ALGORITHMS & DATA STRUCTURES

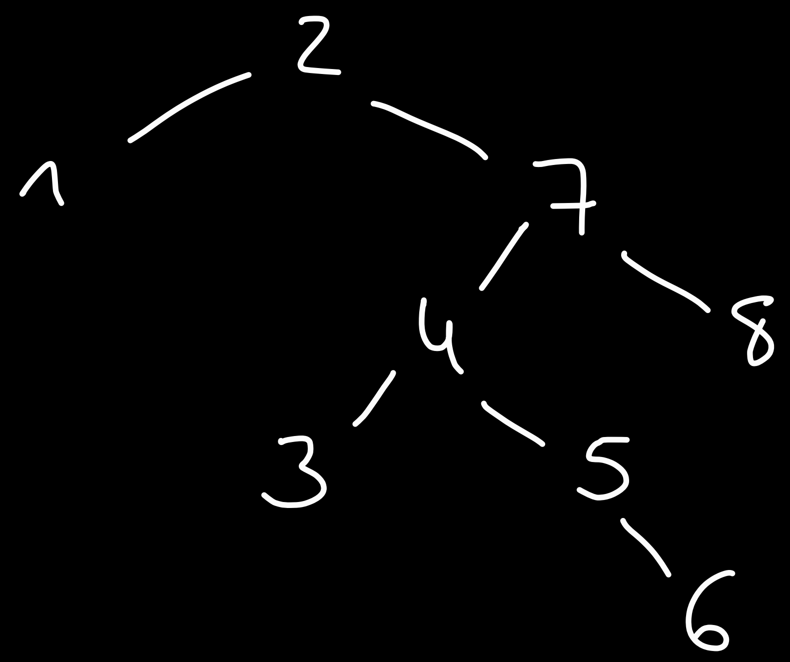
29th November 2021

PLAN FOR TODAY

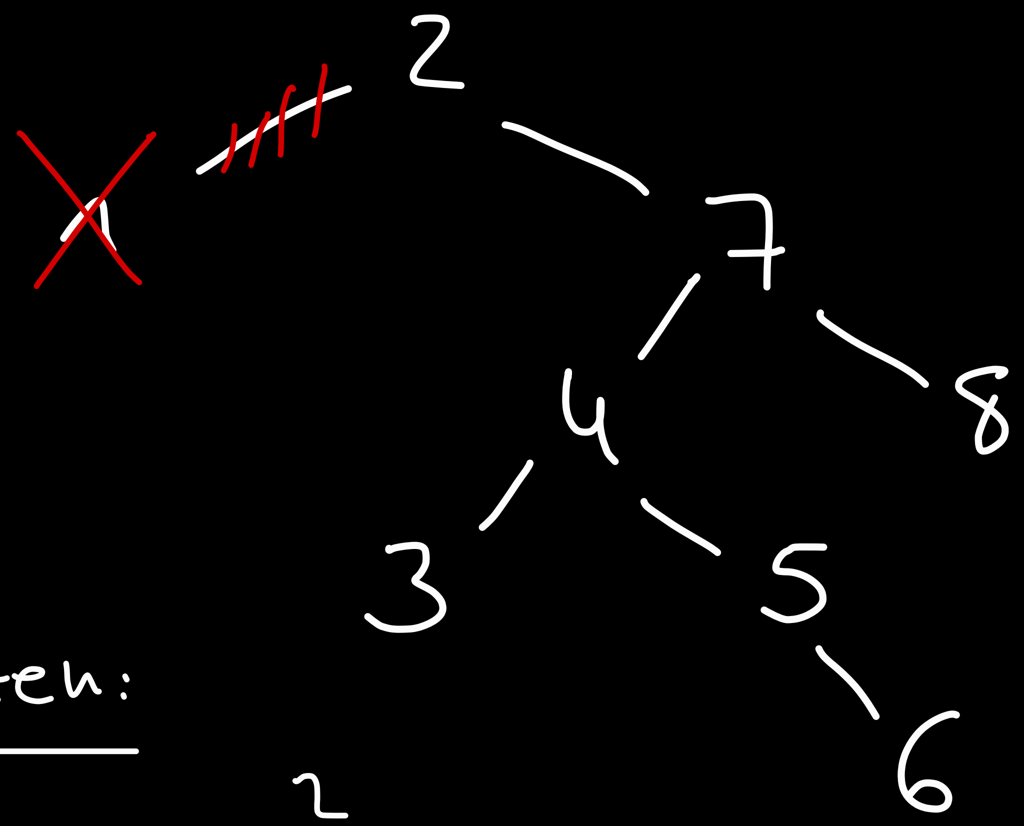
- Bonus Exercises
- Exercise 9.3
- A bit of Graph Theory

EXERCISE 9.1

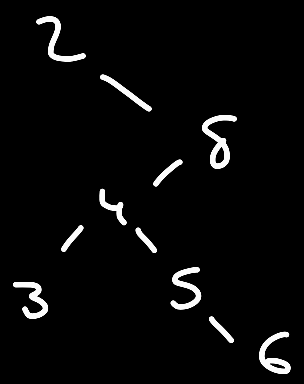
a) Draw the resulting tree when the keys 2, 7, 8, 4, 5, 6, 3, 1 in this order are inserted into an initially empty binary (natural) search tree.

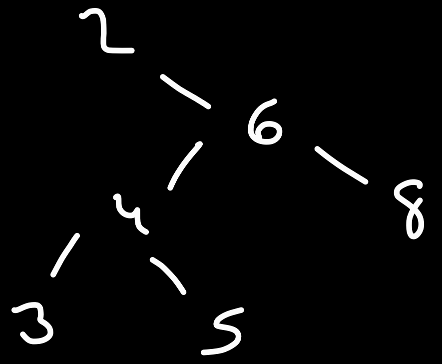


b) Delete key 1 in the above tree, and afterwards delete key 7 in the resulting tree.



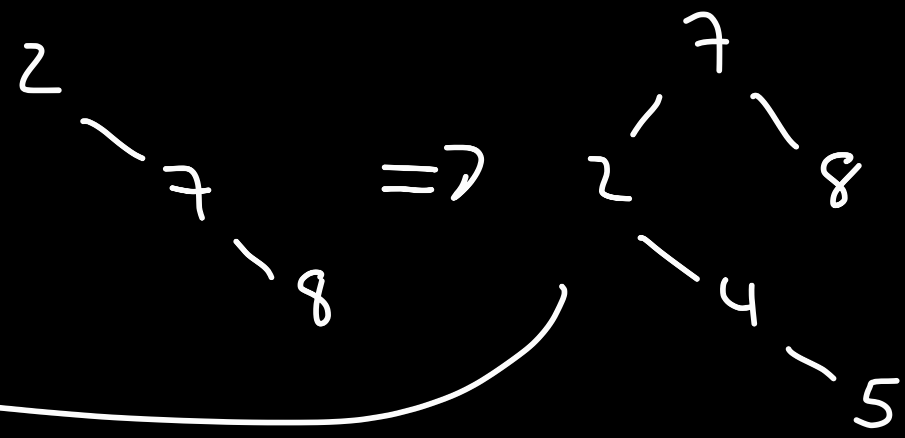
zwei Möglichkeiten:

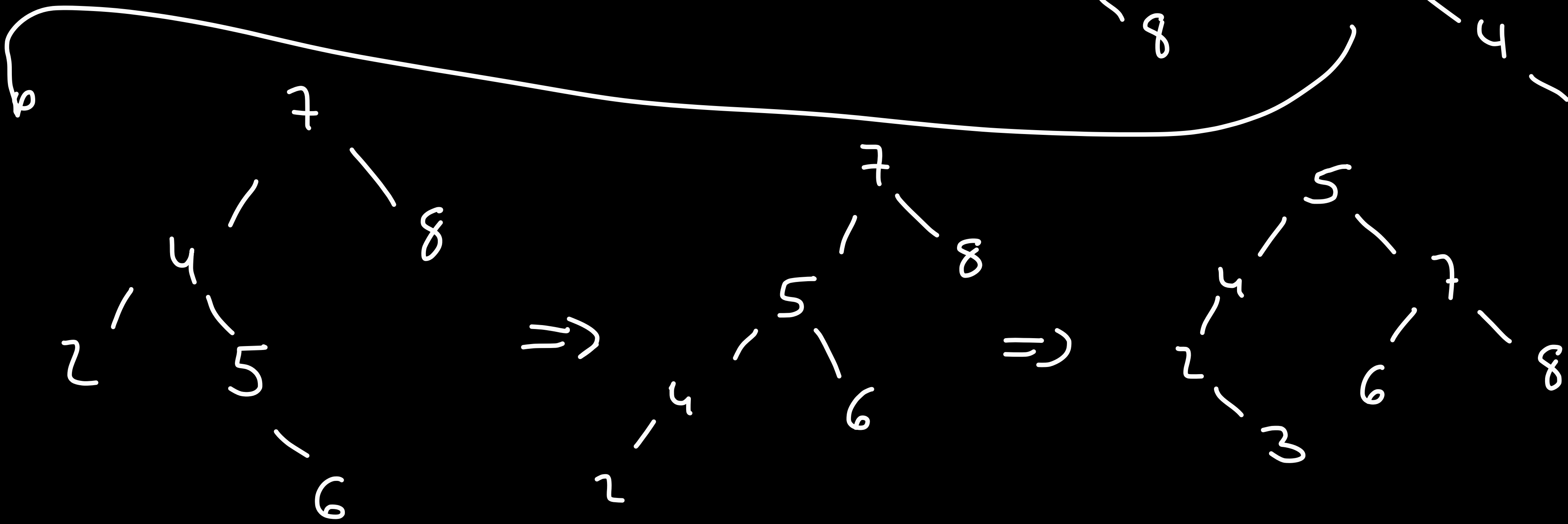


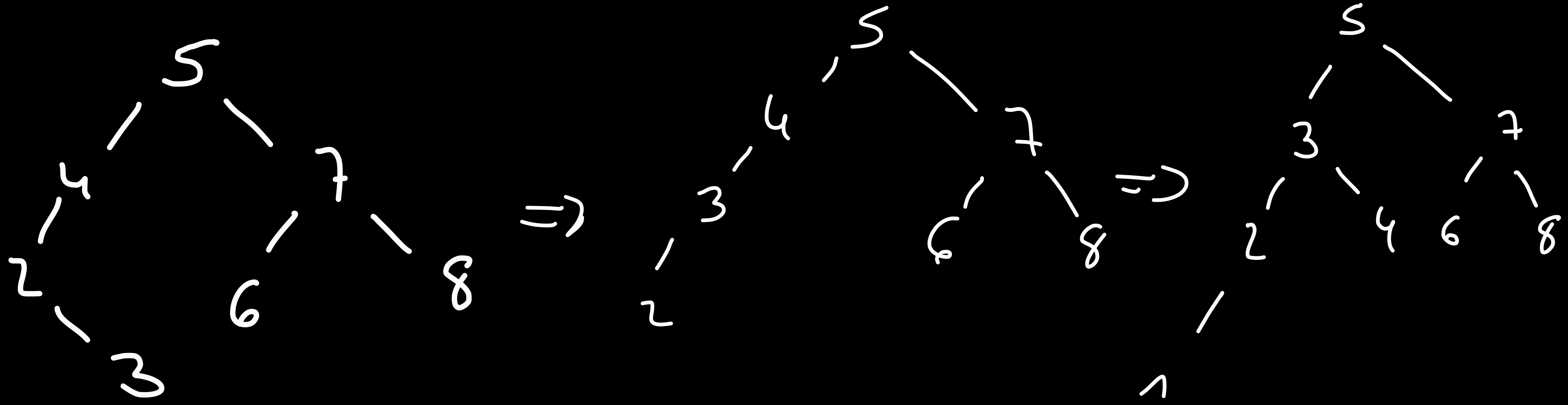


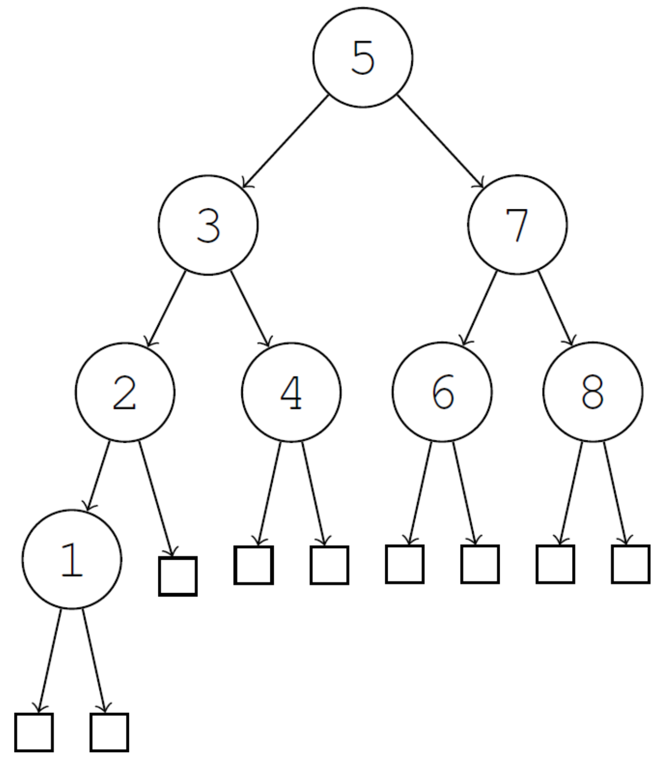
c) Draw the resulting tree when the above keys are inserted into an initially empty AVL tree. Give also the intermediate states before and after each rotation that is performed during the process.

2, 7, 8, 4, 5, 6, 3, 1

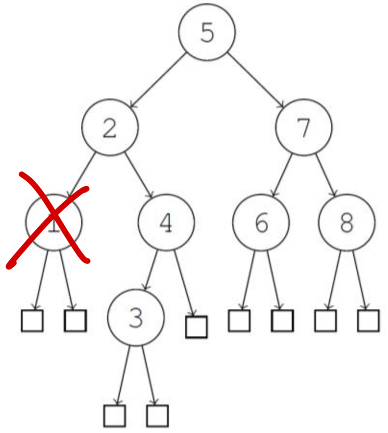




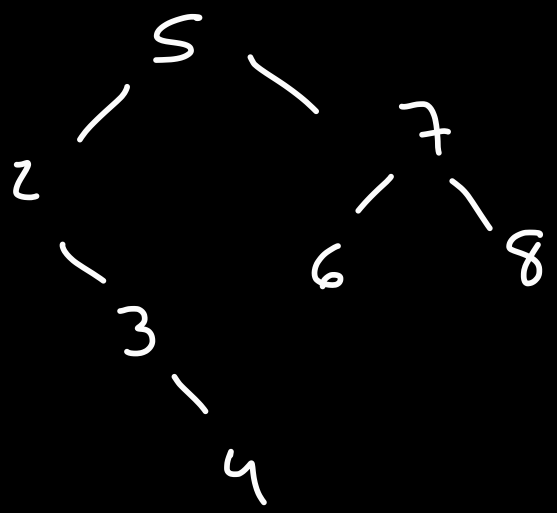



5
3
7
2
4
6
8
1

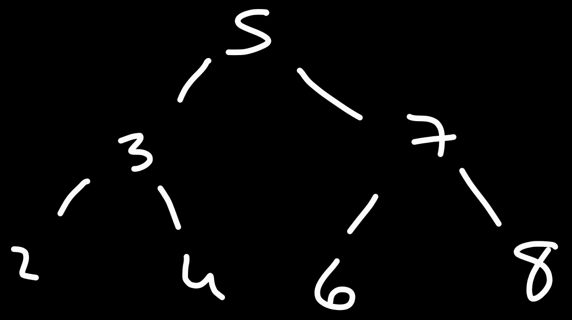
d) Consider the following AVL tree:

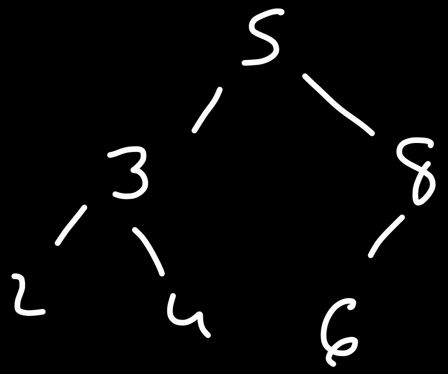


Delete key 1 in this tree, and afterwards delete key 7 in the resulting tree. Give also the intermediate states before and after each rotation is performed during the process.



=>





EXERCISE 9.4

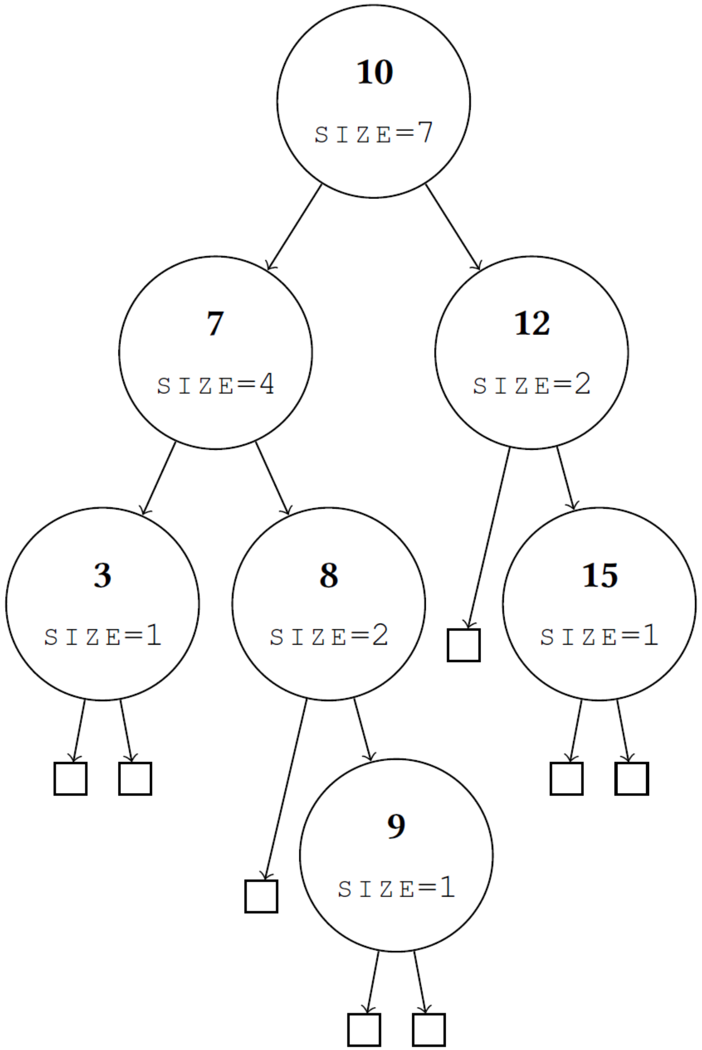


Figure 1: Augmented binary search tree

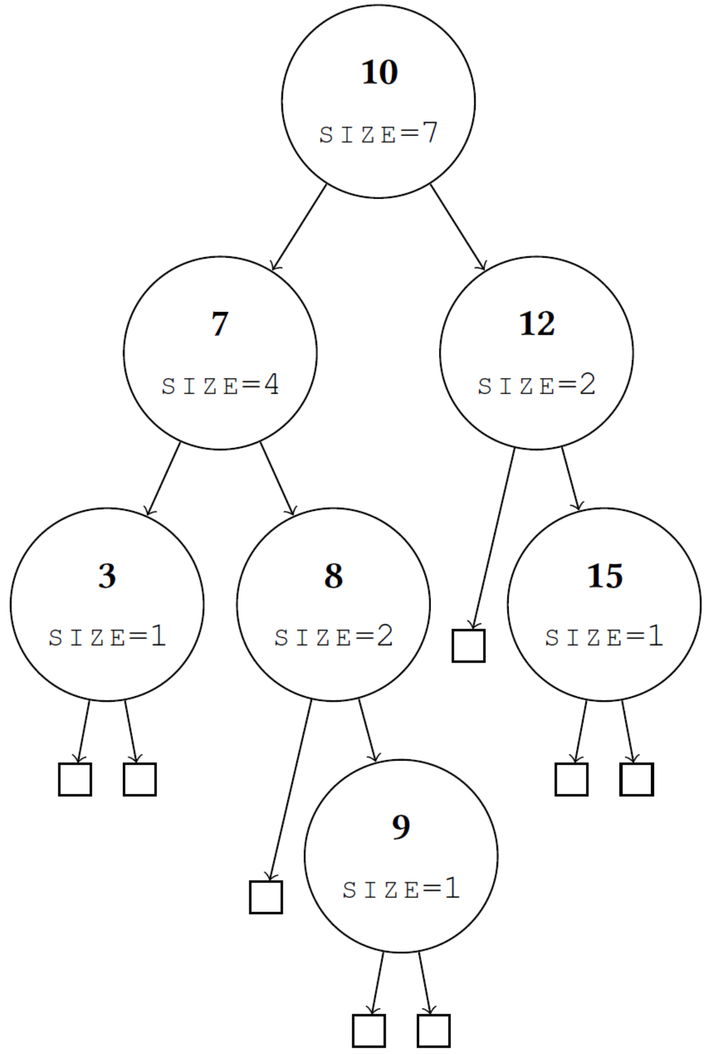


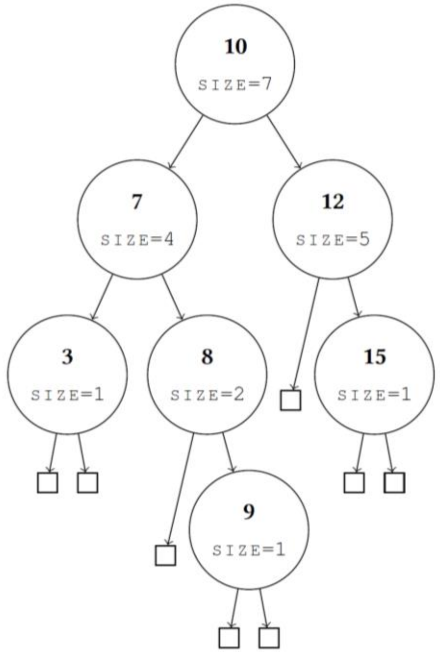
Figure 1: Augmented binary search tree

a) What is the relation between the size of a node and the sizes of its children?

NODE.SIZE = NODE.LEFT.SIZE + NODE.RIGHT.SIZE + 1

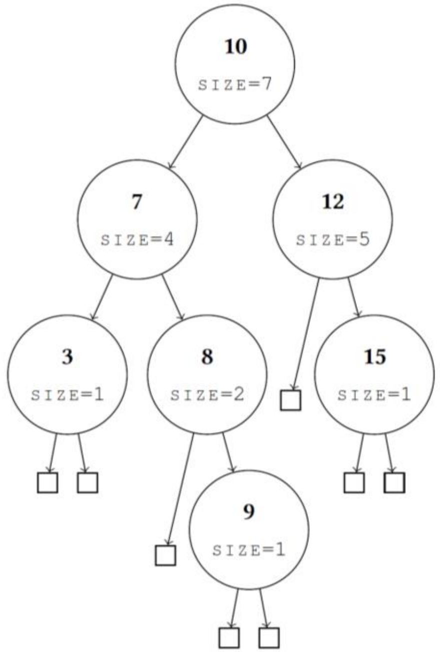
b) Describe in pseudo-code an algorithm VERIFYSIZES(ROOT) that returns TRUE if all the sizes in the tree are correct, and returns FALSE otherwise. For example, it should return TRUE given the tree in Fig. 1, but FALSE given the tree in Fig. 2.

What is the running time of your algorithm? Justify your answer.



b) Describe in pseudo-code an algorithm VERIFYSIZES(ROOT) that returns TRUE if all the sizes in the tree are correct, and returns FALSE otherwise. For example, it should return TRUE given the tree in Fig. 1, but FALSE given the tree in Fig. 2.

What is the running time of your algorithm? Justify your answer.

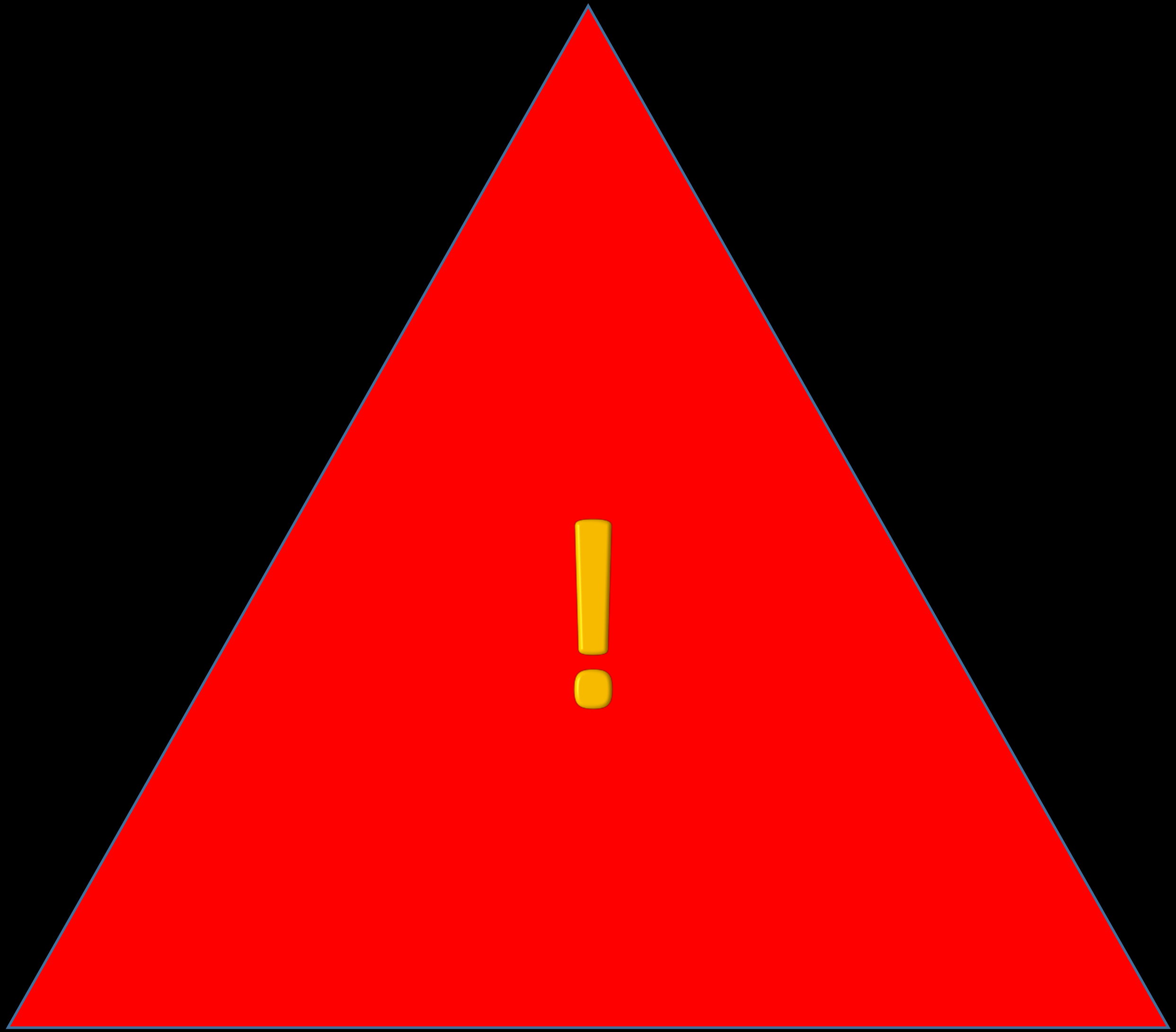


left ← 0
right ← 0
if (root.left != NULL)
    left ← root.left.size
if (root.right != NULL)
    right ← root.right.size
correct ← left + right + 1
return correct == root.size

Algorithm 1 Verifying the sizes of the tree

function VERIFYSIZES(ROOT)
    if ROOT = NULL **then**
        return TRUE
    else if VERIFYSIZES(ROOT.LEFT) = FALSE OR VERIFYSIZES(ROOT.RIGHT) = FALSE **then**
        return FALSE
    else
        CORRECTSIZE ← 1 + ROOT.LEFT.SIZE + ROOT.RIGHT.SIZE
        return CORRECTSIZE = ROOT.SIZE

$O(n)$



c) Suppose we have an augmented AVL tree (i.e., as above, each node has a SIZE member variable). Describe in pseudo-code an algorithm SELECT(ROOT, k) which, given an augmented AVL tree and an integer k, returns the k-th smallest element in the tree in $O(\log n)$ time.

c) Suppose we have an augmented AVL tree (i.e., as above, each node has a $\textsc{size}$ member variable). Describe in pseudo-code an algorithm $\textsc{Select}(\textsc{root}, k)$ which, given an augmented AVL tree and an integer k, returns the k-th smallest element in the tree in $O(\log n)$ time.

Algorithm 2 Selecting the k-th smallest element

```
function Select(root, k)
    current ← root.left.size + 1
    if k = current then
        return root.data
    else if k < current then
        return Select(root.left, k)
    else
        return Select(root.right, k − current)
```

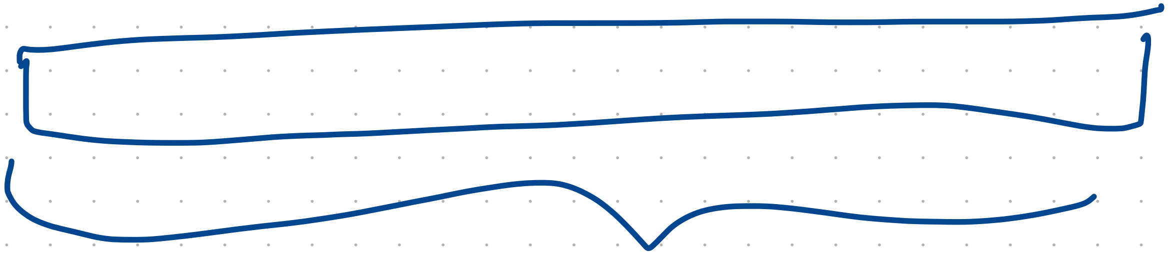
EXERCISE 9.3

Exercise 9.3 *Online supermarket.*

Assume that you work in a large online supermarket that offers different types of goods. At every moment you have to know the number of goods of each type that the supermarket currently offers. Let us denote the number of goods of type t by S_t. At any moment S_t can either be decreased (if someone has bought some goods of type t) or increased (if some goods of type t have been delivered from the manufacturer). Also your boss can ask you how many goods of type t does the supermarket currently offer. So you can receive three types of queries: to decrease S_t by $0 < x \leq S_t$, to increase S_t by $x > 0$ or to return S_t.

Assume that at each moment number of different types of goods that the supermarket offers at that moment is bounded by $n > 0$, but the number of types of goods that the supermarket can potentially offer might be much larger than n. Consider the following example: $n = 3$, at 14:00 the supermarket can offer 5 balls, 1 doll and 4 phones and at 14:15 it can offer 6 balls, 3 chairs and 12 pencils.

Provide an algorithm that can handle each query in time $\mathcal{O}(\log n)$. You may assume that initially all S_t are zero.

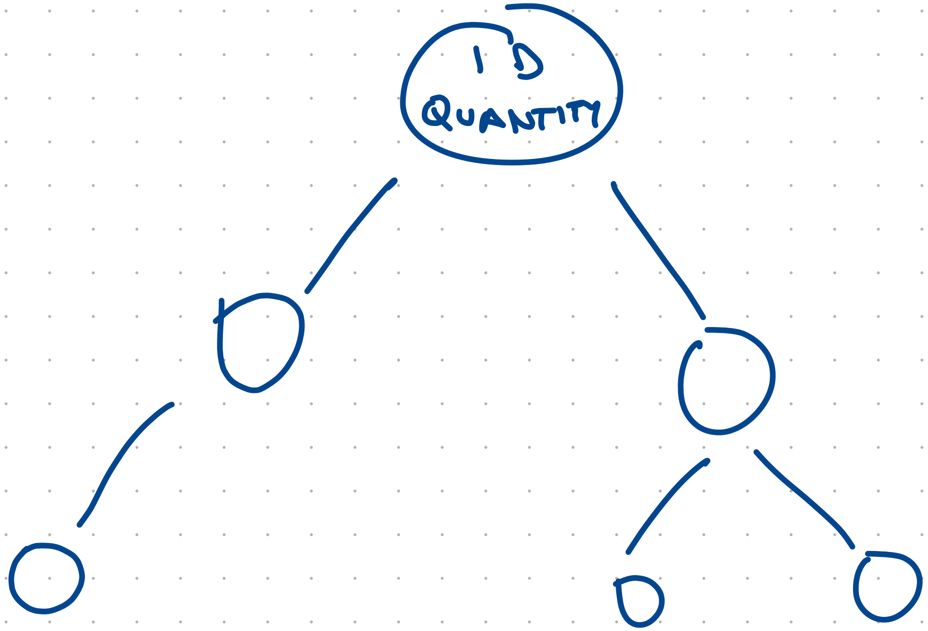


Mögliche Produkte

$O(1)$ ✓

viel Speicher ✗

AVL Baum, die das Produkt-ID als key-benutzt



$O(\log n)$

$O(n)$

GRAPH THEORY

DATA STRUCTURES FOR GRAPHS

Adjacency Matrix

Adjacency List

DATA STRUCTURES FOR GRAPHS

WHICH ONE IS SPARSE? (I.E. POTENTIALLY REQUIRES LESS MEMORY)

DATA STRUCTURES FOR GRAPHS

Adjacency Matrix

- O(n^2) space

Adjacency List

- Sparse (O(n + m) space)

DATA STRUCTURES FOR GRAPHS

WHICH ONE IS BETTER TO CHECK, WHETHER (u, v) IS AN EDGE IN G?

DATA STRUCTURES FOR GRAPHS

Adjacency Matrix

- $O(n^2)$ space
- Check edge presence in constant time

Adjacency List

- Sparse ($O(n + m)$ space)
- Check edge presence in $O(\deg(u))$ time

DATA STRUCTURES FOR GRAPHS

WHICH ONE IS BETTER IN FINDING, HOW MANY EDGES ARE GOING OUT FROM A NODE?

DATA STRUCTURES FOR GRAPHS

Adjacency Matrix

- $O(n^2)$ space
- Check edge presence in constant time
- Count outgoing edges in linear time

Adjacency List

- Sparse ($O(n + m)$ space)
- Check edge presence in $O(deg(u))$ time
- Count outgoing edges in $O(deg(u))$ time

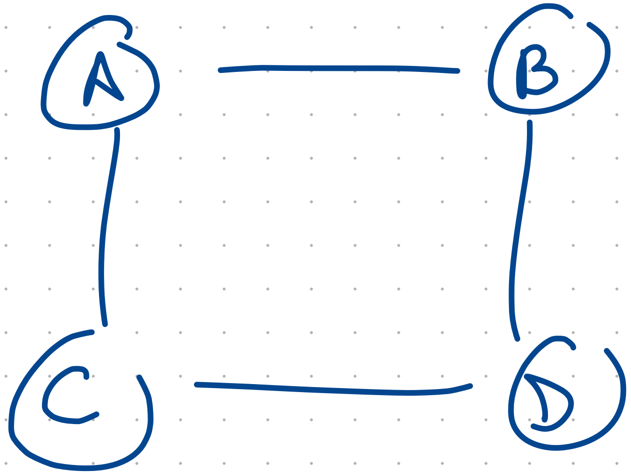
DATA STRUCTURES FOR GRAPHS

WALKS

WHICH ONE IS THE MOST USEFUL, TO DETERMINE NUMBER OF ~~PATH~~ OF LENGTH k BETWEEN TWO NODES?

Bei Adj. Matrix A

$(A^k)_{ij} \Rightarrow$ # Wege zwischen i und j von Länge genau k



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix}$$

DATA STRUCTURES FOR GRAPHS

Adjacency Matrix

- O(n^2) space
- Check edge presence in constant time
- Count outgoing edges in linear time
- Useful trick for number of paths!

Adjacency List

- Sparse (O(n + m) space)
- Check edge presence in O(deg(u)) time
- Count outgoing edges in O(deg(u)) time

DATA STRUCTURES FOR GRAPHS

IMPORTANT: DEPENDING ON THE ALGORITHM YOU WANT TO USE, IT COULD BE MORE APPROPRIATE TO USE ADJACENCY MATRIX/ LIST

HOW TO CHECK, WHETHER THE GRAPH IS CONNECTED/ THERE IS A PATH FROM u to v?

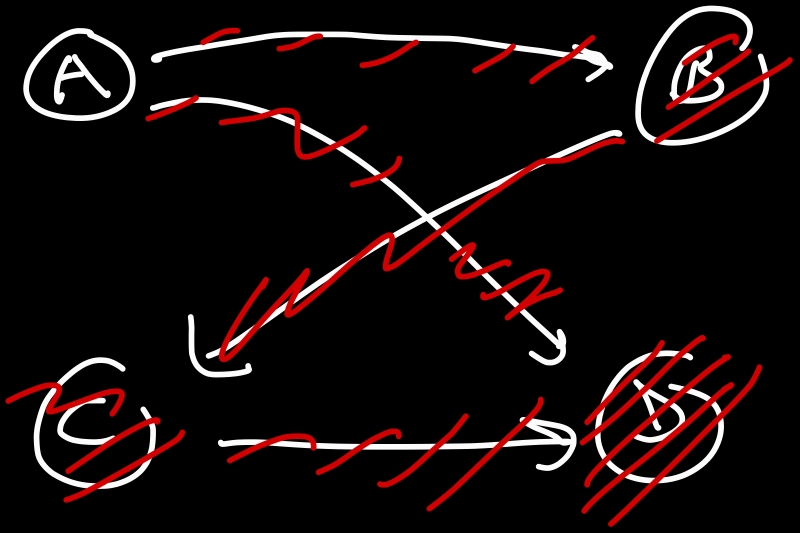
HOW TO CHECK, WHETHER THE GRAPH IS CONNECTED/ THERE IS A PATH FROM u to v?

- DFS
- BFS

TWO IMPORTANT ALGORTIHMS

DFS

- Stack
- Useful for topological sorting



BFS

- Queue
- Useful to find shortest paths (in graphs with uniform edge weights)

A , B , C , D

OLD EXAMS