Data Structures and Algorithms

Géza Horváth

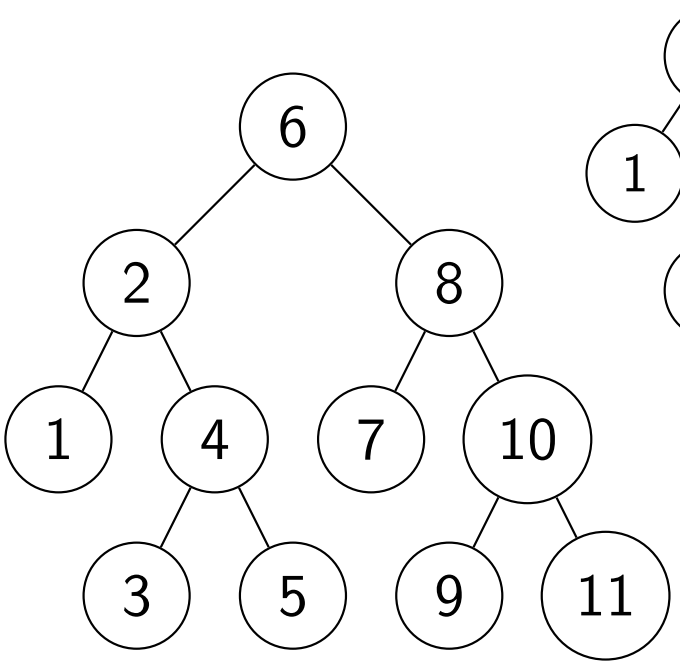
Seventh Lecture

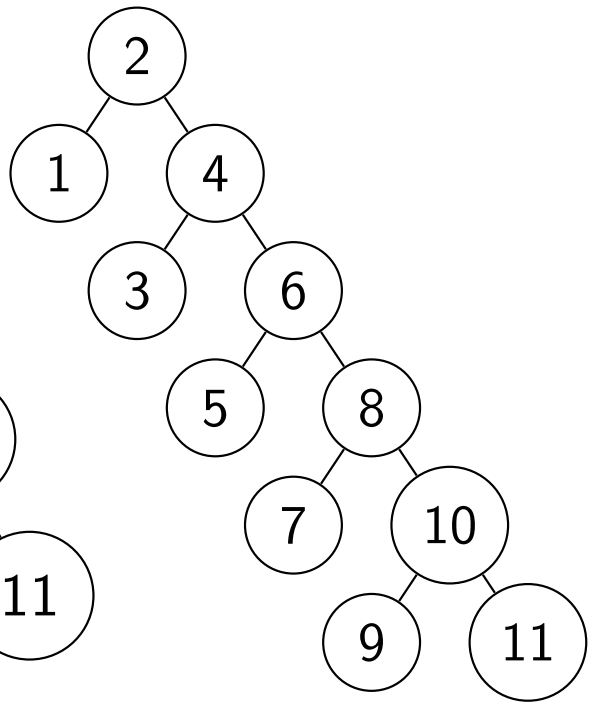
Lecture topics

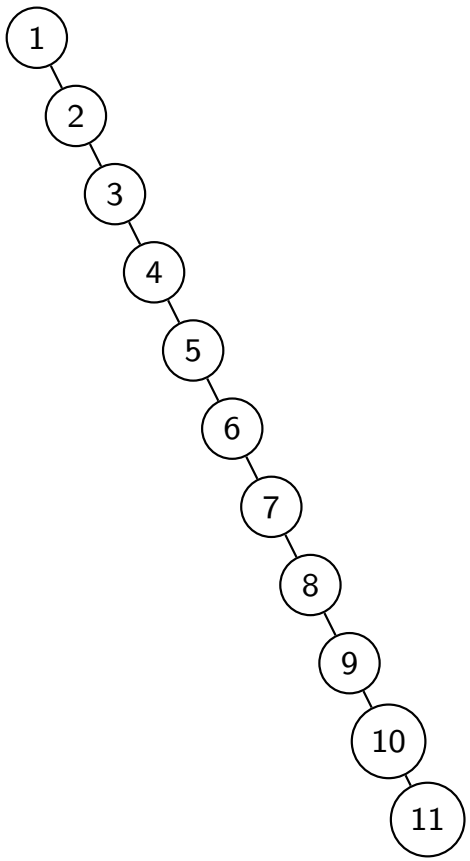
1. Introductory examples of the design and analysis of algorithms.
2. Asymptotic behavior of functions, asymptotic notation, further examples of simple algorithms.
3. Basic notions concerning data structures: modeling, abstraction. Set, multiset, array, matrix.
4. Elementary data structures: stacks, queues, lists.
5. Tables, hash tables, hash functions, collisions.
6. Trees, binary trees, binary search trees, traversal, searching, insertion, deletion.
7. Balanced binary search trees, AVL trees.
8. Red-black trees.
9. B-trees.
10. Graphs, traversal, shortest path.
11. Parallel algorithms: matrix multiplication, merge sort.
12. Decision problems, the complexity classes P and NP.
13. Summary.

Search trees

How long it takes to find a key?

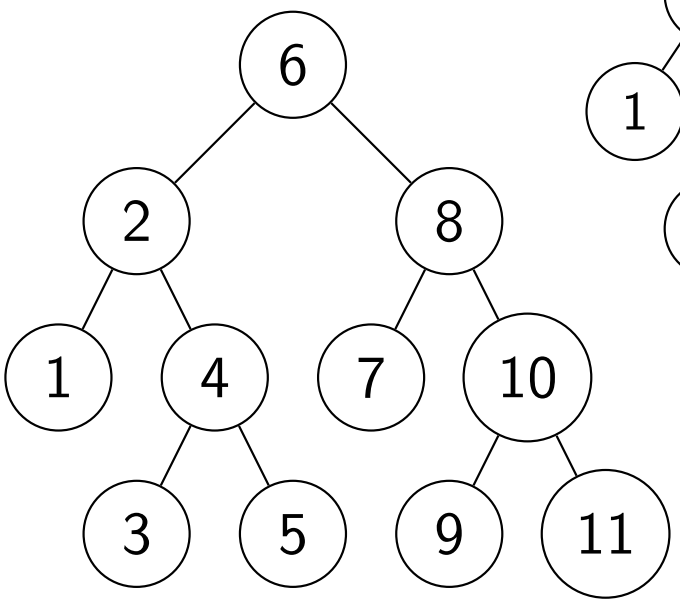




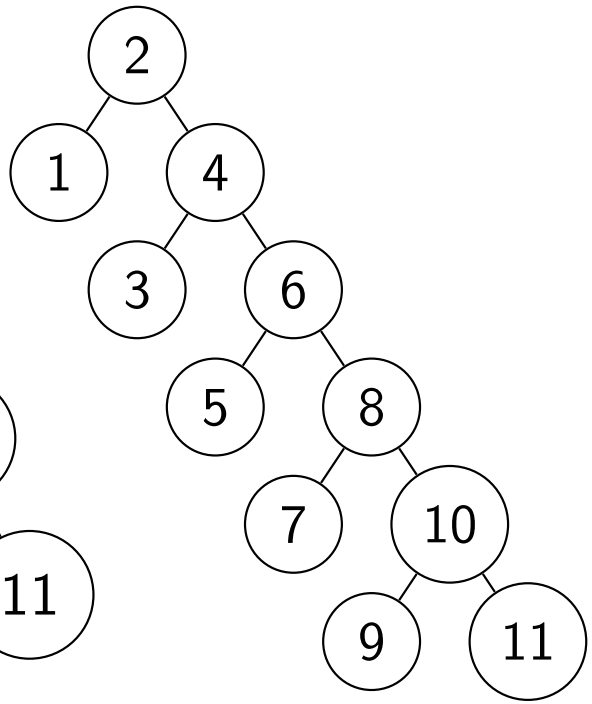


Search trees

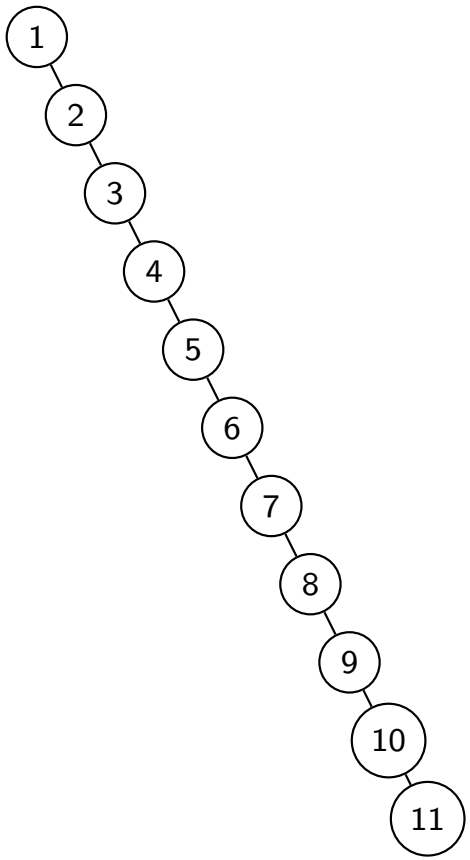
How long it takes to find a key?



Logarithmic!

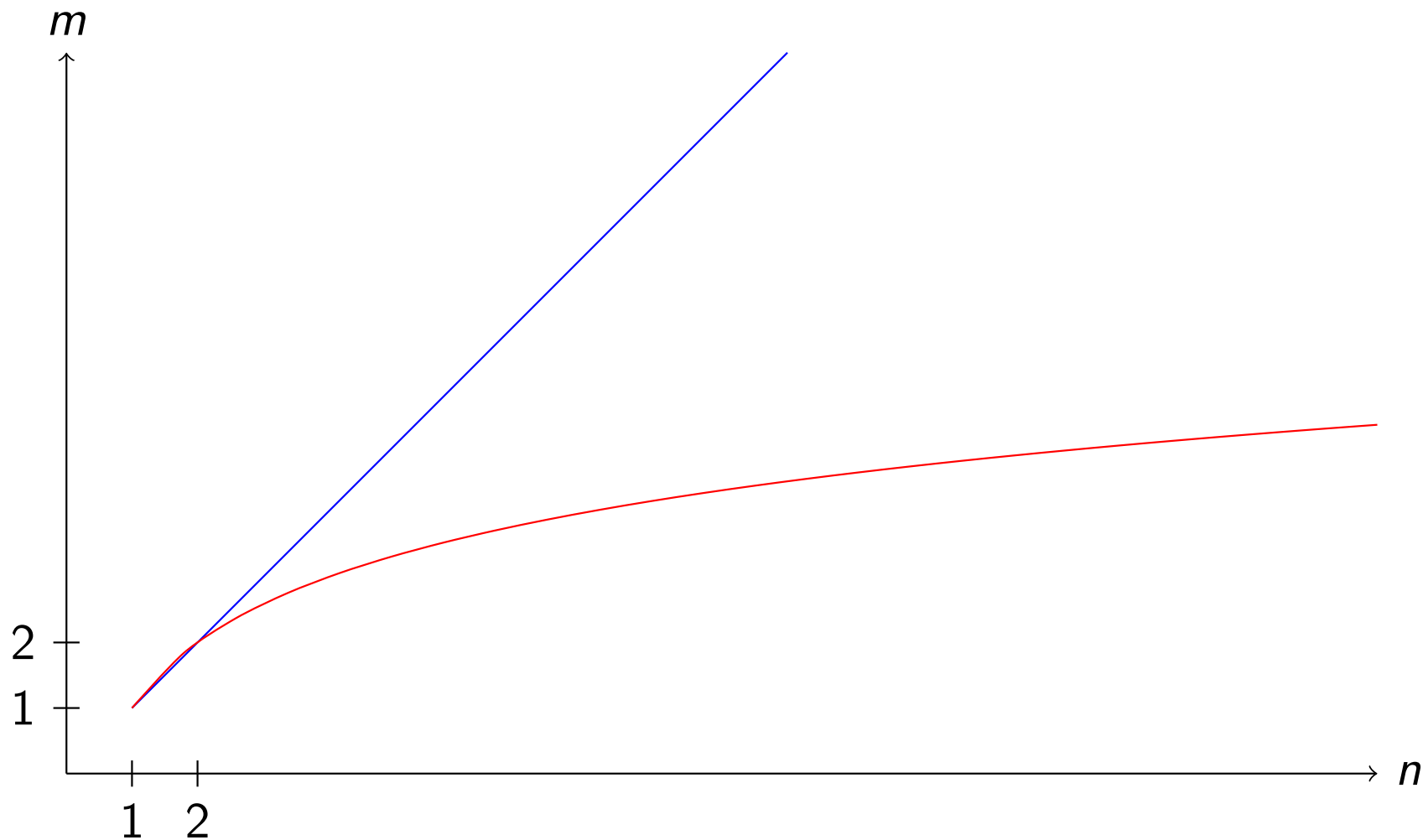


???



Linear.

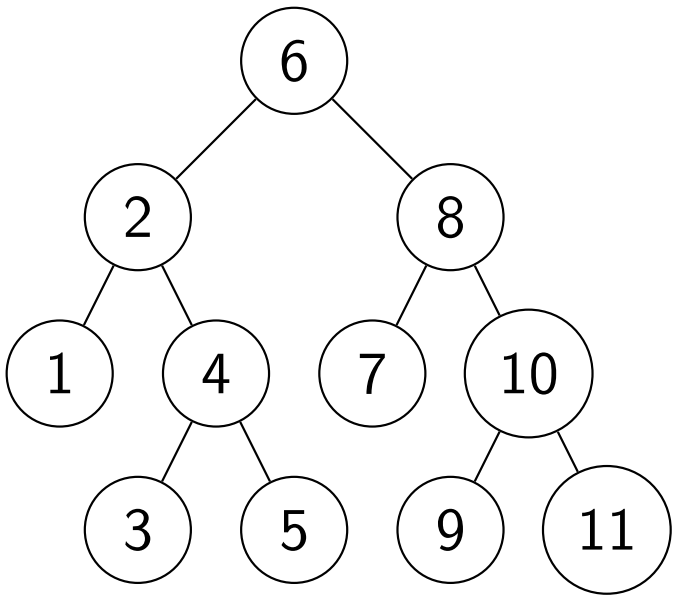
Linear and logarithmic search comparison

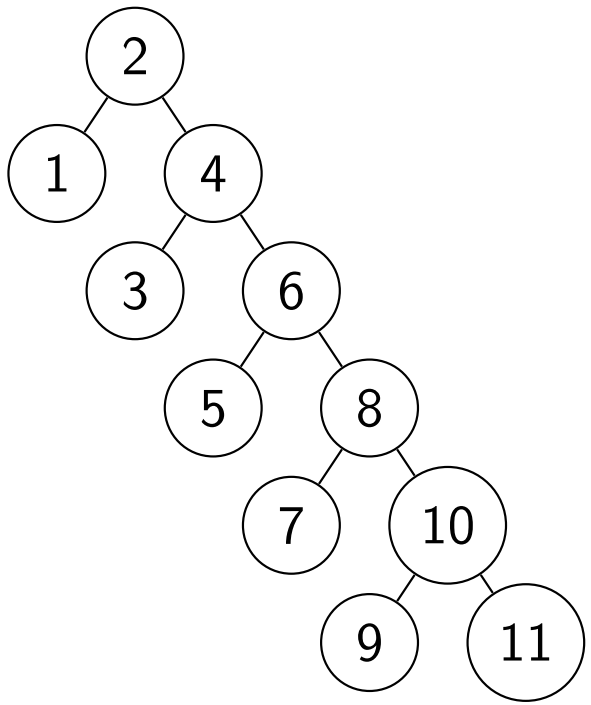


n – **input size**

m – **running time**

Search trees



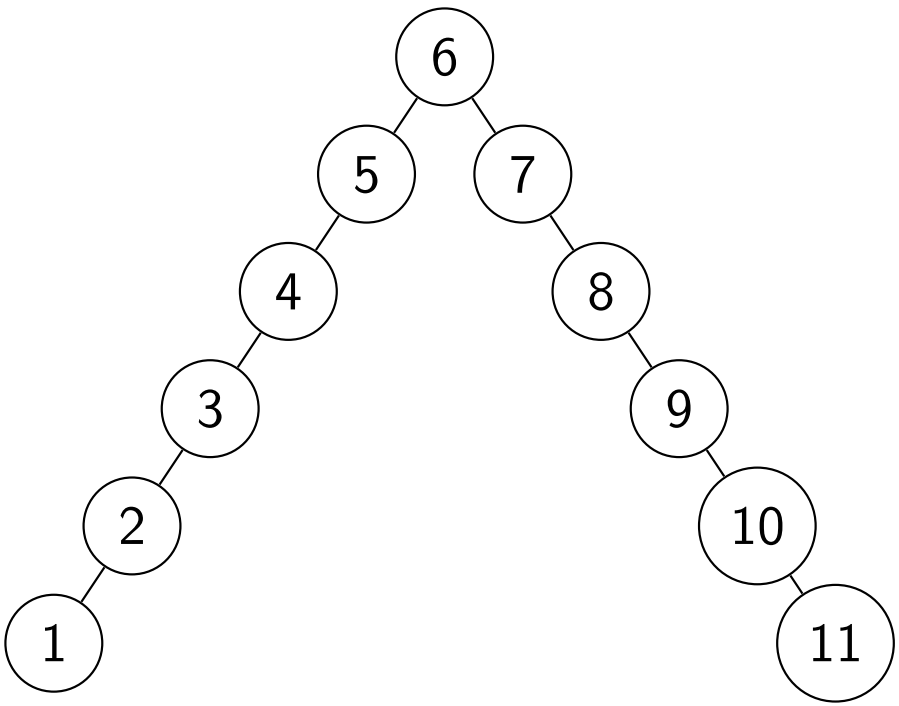


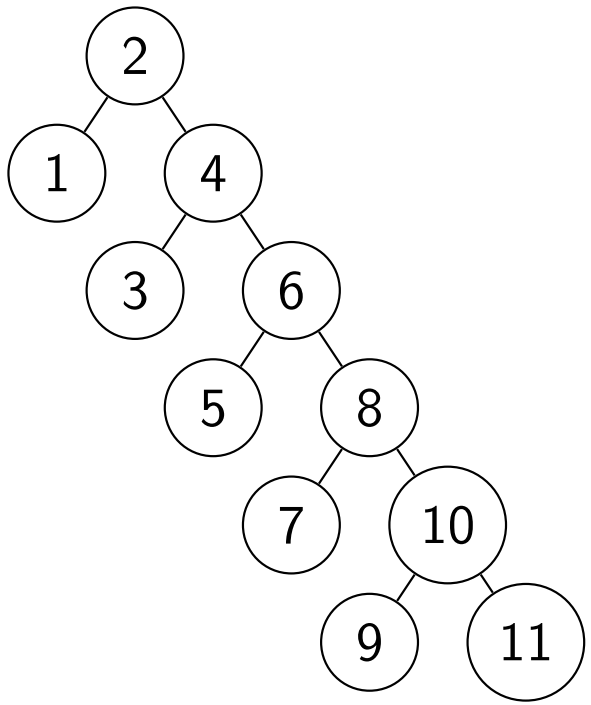
Good case, **balanced tree.**

- TREE-SEARCH – $\mathcal{O}(log_2 n)$
- TREE-INSERT – $\mathcal{O}(log_2 n)$
- TREE-DELETE – $\mathcal{O}(log_2 n)$

- TREE-SEARCH – $\mathcal{O}(n)$
- TREE-INSERT – $\mathcal{O}(n)$
- TREE-DELETE – $\mathcal{O}(n)$

Search trees





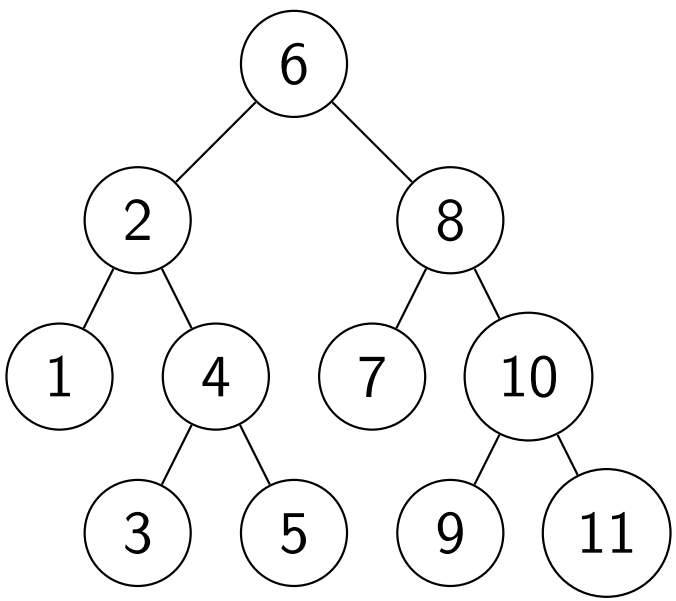
- Tree-Search – $\mathcal{O}(n)$
- Tree-Insert – $\mathcal{O}(n)$
- Tree-Delete – $\mathcal{O}(n)$

- Tree-Search – $\mathcal{O}(n)$
- Tree-Insert – $\mathcal{O}(n)$
- Tree-Delete – $\mathcal{O}(n)$

Balanced binary search tree

Definition

A balanced binary search tree is a binary search tree in which the left and right subtrees of every node differ in height by no more than 1.

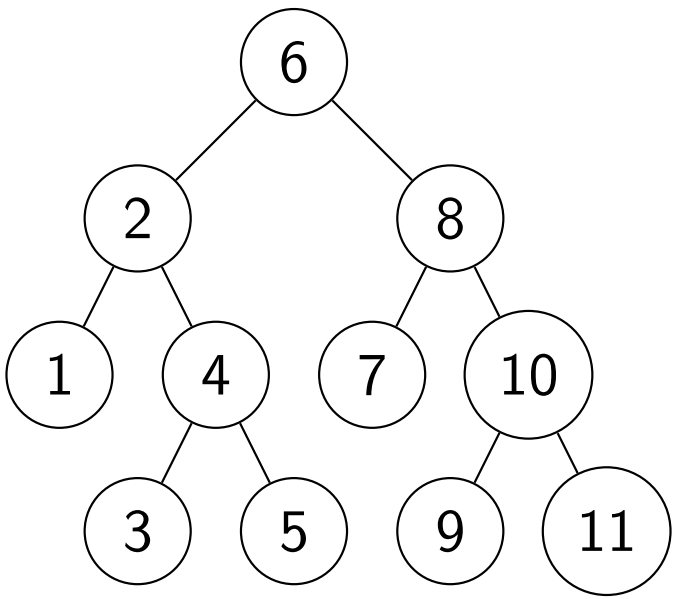


See more examples on the whiteboard!

Balanced binary search tree

Theorem

Searching, insertion, and deletion all take $\mathcal{O}(log_2 n)$ time in both the average and worst cases.



Which operations can disturb the balance?

AVL tree

In computer science, an AVL tree (named after inventors Georgy Adelson-Velsky and Evgenii Landis) is a self-balancing binary search tree. It was the first such data structure to be invented (in 1962). In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

AVL tree – data structure operations

Implemented data structure operations:

- **create** data structure
- **modify** data structure
  - add elements – Tree-Insert + Rebalance
  - delete elements – Tree-Delete + Rebalance
- **access** – Tree-Search

AVL tree – balance factor

Definition

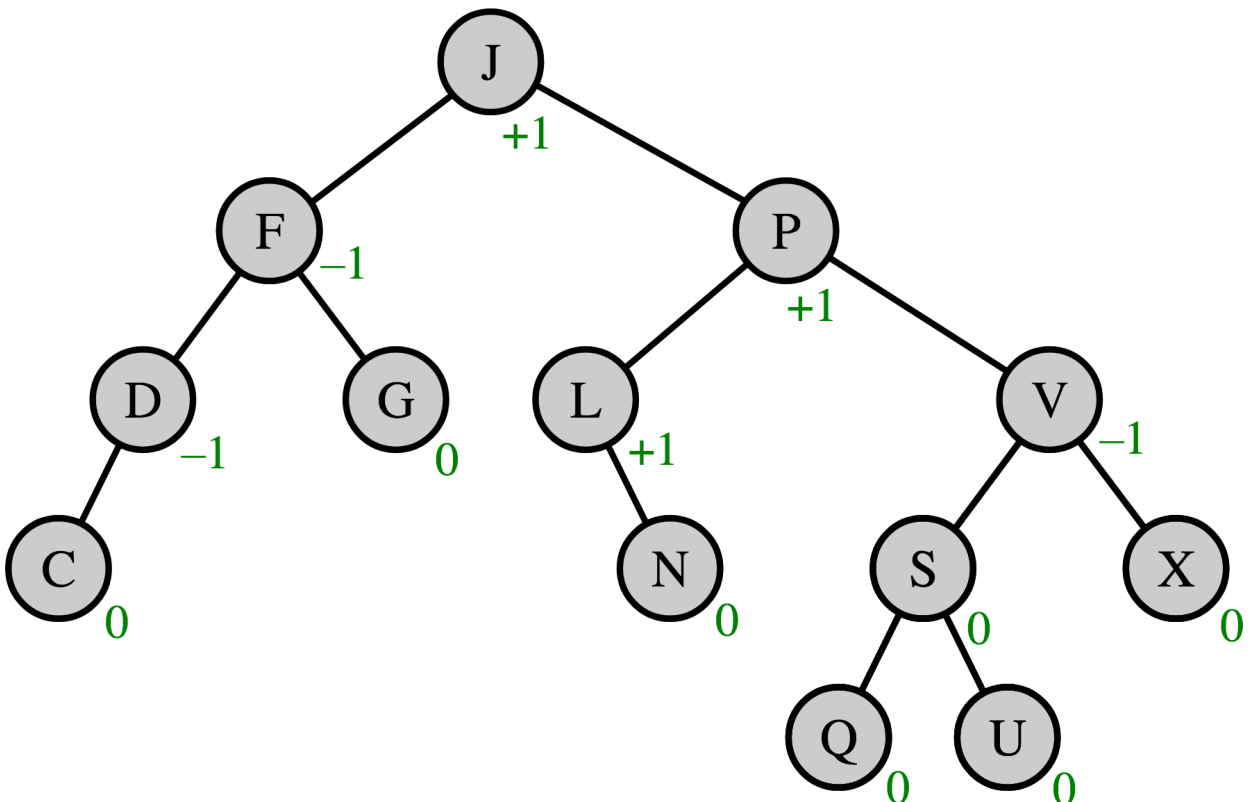
In an AVL tree the balance factor of a node N is defined to be the height difference: BalanceFactor(N)=Height(RightSubtree(N))–Height(LeftSubtree(N)).

Remark: The height of a tree with only one node is 0 by definition, and the height of the empty tree is undefined, but the height of a tree with one node minus the height of the empty tree should be one, so, – in this case, – we use -1 as the height of the empty tree, during the calculations of the balance factors.

AVL tree – balance factor

Definition

In an AVL tree the balance factor of a node N is defined to be the height difference: BalanceFactor(N)=Height(RightSubtree(N))–Height(LeftSubtree(N)).



AVL tree – rebalancing

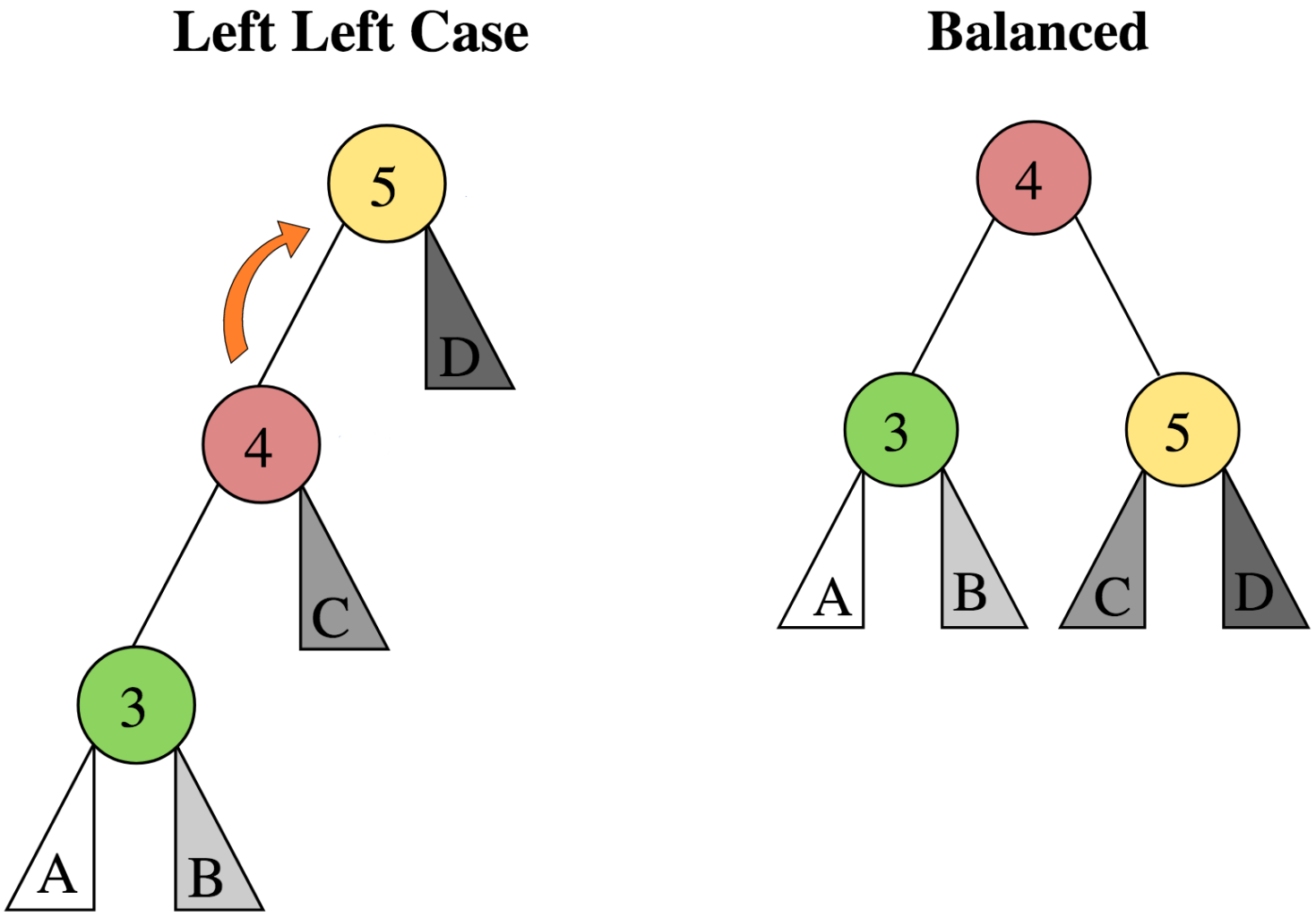
Rebalancing is not always necessary, only if during a modifying operation (e.g. insert, delete) a (temporary) height difference of more than one arises between two child subtrees. In this case the parent subtree has to be rebalanced.

There are 4 different cases, when rebalancing is necessary:

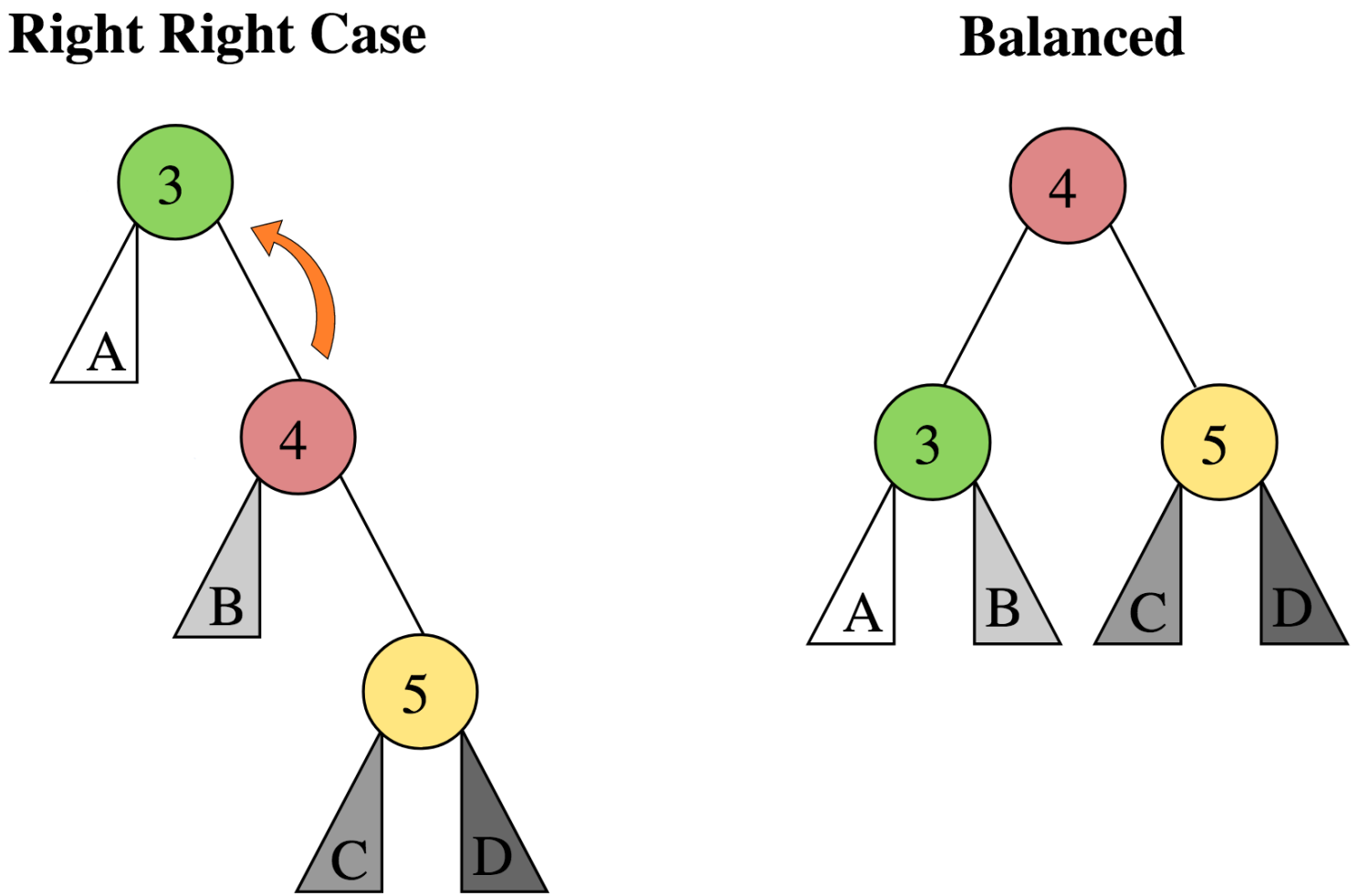
- Left-Left Case (LL)
- Left-Right Case (LR)
- Right-Left Case (RL)
- Right-Right Case (RR)

The given repair tools are the so-called **tree rotations.**

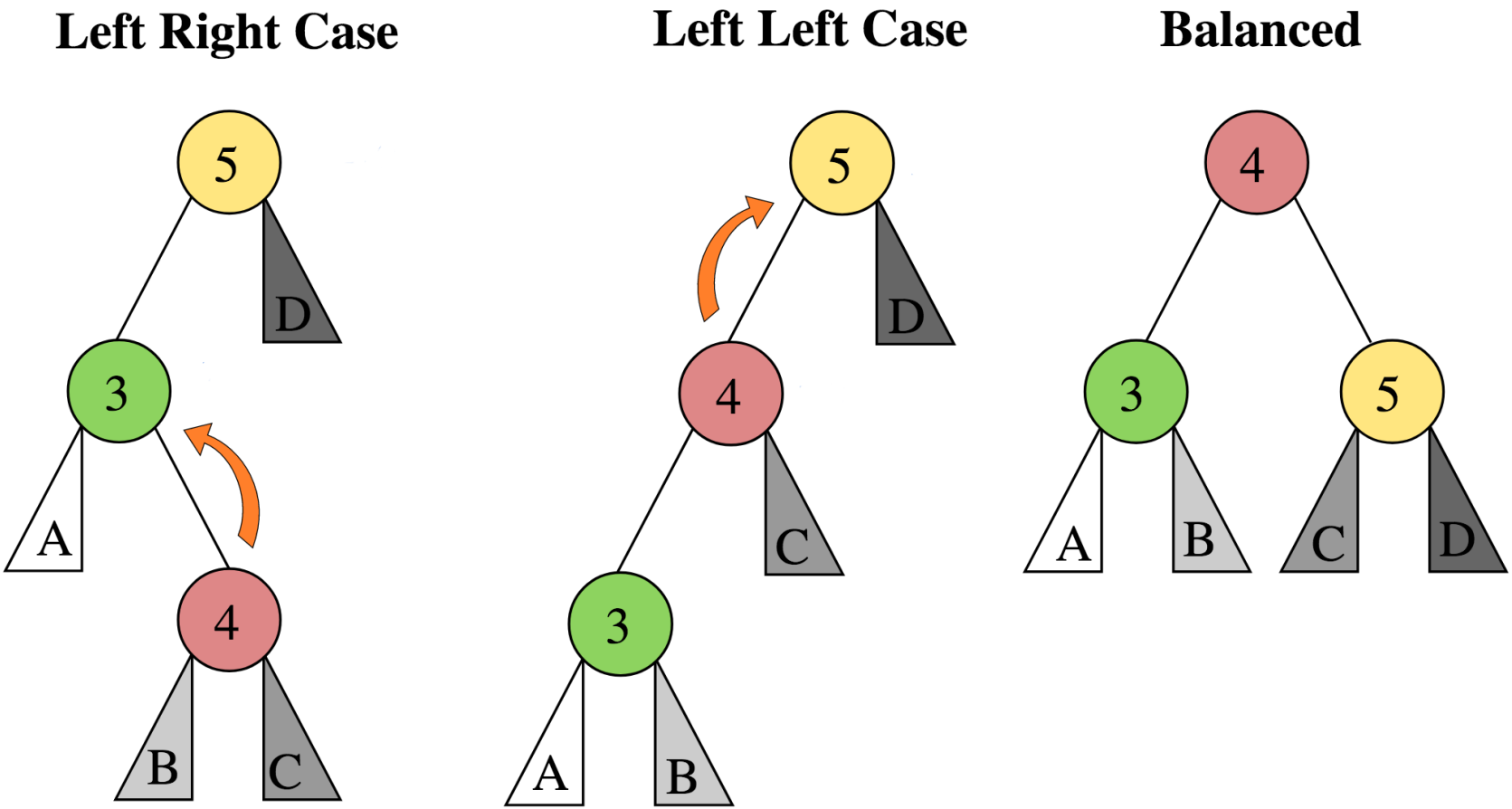
AVL tree – rebalancing LL case



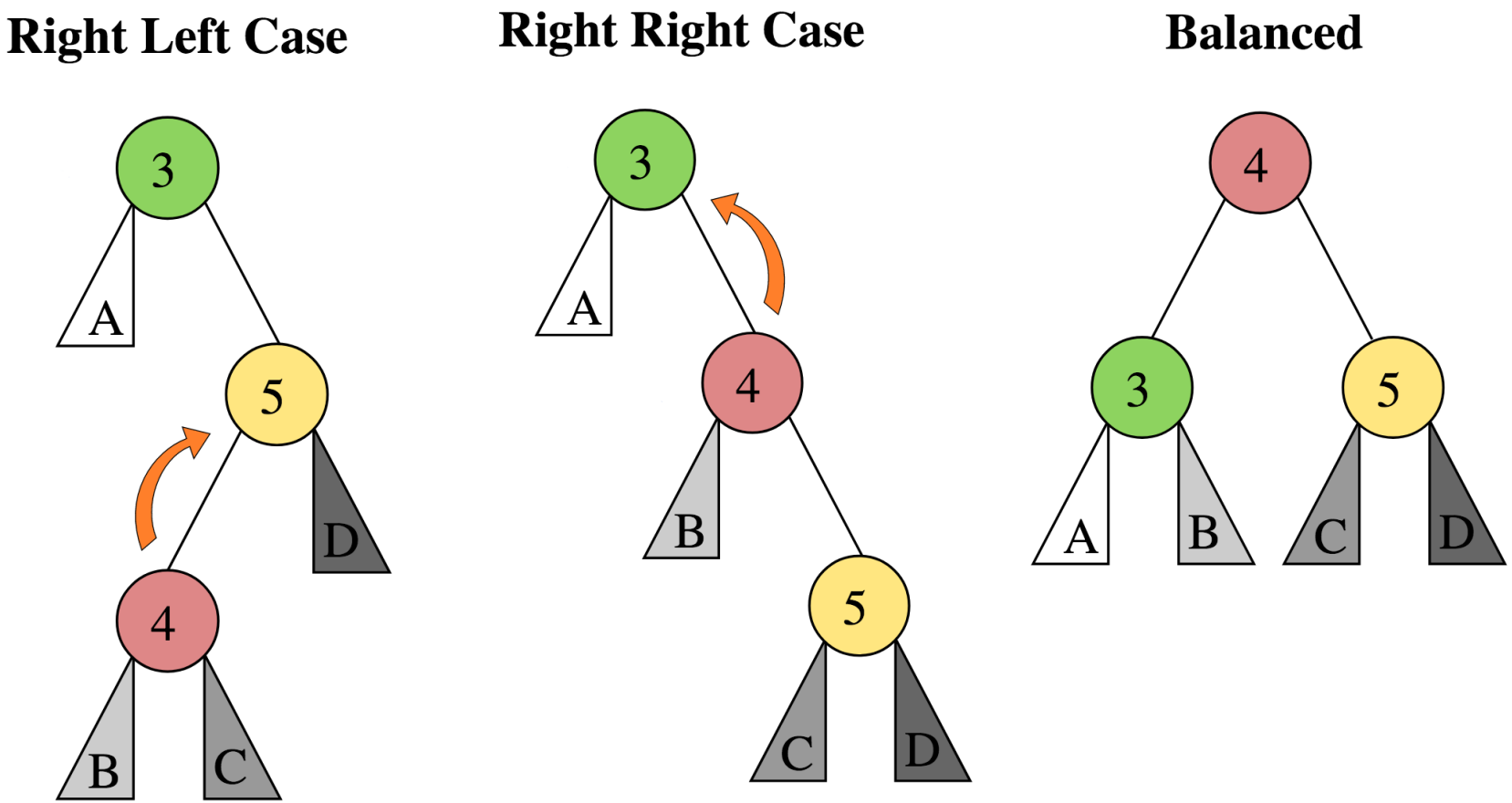
AVL tree – rebalancing RR case



AVL tree – rebalancing LR case



AVL tree – rebalancing RL case



References

- Wikipedia – it is ok for this topic!

This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.