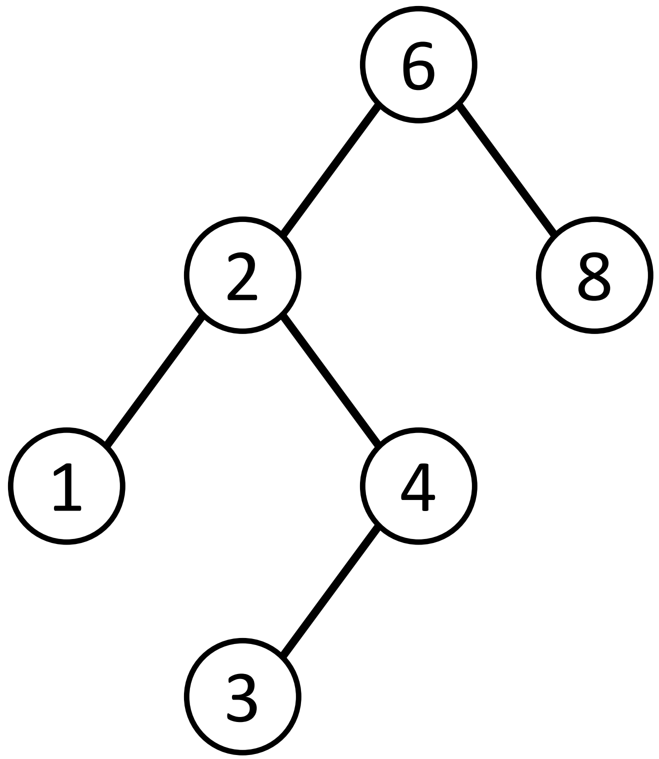
Containers: Binary Search Trees

Jordi Cortadella and Jordi Petit
Department of Computer Science

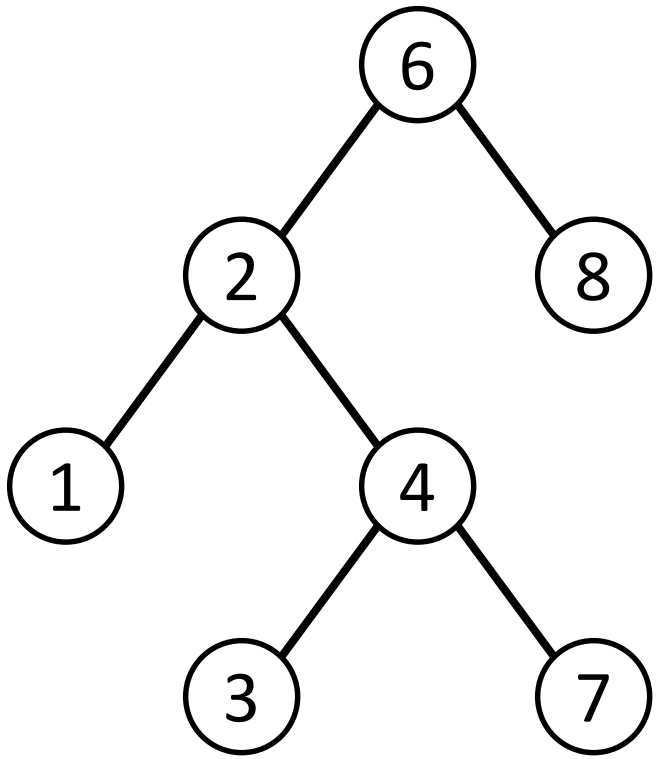
Binary Search Trees

BST property: for every node in the tree with value V:

- All values in the left subtree are smaller than V.
- All values in the right subtree are larger than V.

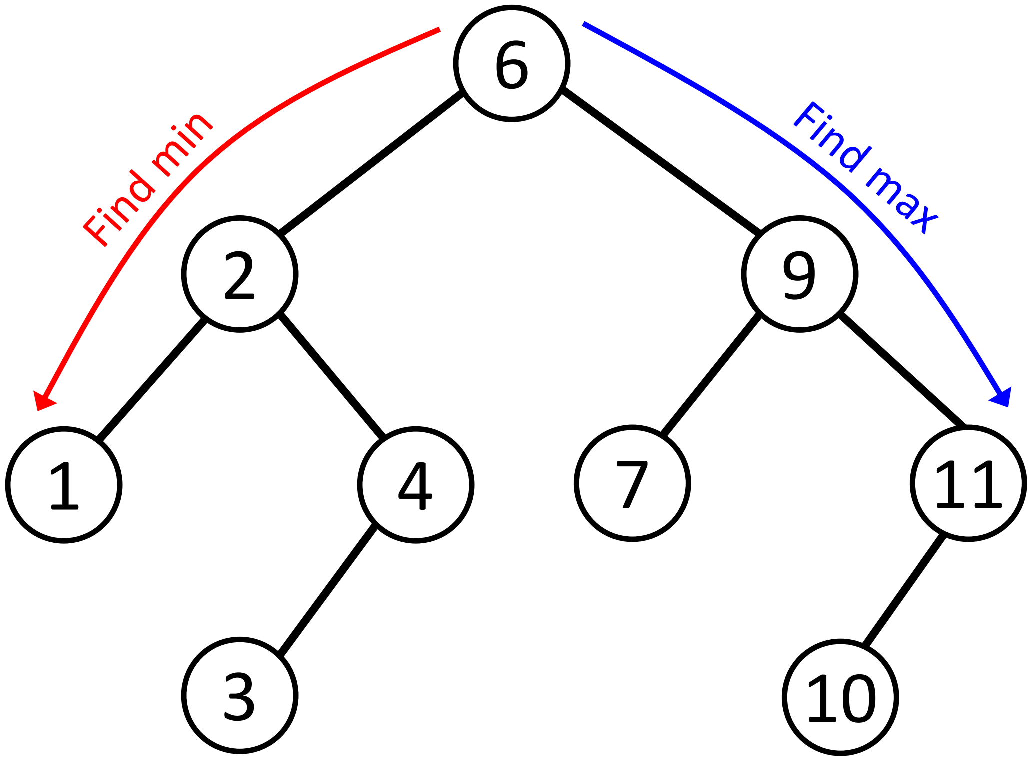


This is a binary search tree



This is ***not*** a binary search tree

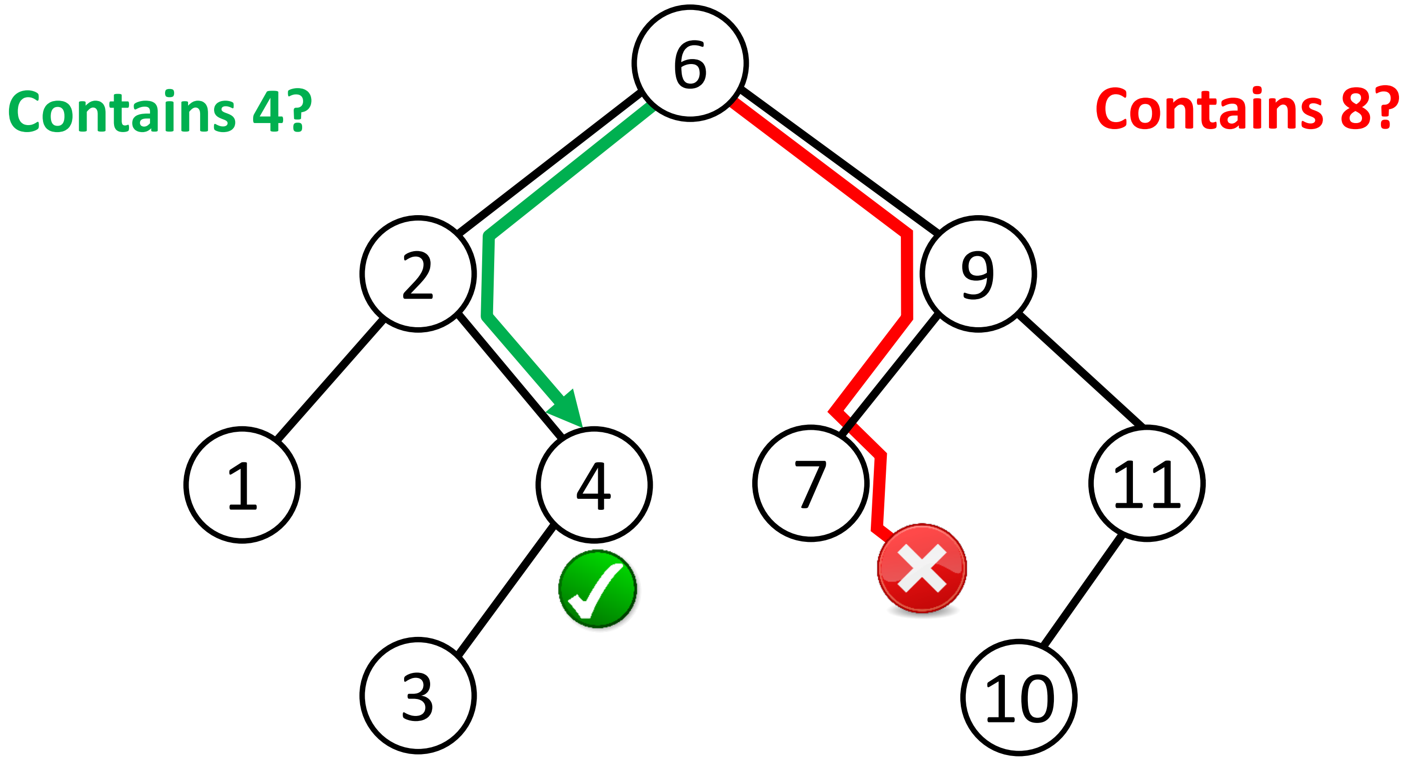
BST: find min/max



Find min: Go to the leftmost element.

Find max: Go to the rightmost element.

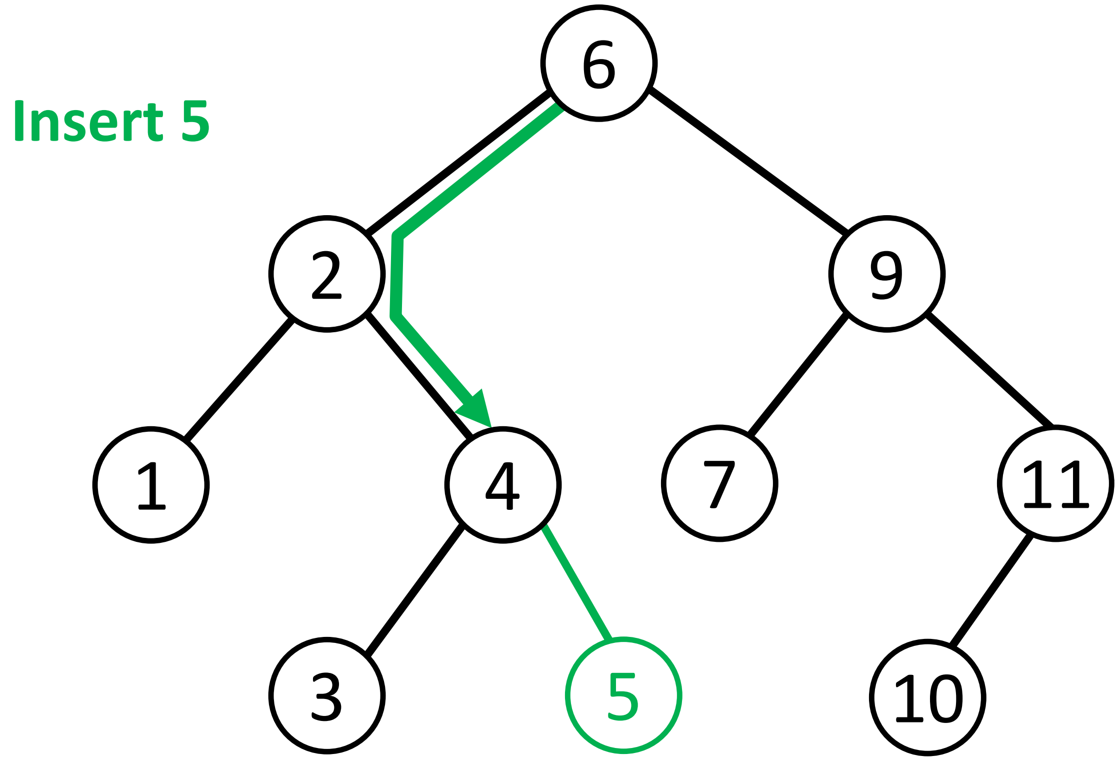
BST: find an element



Find an element:

- Move to left/right depending on the value.
- Stop when:
  - The value is found (contained)
  - No more elements exist (not contained)

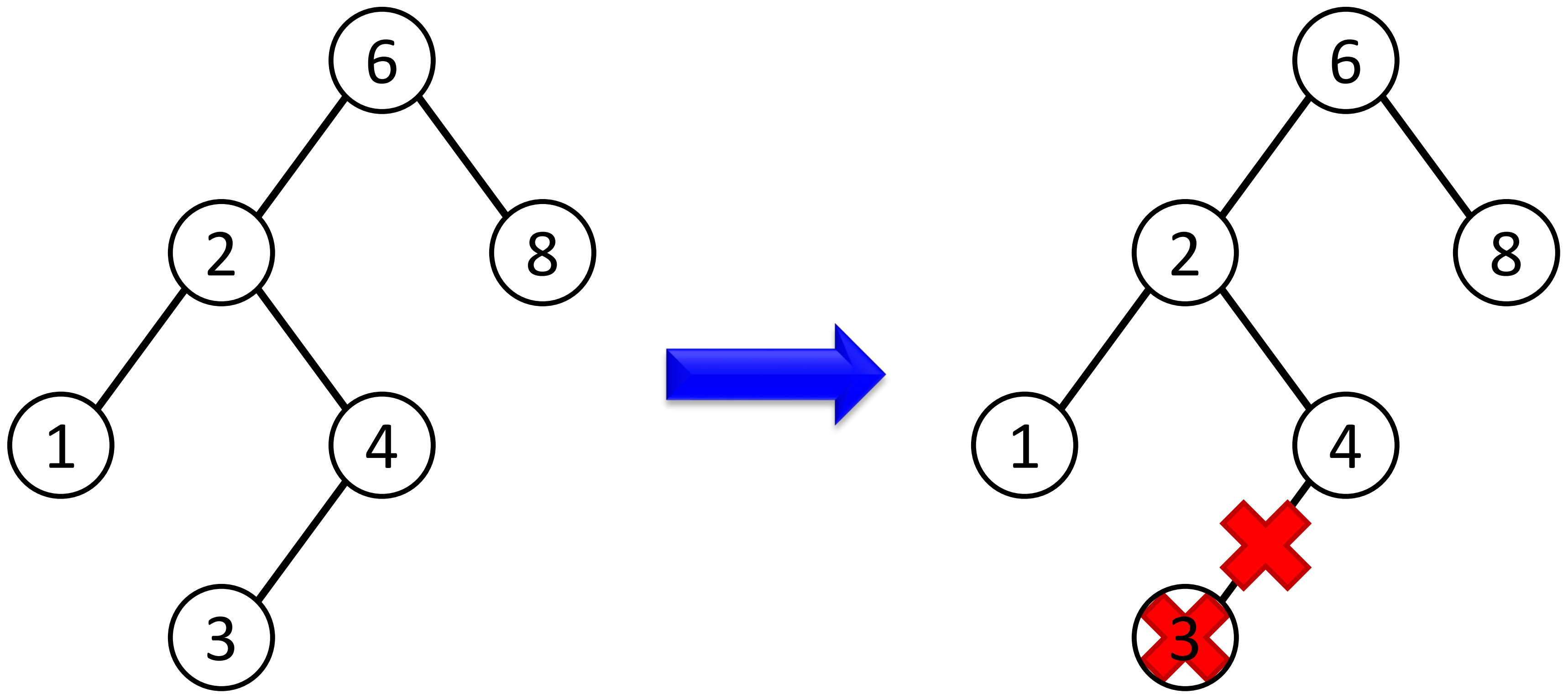
BST: insert an element



Insert:

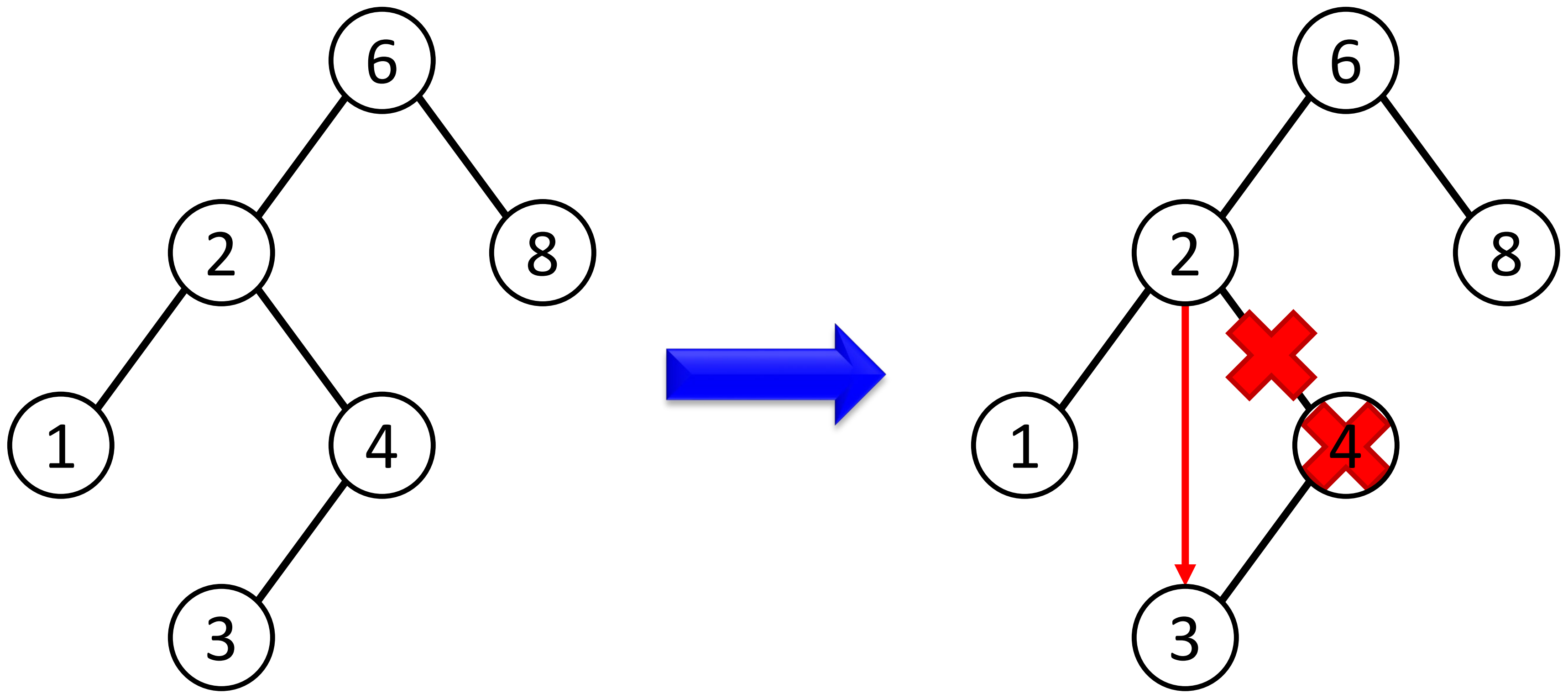
- Move to left/right depending on the value.
- Stop when the element is found (nothing to do) or a null is found.
- If not found, substitute null by the new element.

remove: simple case (no children)



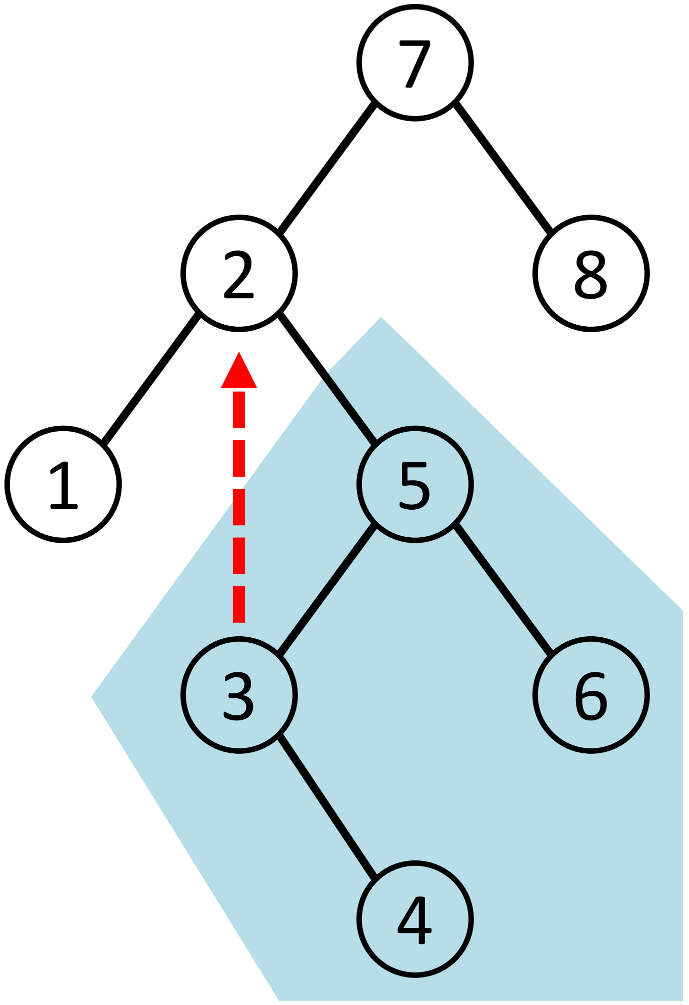
```
remove(3)
```

remove: simple case (one child)



`remove(4)`

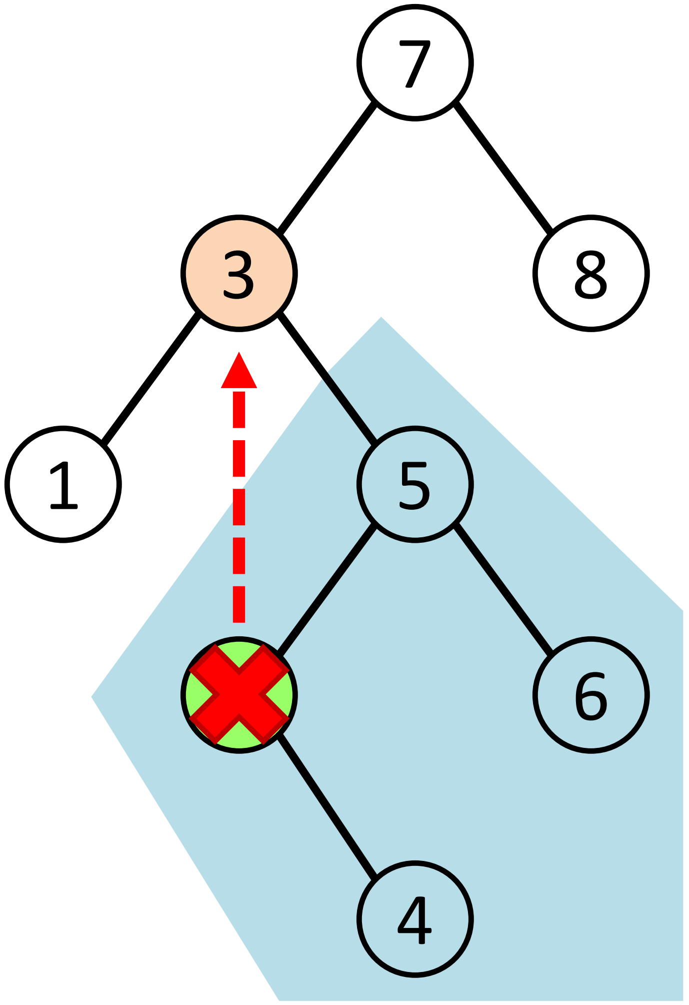
remove: complex case (two children)



`remove(2)`

1. Find the element.
2. Find the min value of the right subtree.
3. Copy the min value onto the element to be removed.

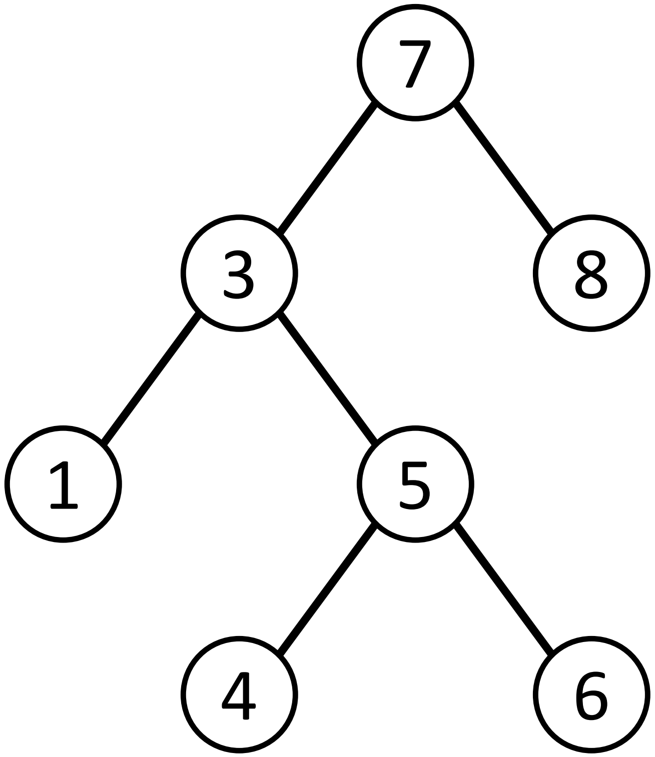
remove: complex case (two children)



`remove(2)`

1. Find the element.
2. Find the min value of the right subtree.
3. Copy the min value onto the element to be removed.
4. Remove the min value in the right subtree (simple case).

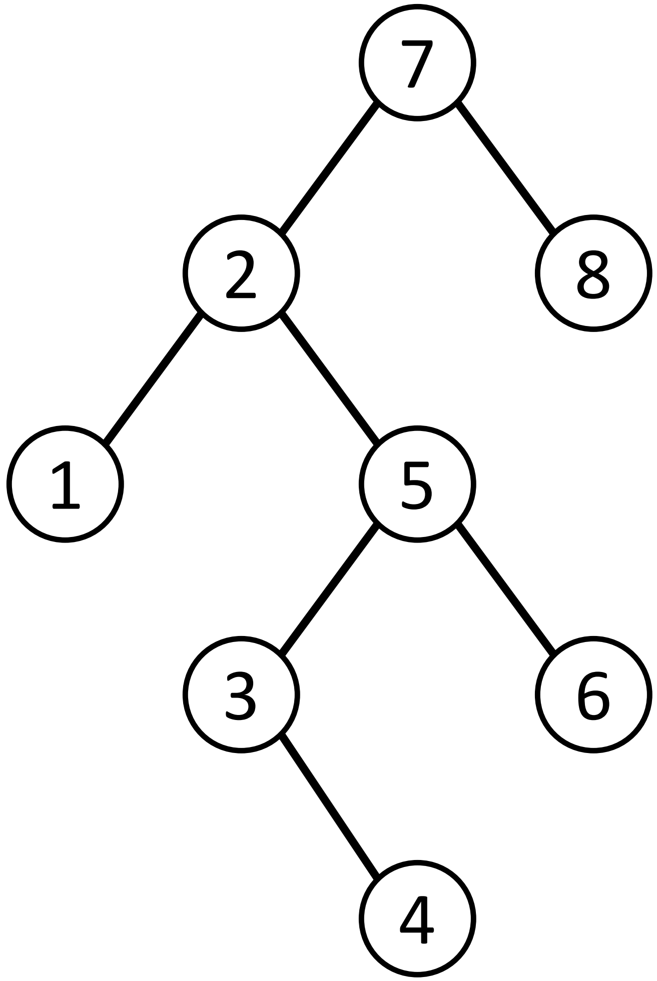
remove: complex case (two children)



1. Find the element.
2. Find the min value of the right subtree.
3. Copy the min value onto the element to be removed.
4. Remove the min value in the right subtree (simple case).

```
remove(2)
```

Visiting the items in ascending order



Question:

How can we visit the items of a BST in ascending order?

Answer:

Using an **in-order** traversal

BST: runtime analysis

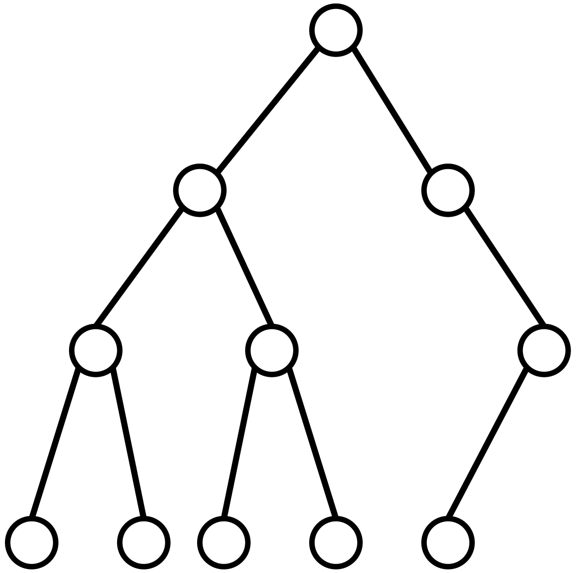
- Copying and deleting the full tree takes $\mathrm{O}(n)$.

- We are mostly interested in the runtime of the `insert`/`remove`/`contains` methods.
  - The complexity is $\mathrm{O}(d)$, where d is the depth of the node containing the required element.

- But, how large is d?

BST: runtime analysis

- Internal path length (IPL): The sum of the depths of all nodes in a tree. Let us calculate the average IPL considering all possible insertion sequences.



$d = 0$

$d = 1$

$d = 2$

$d = 3$

$$\text{ILP} = 0 \times 1 + 1 \times 2 + 2 \times 3 + 3 \times 5 = 23$$

$$\text{Avg. IPL} = \frac{23}{11} \approx 2.09$$

BST: runtime analysis

- Internal path length (IPL): The sum of the depths of all nodes in a tree. Let us calculate the average IPL considering all possible insertion sequences.

- $D(n)$ is the IPL of a tree with n nodes. $D(1) = 0$. The left subtree has i nodes and the right subtree has $n - i - 1$ nodes. Thus,

$$D(n) = D(i) + D(n - i - 1) + (n - 1)$$

- If all subtree sizes are equally likely, then the average value for $D(i)$ and $D(n - i - 1)$ is

$$\frac{1}{n}\sum_{j=0}^{n-1} D(j)$$

BST: runtime analysis

- Therefore,

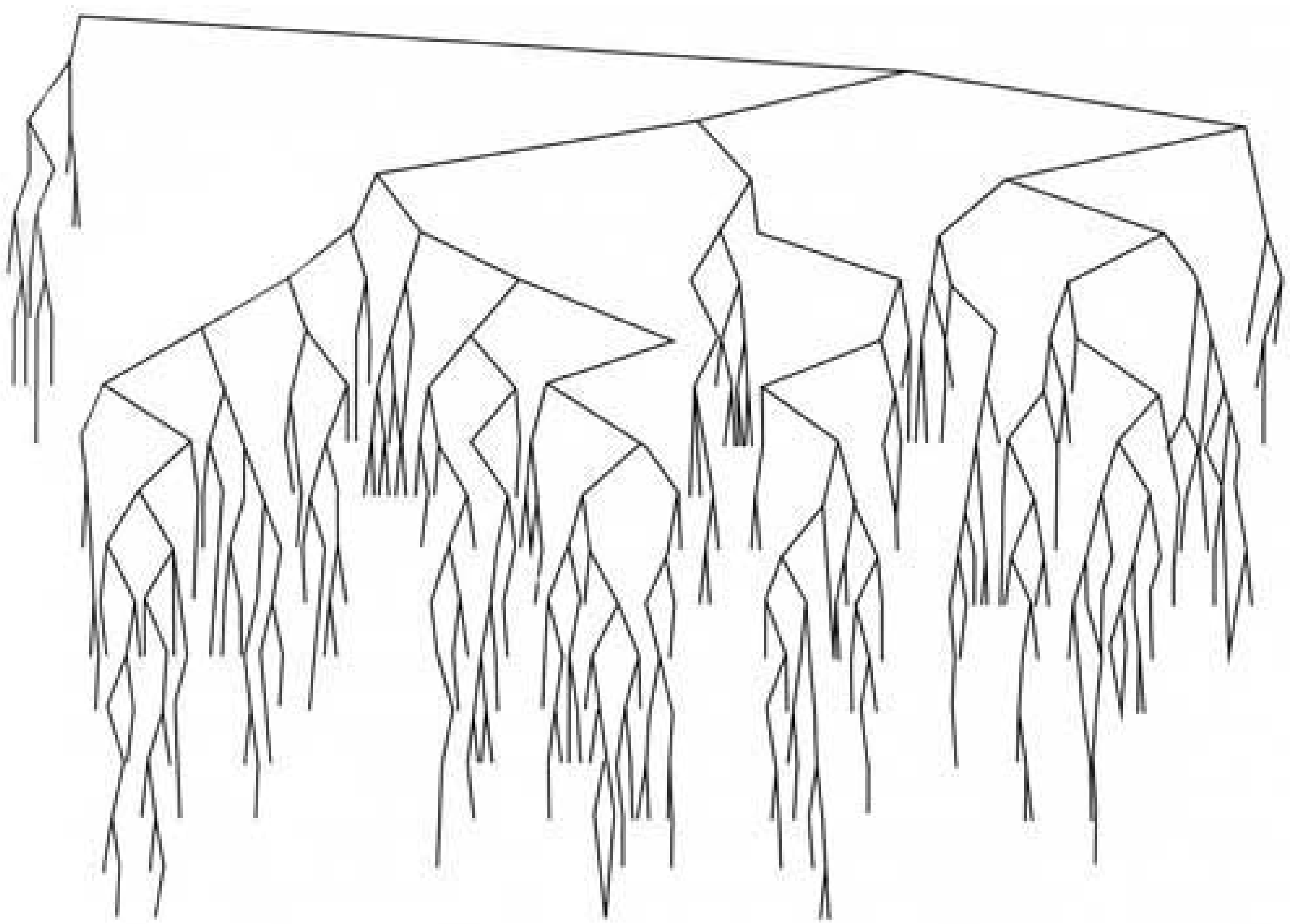
$$D(n) = \frac{2}{n}\left[\sum_{j=0}^{n-1} D(j)\right] + n - 1$$

- The previous recurrence gives: $D(n) = \mathrm{O}(n \log n)$

- The average height of nodes after n random insertions is $\mathrm{O}(\log n)$.

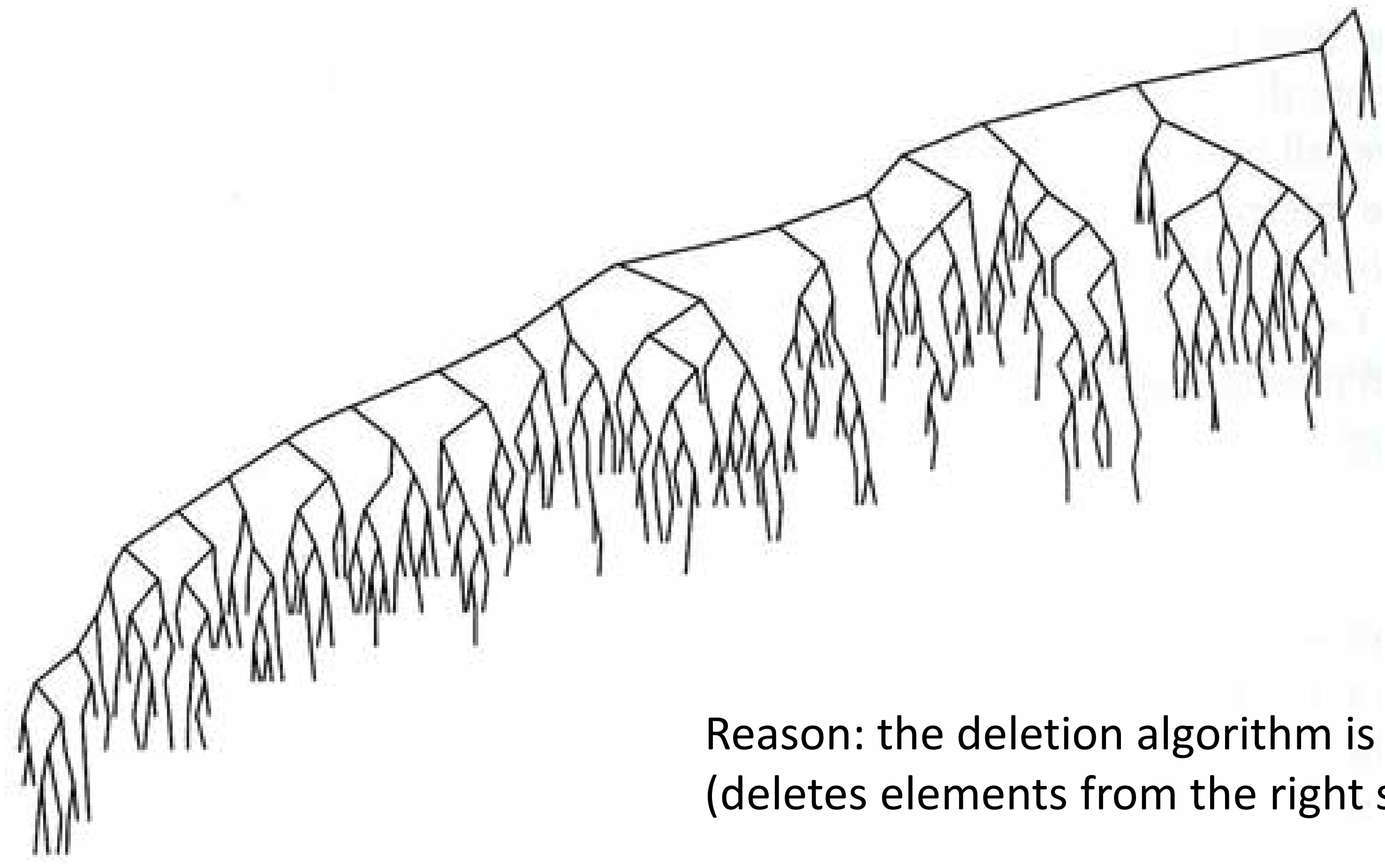
- However, the $\mathrm{O}(\log n)$ average height is not preserved when doing deletions.

Random BST



Source: Fig 4.29 of Weiss textbook

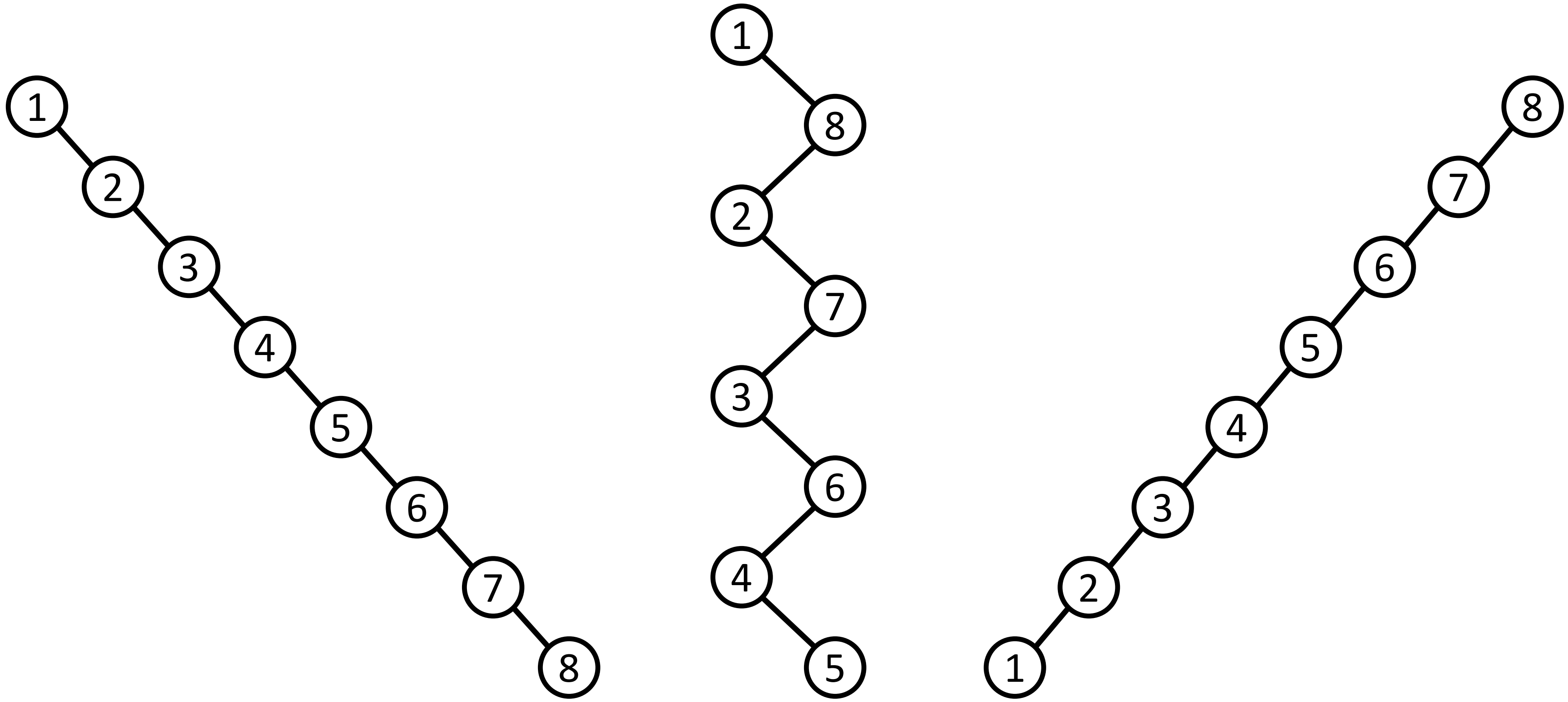
Random BST after n^2 insert/removes



Reason: the deletion algorithm is asymmetric (deletes elements from the right subtree)

Source: Fig 4.30 of Weiss textbook

Worst-case runtime: $O(n)$



Self-balancing BST

- The worst-case complexity for insert, remove and search operations in a BST is $\mathrm{O}(n)$, where n is the number of elements.

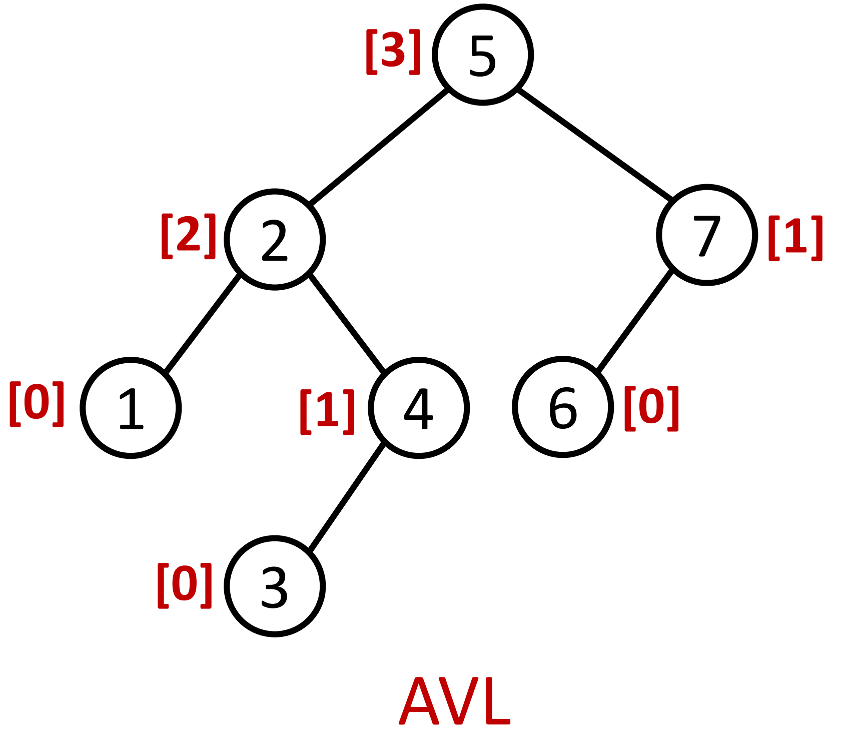
- Various representations have been proposed to keep the height of the tree as $\mathrm{O}(\log n)$:
  - AVL trees
  - Red-Black trees

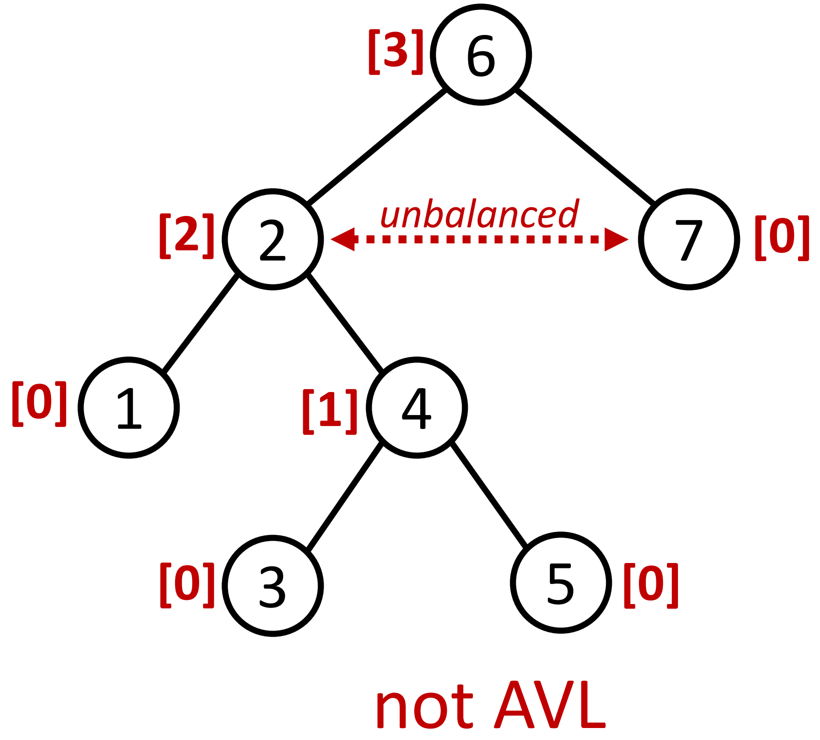
AVL trees

- Named after Adelson-Velsky and Landis (1962).
- Main idea: invest some additional time to balance the tree each time a new element is inserted or deleted.
- Properties:
  - The height of the tree is always $\Theta(\log n)$.
  - The time devoted to balancing is $\mathrm{O}(\log n)$.

AVL tree: definition

- An AVL tree is a BST such that, for every node, the difference between the heights of the left and right subtrees is at most 1.





AVL tree in action

https://en.wikipedia.org/wiki/AVL_tree

Complexity

- Single and double rotations only need the manipulation of few pointers and the height of the nodes ($\mathrm{O}(1)$).

- Insertion: the height of the subtree after a rotation is the same as the height before the insertion. Therefore, at most only one rotation must be applied for each insertion.

- Deletion: more complicated. More than one rotation might be required.

- Worst case for deletion: $\mathrm{O}(\log n)$ rotations (a chain effect from leaves to root).

EXERCISES

BST

- Starting from an empty BST, depict the BST after inserting the values 32, 15, 47, 67, 78, 39, 63, 21, 12, 27.
- Depict the previous BST after removing the values 63, 21, 15 and 32.

Methods of a BST

Use the class **BinTree** from the previous chapter and implement the following methods for a BST:

```
def find_min(t: BinTree[T]) -> BinTree[T]:
    """Returns the tree (node) containing the min element"""

def find_max(t: BinTree[T]) -> BinTree[T]:
    """Returns the tree (node) containing the max element"""

def find(t: BinTree[T], data: T) -> BinTree[T]:
    """Returns the tree (node) where data is located,
       or None if not found"""

def insert(t: BinTree[T], data: T) -> BinTree[T]:
    """Inserts data on the BST. It returns the new
       BinTree after insertion (t if it was not empty)"""

def remove(t: BinTree[T], data: T) -> BinTree[T]:
    """Removes data from the BST. It returns the new
       BinTree (None if it becomes empty)"""
```

Merging BSTs

- Describe an algorithm to generate a sorted list from a BST. What is its cost?
- Describe an algorithm to create a balanced BST from a sorted list. What is its cost?
- Describe an algorithm to create a balanced BST that contains the union of the elements of two BSTs. What is its cost?