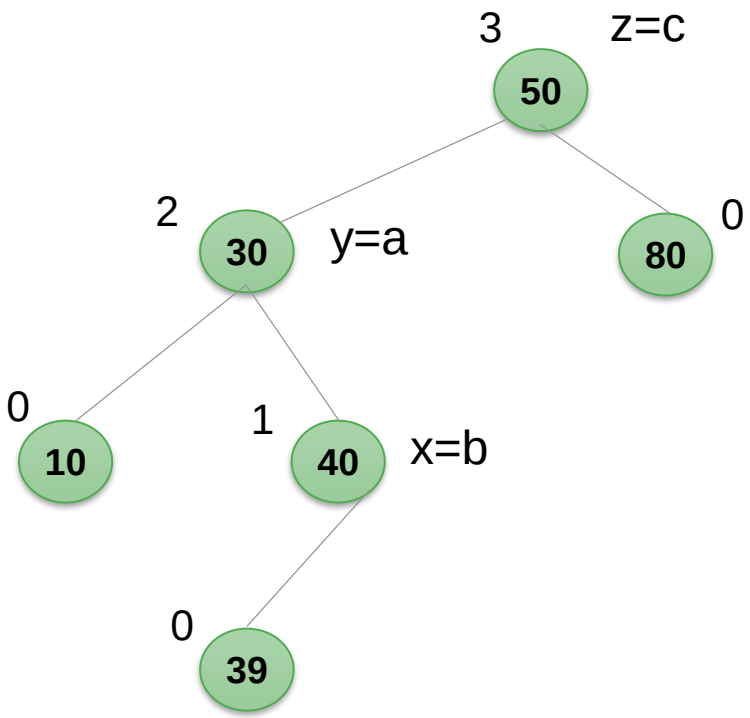
Rename **x**,**y**,**z** in **a**,**b**,**c** (according to Inorder traversal!)

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



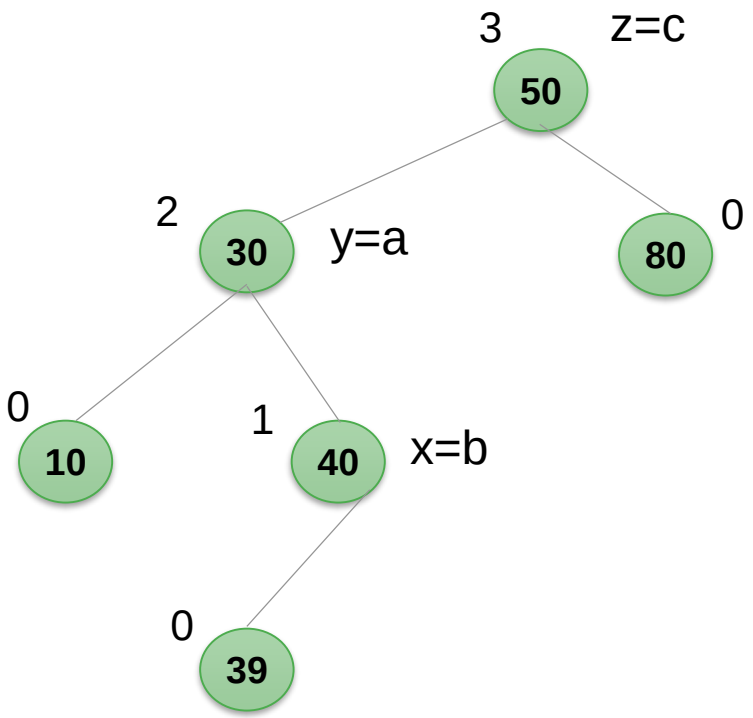
Rename **x**,**y**,**z** in **a**,**b**,**c** (according to Inorder traversal!)

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



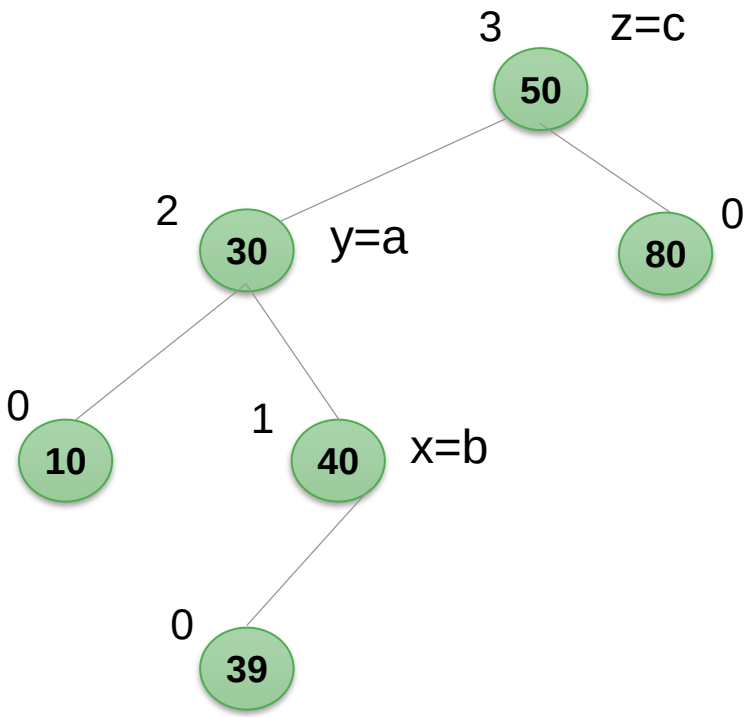
Assign subtrees T0...T3 based on 4 cases

AVL TREES :: DISCUSSION

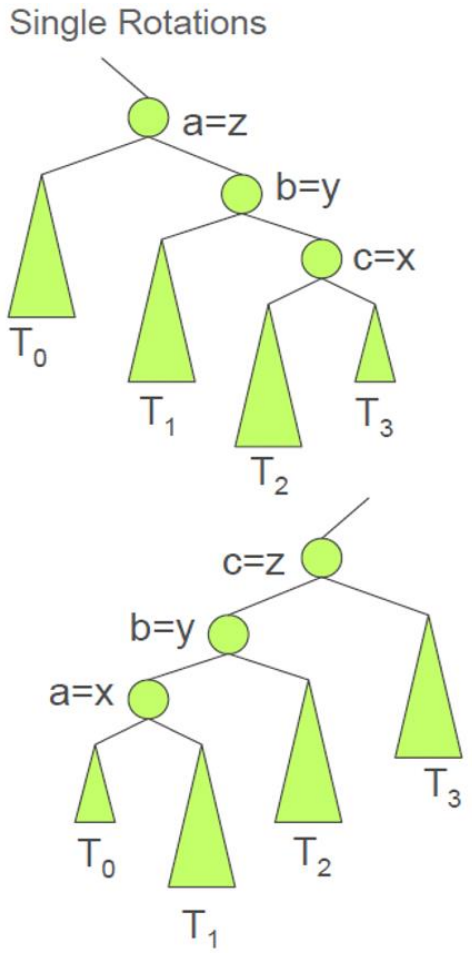
insert(39)

valid AVL tree?

-> rebalance



Assign subtrees T0...T3 based on 4 cases

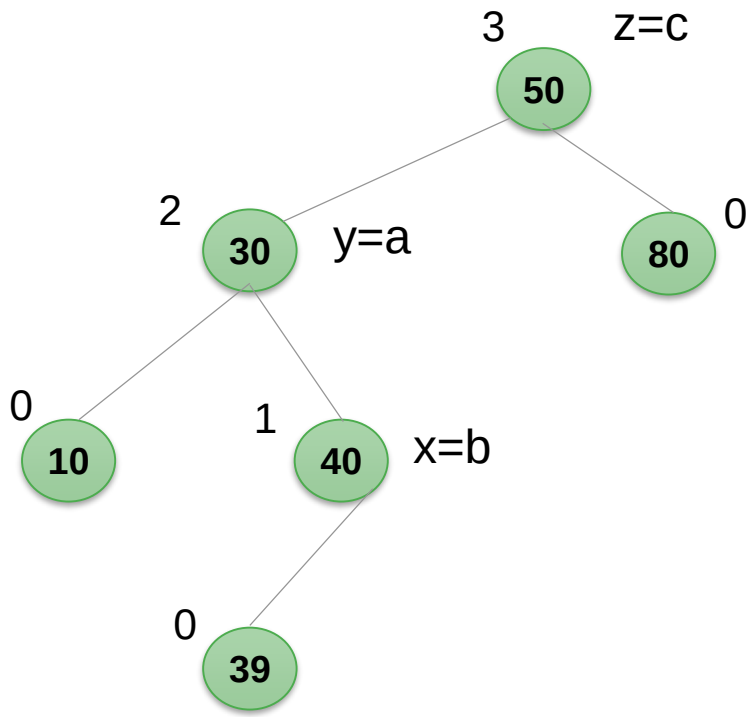


AVL TREES :: DISCUSSION

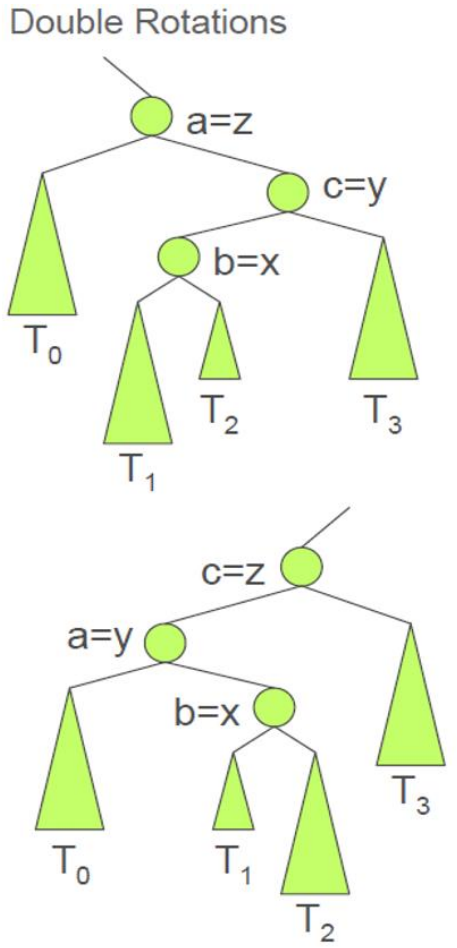
insert(39)

valid AVL tree?

-> rebalance



Assign subtrees T0...T3 based on 4 cases

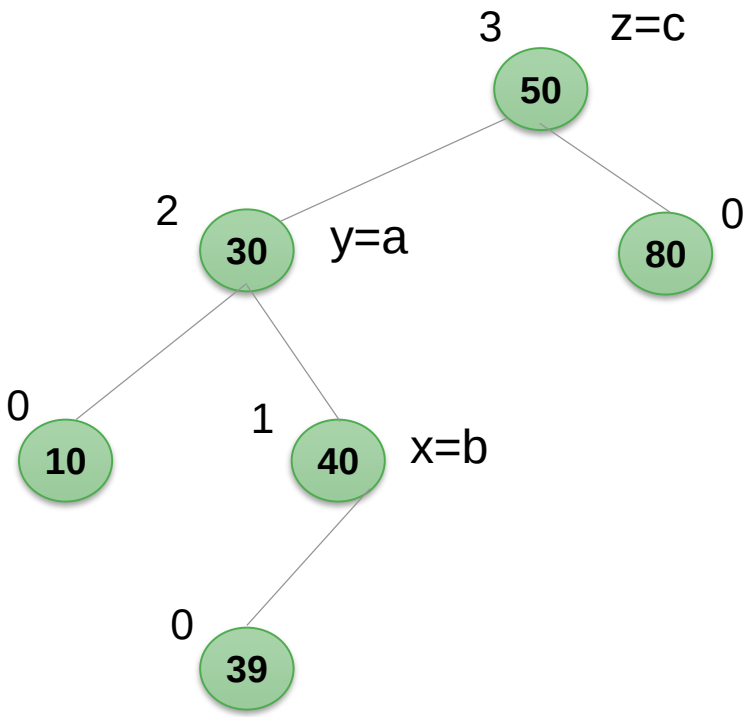


AVL TREES :: DISCUSSION

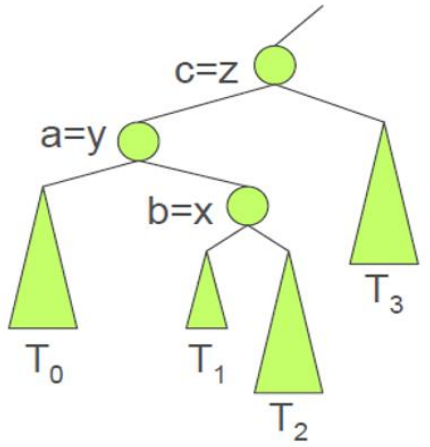
insert(39)

valid AVL tree?

-> rebalance



Assign subtrees T0...T3 based on 4 cases

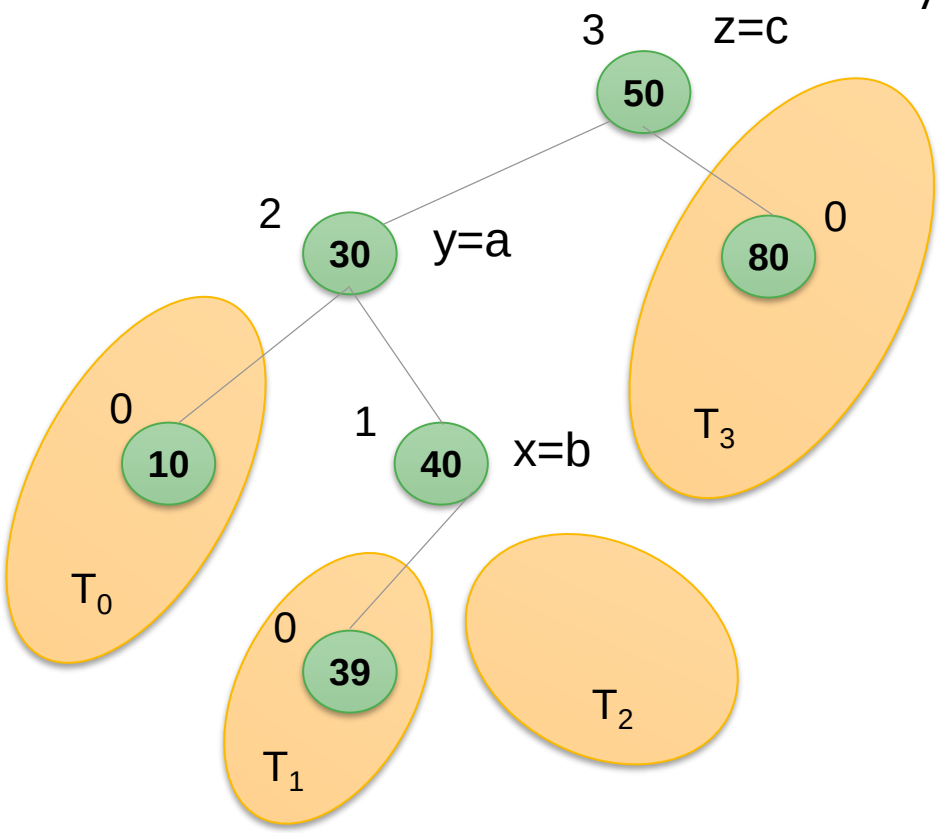


AVL TREES :: DISCUSSION

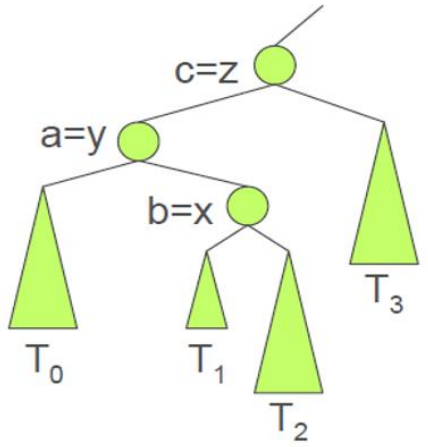
insert(39)

valid AVL tree?

-> rebalance



Assign subtrees T0...T3 based on 4 cases

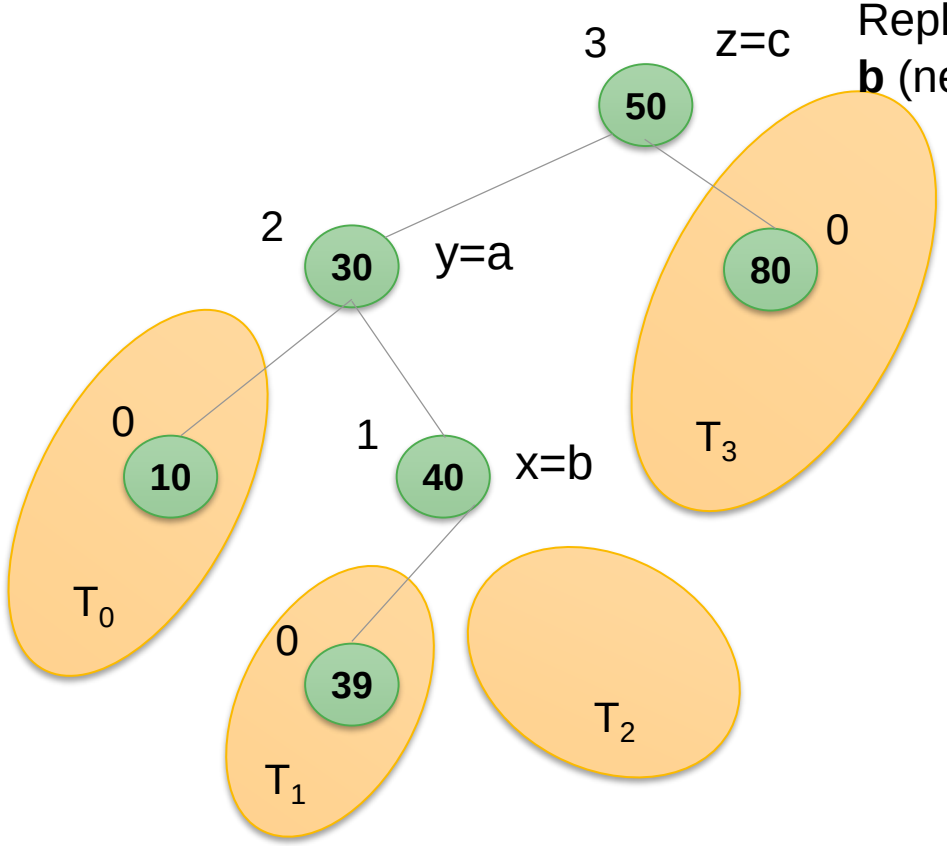


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



Replace **z** (old subroot of unsorted part-tree) by **b** (new subroot of sorted part-tree)

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance

x=b

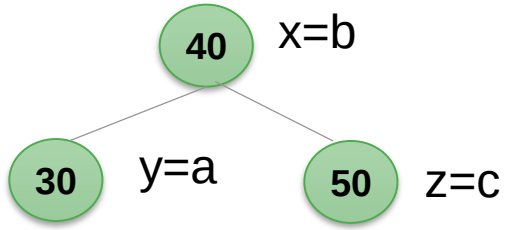
Replace **z** (old subroot of unsorted part-tree) by **b** (new subroot of sorted part-tree)

AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

-> rebalance



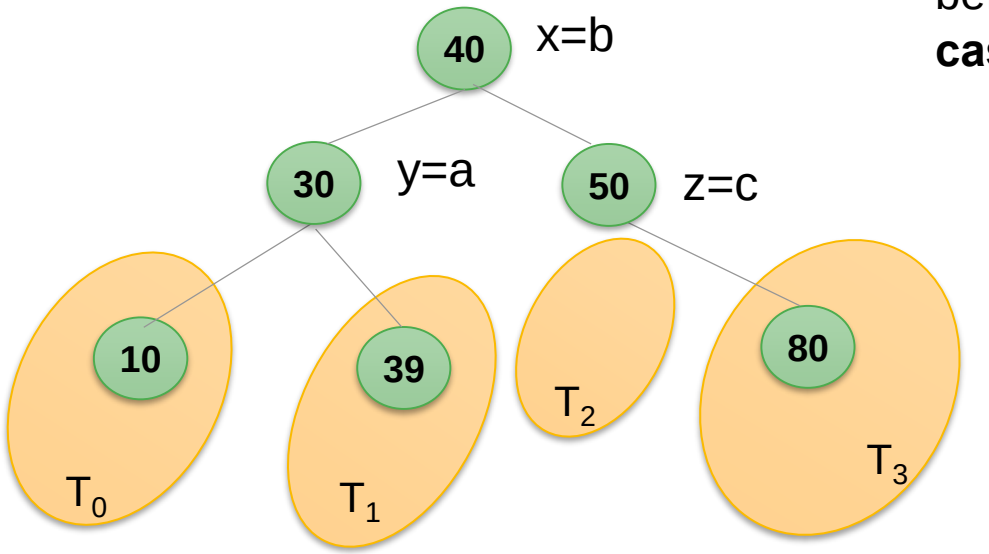
Children of **b** are now **a** (left) and **c** (right)

AVL TREES :: DISCUSSION

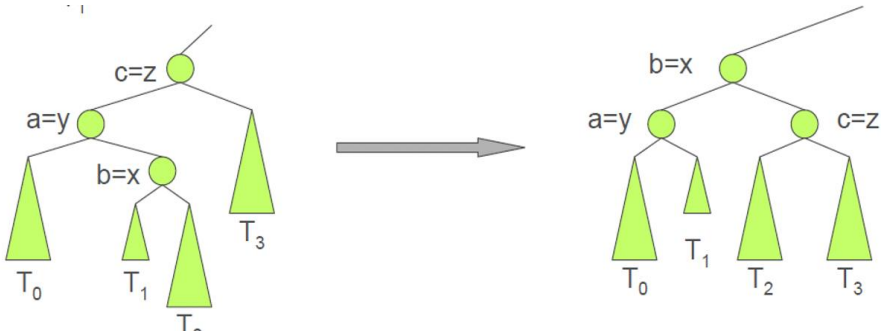
insert(39)

valid AVL tree?

-> rebalance



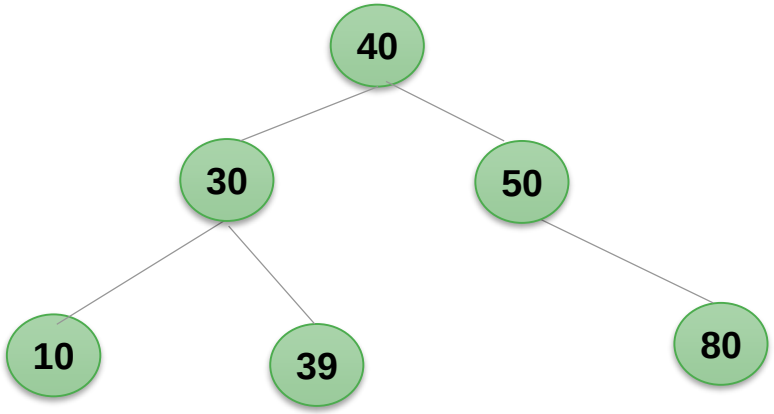
1. Children of **a** and **c** are the subtrees T_0 … T_3, which have been children of **x**, **y** and **z** before → reassign and distinguish **4 cases…**



AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

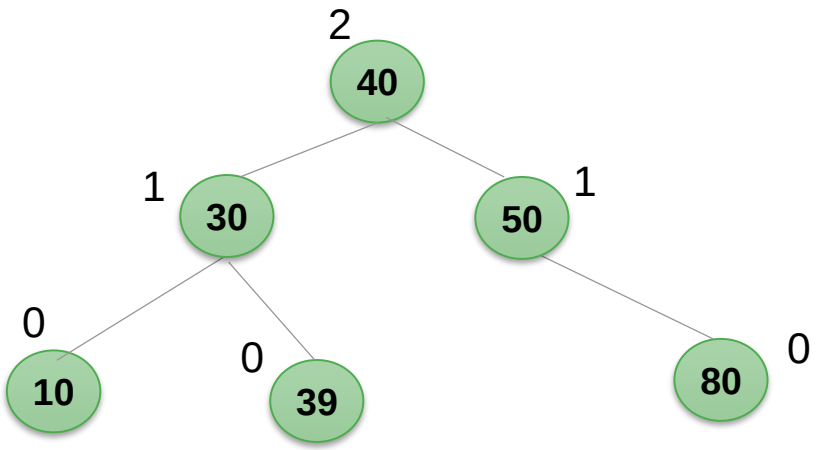


AVL TREES :: DISCUSSION

insert(39)

valid AVL tree?

check heights



JOHANNES KEPLER
UNIVERSITY LINZ

FAST SEARCHING / BALANCED TREES

Martin Schobesberger, Markus Weninger,
Markus Jäger, Florian Beck, Achref Rihani

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

Algorithms and Data Structures 2
Exercise – 2023W

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

JOHANNES KEPLER
UNIVERSITY LINZ