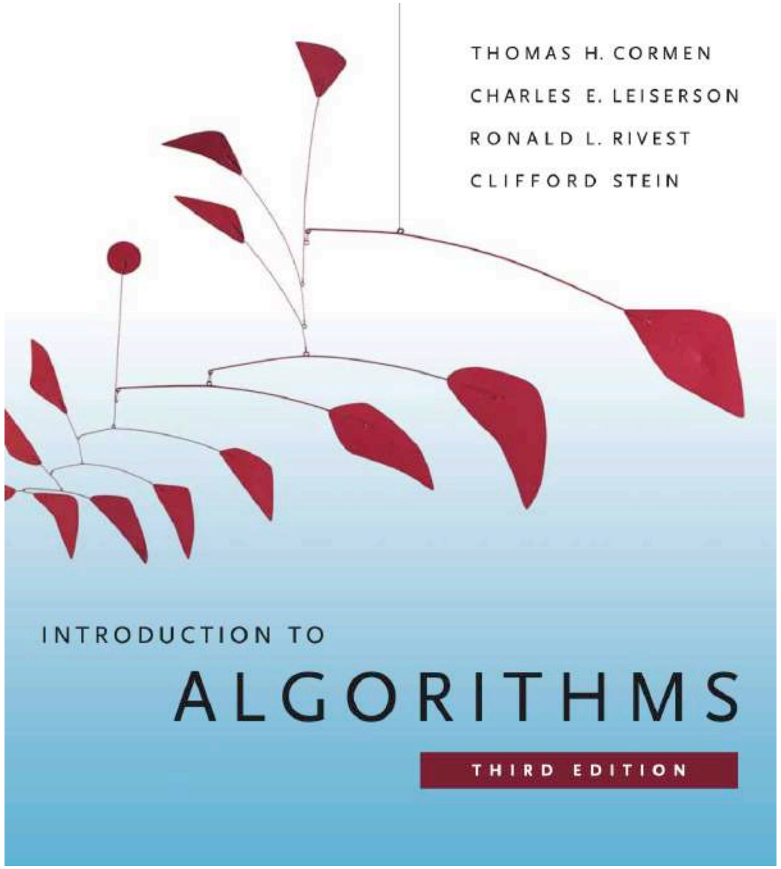
Algorithmen und Datenstrukturen

Prof. Dr. Thomas Wiemann - FB AI

Hochschule Fulda
University of Applied Sciences

Hochschule Fulda University of Applied Sciences

Inhalt

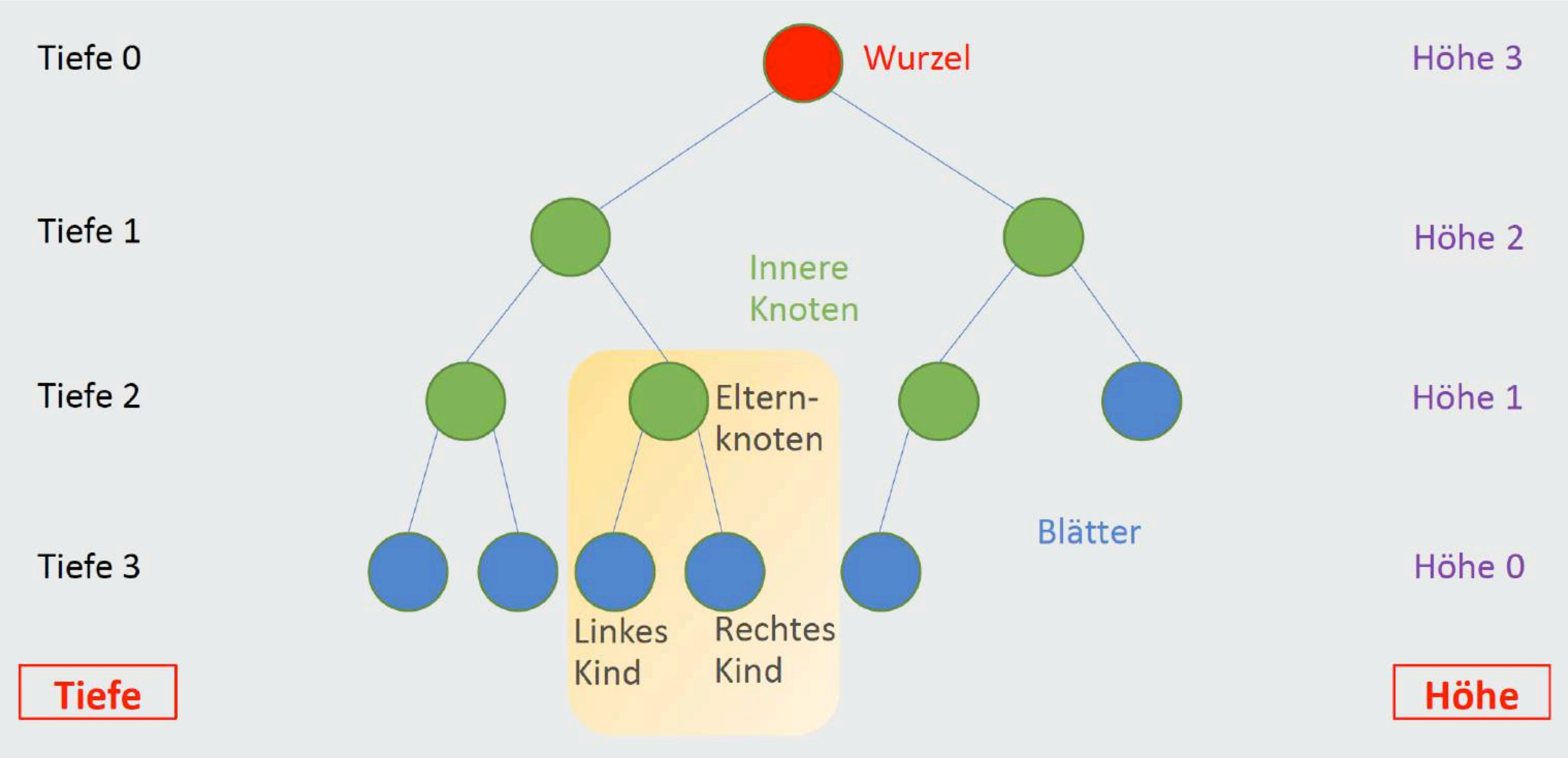


Gliederung

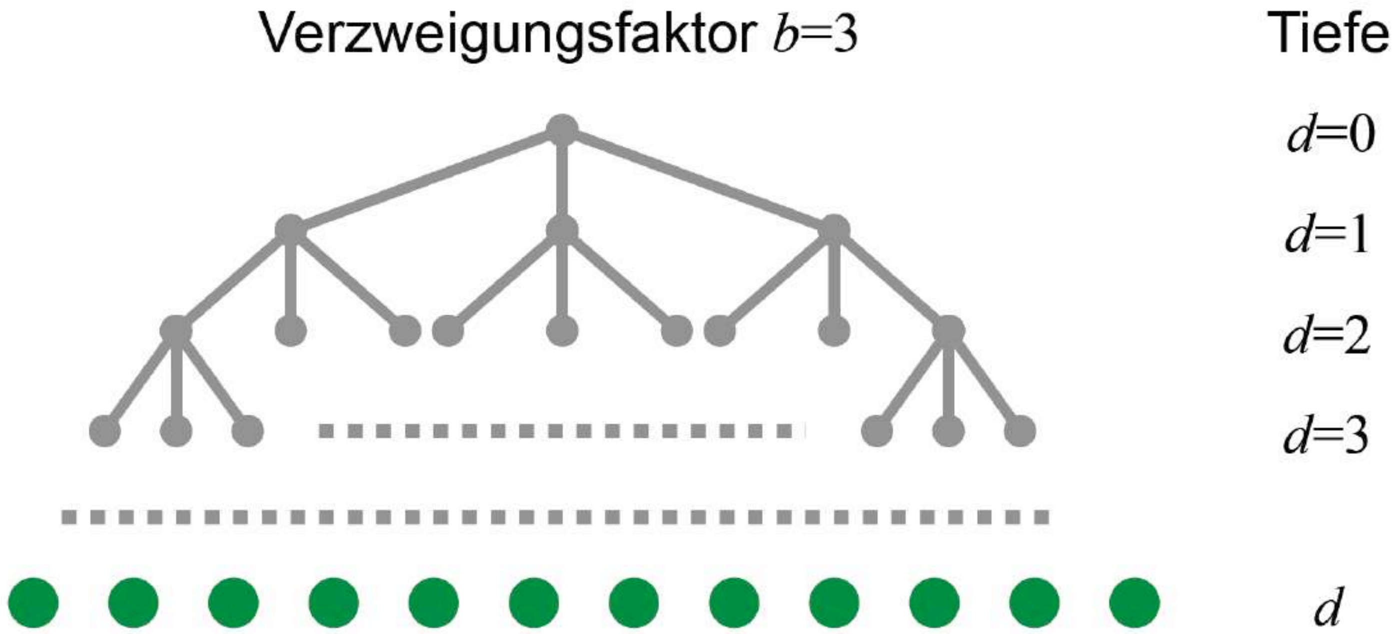
1. Laufzeit und Komplexität
2. Sortieren
3. Abstrakte Datentypen
4. Hashing
5. **Suchbäume**
6. Graphen- und Graphenalgorithmen
7. Ausblick

https://edutechlearners.com/download/Introduction_to_algorithms-3rd Edition.pdf

Erinnerung: Bäume - Begriffe



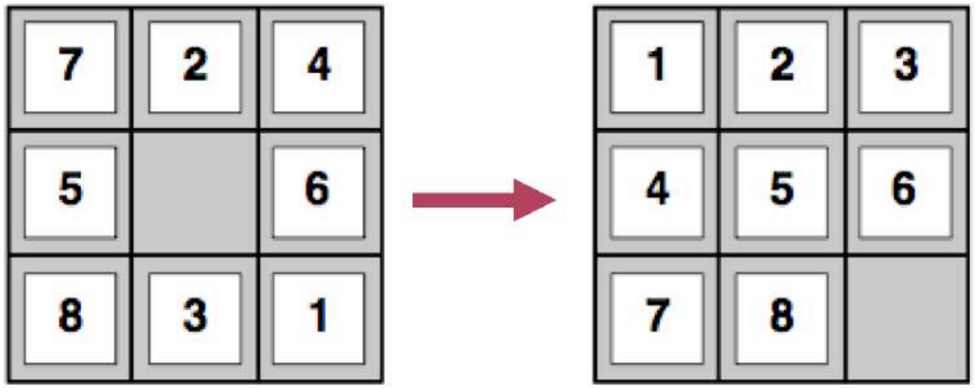
Verweigungsfaktor, Knoten und Blätter



$O(b^d)$ # Knoten der Tiefe d: b^d

$O(b^d)$ # alle Knoten bis einschl. Tiefe d: $\sum_{i=0}^{d} b^i = \frac{b^{d+1}-1}{b-1}$

Beispiel: Verschiebespiel



Startzustand **Zielzustand**

Zustand

Sequenz der Zahlen/ Leerfeld auf den 9 Feldern

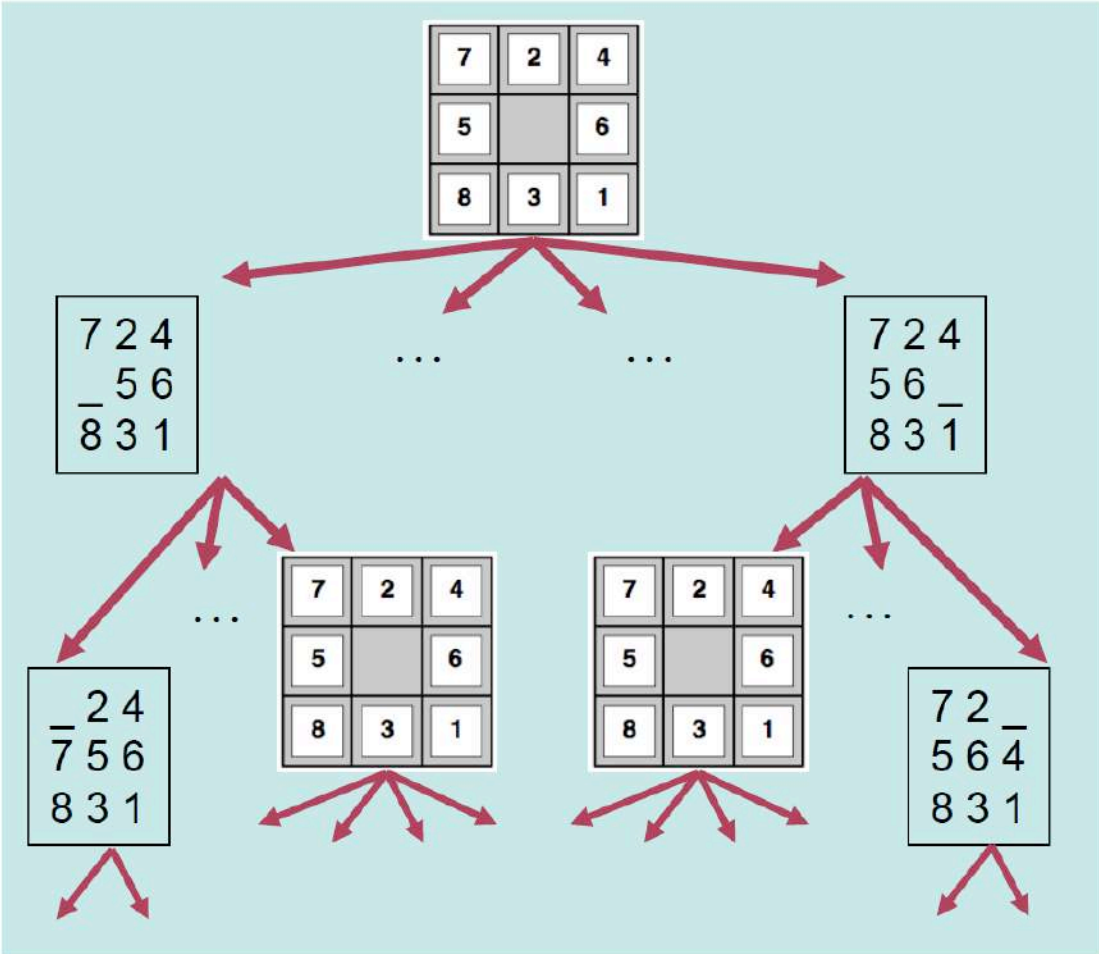
Aktionen

Left, Right, Up, Down (Verschiebung des Leerfelds, wenn‘s geht)

Kosten

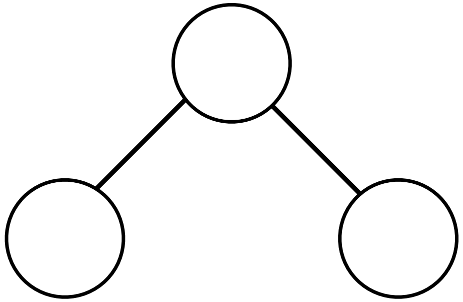
Konstant (1) pro Aktion

Beispiel: Verschiebespiel



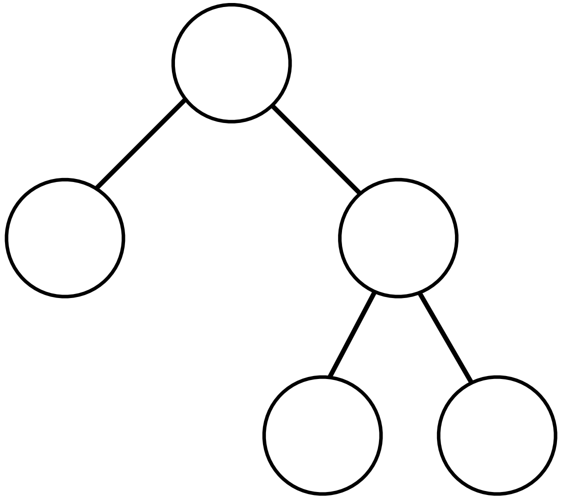
Binärbaum

- Jeden Knoten sind maximal zwei binäre Unterbäume zugeordnet



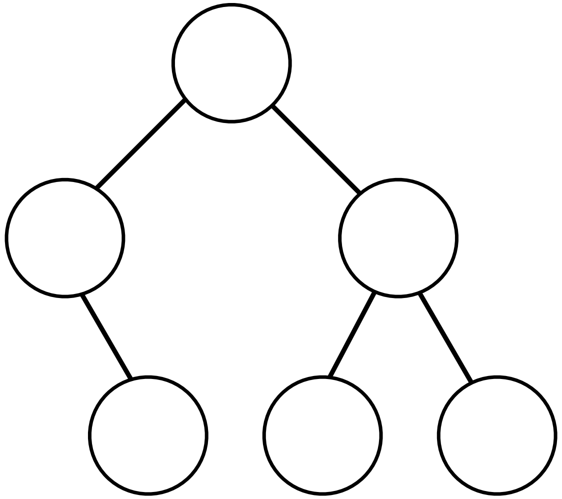
Binärbaum

- Jeden Knoten sind maximal zwei binäre Unterbäume zugeordnet



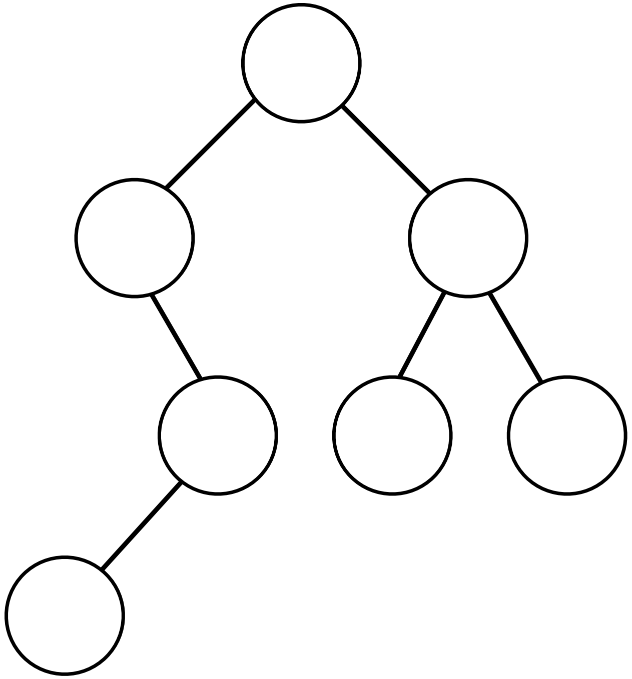
Binärbaum

- Jeden Knoten sind maximal zwei binäre Unterbäume zugeordnet

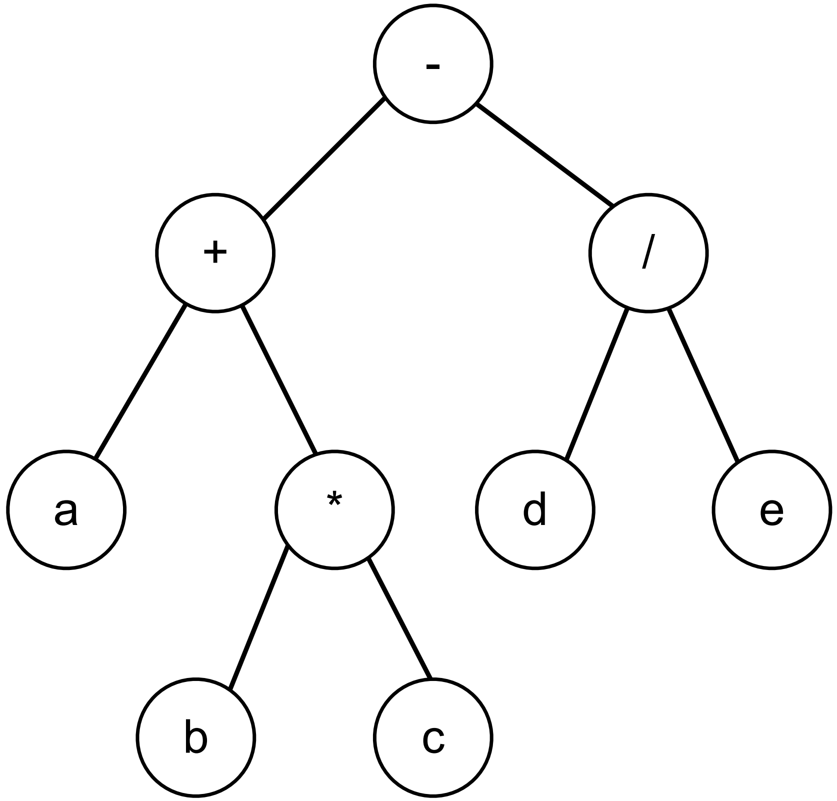


Binärbaum

- Jeden Knoten sind maximal zwei binäre Unterbäume zugeordnet



Traversierungen (im Binärbaum)

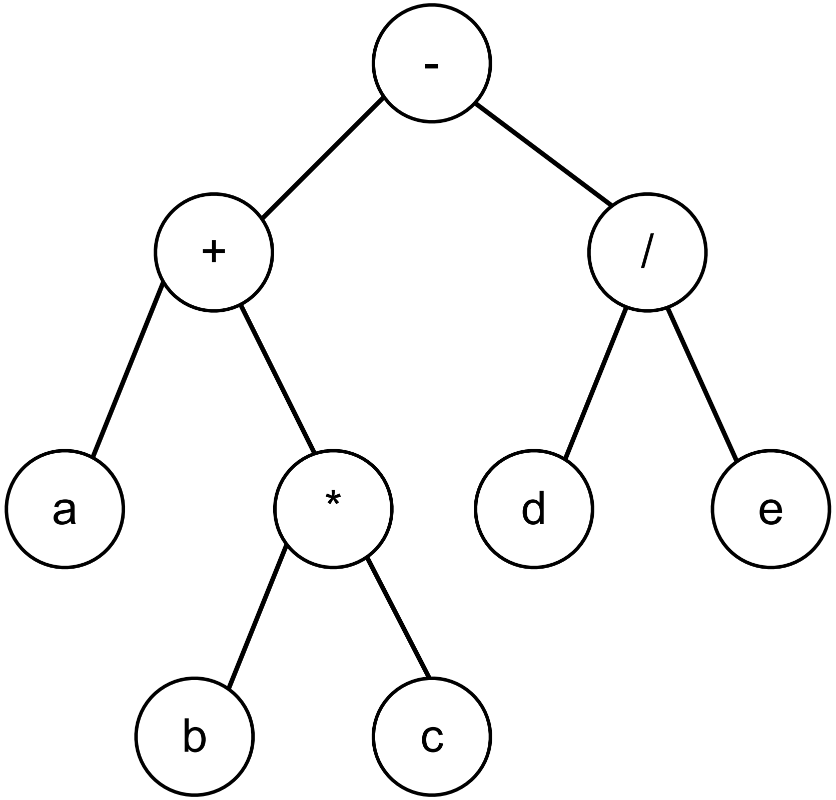


Inorder:

“links Vater rechts“

a + b * c - d / e

Traversierungen

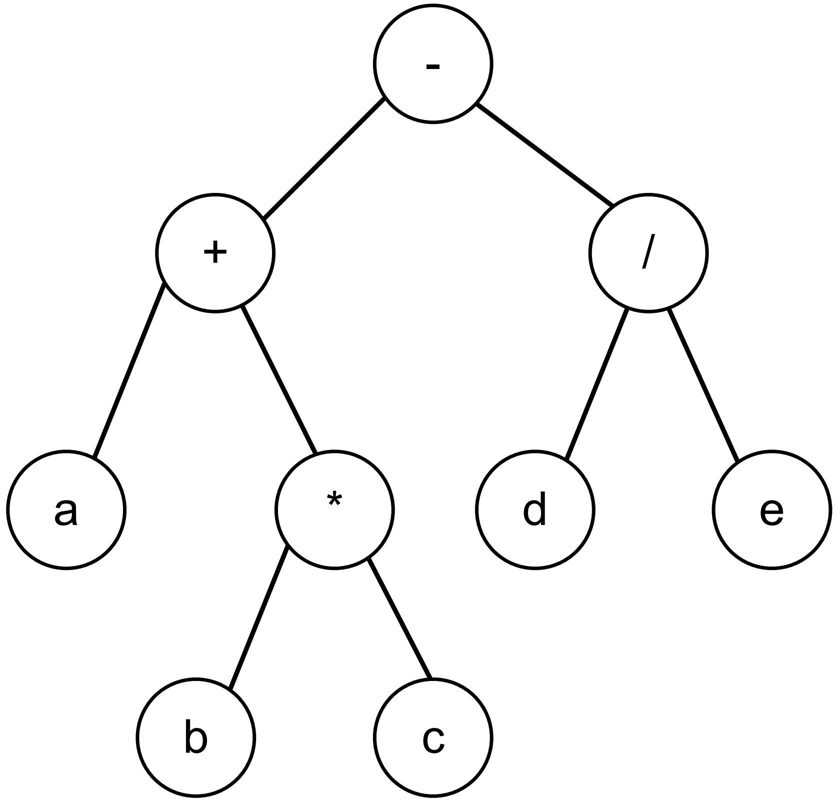


Preorder:

“Vater links rechts“

- + a * b c / d e

Traversierungen



Postorder:

“links rechts Vater“

a b c * + d e / -

Inorder in Java

```
public class Node
{
    public Node left;
    public Node right;
    public Comparable data;
    public bool empty();
    Node(Comparable x)
    {
        data = x;
        left = null;
        right = null;
    }
}
```

```
public static void inorder(Node n) {
    if(!n.empty()){
        inorder(n.left);
        System.out.println(n.data.toString());
        inorder(n.right);
    }
}
```

Preorder in Java

```
public class Node
{
    public Node left;
    public Node right;
    public Comparable data;
    public bool empty();
    Node(Comparable x)
    {
        data = x;
        left = null;
        right = null;
    }
}
```

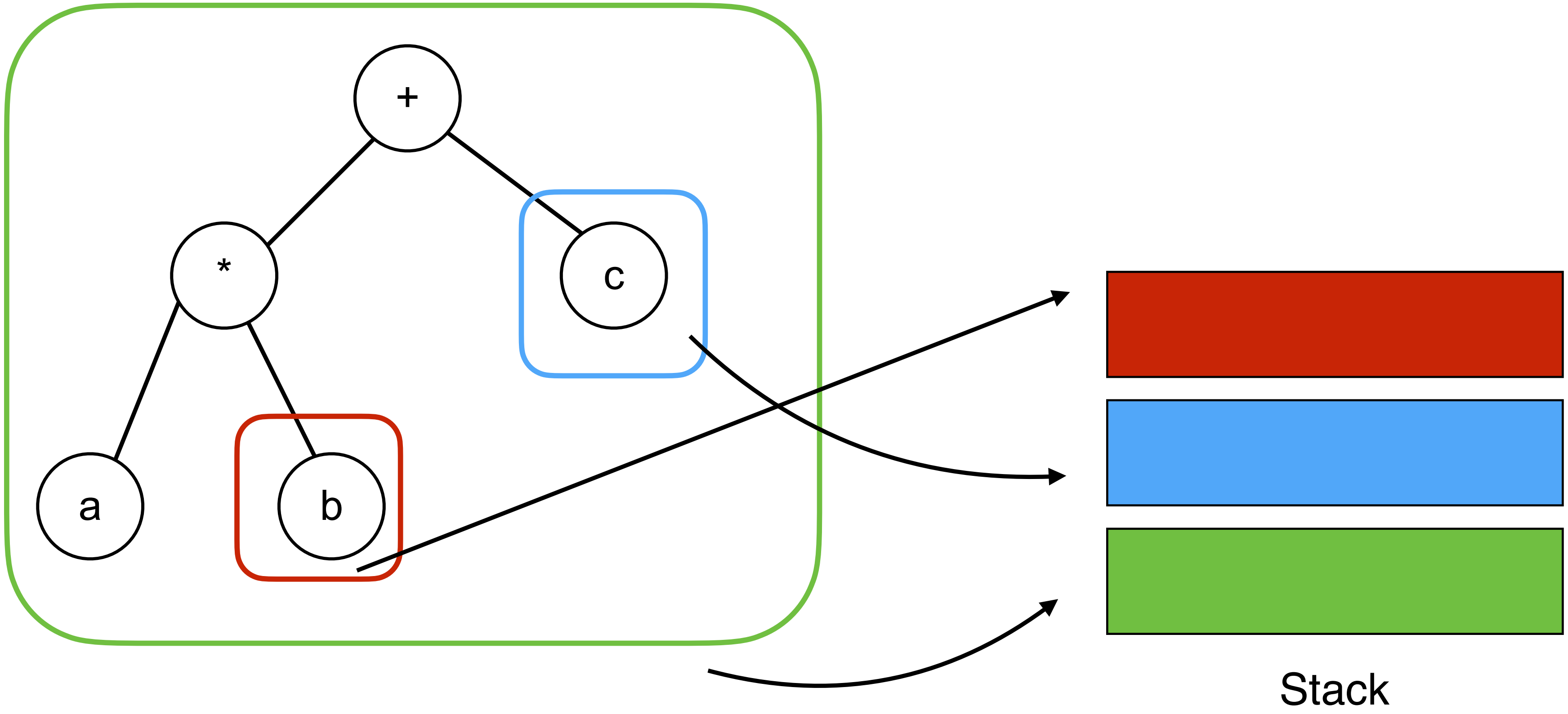
```
public static void preorder(Node n) {
    if(!n.empty()) {
        System.out.println(n.data.toString());
        preorder(n.left);
        preorder(n.right);
    }
}
```

Postorder in Java

```
public class Node
{
    public Node left;
    public Node right;
    public Comparable data;
    public bool empty();
    Node(Comparable x)
    {
        data = x;
        left = null;
        right = null;
    }
}
```

```
public static void postorder(Node n) {
    if(!n.empty()) {
        postorder(n.left);
        postorder(n.right);
        System.out.println(n.data.toString());
    }
}
```

Pre-order ohne Rekursion

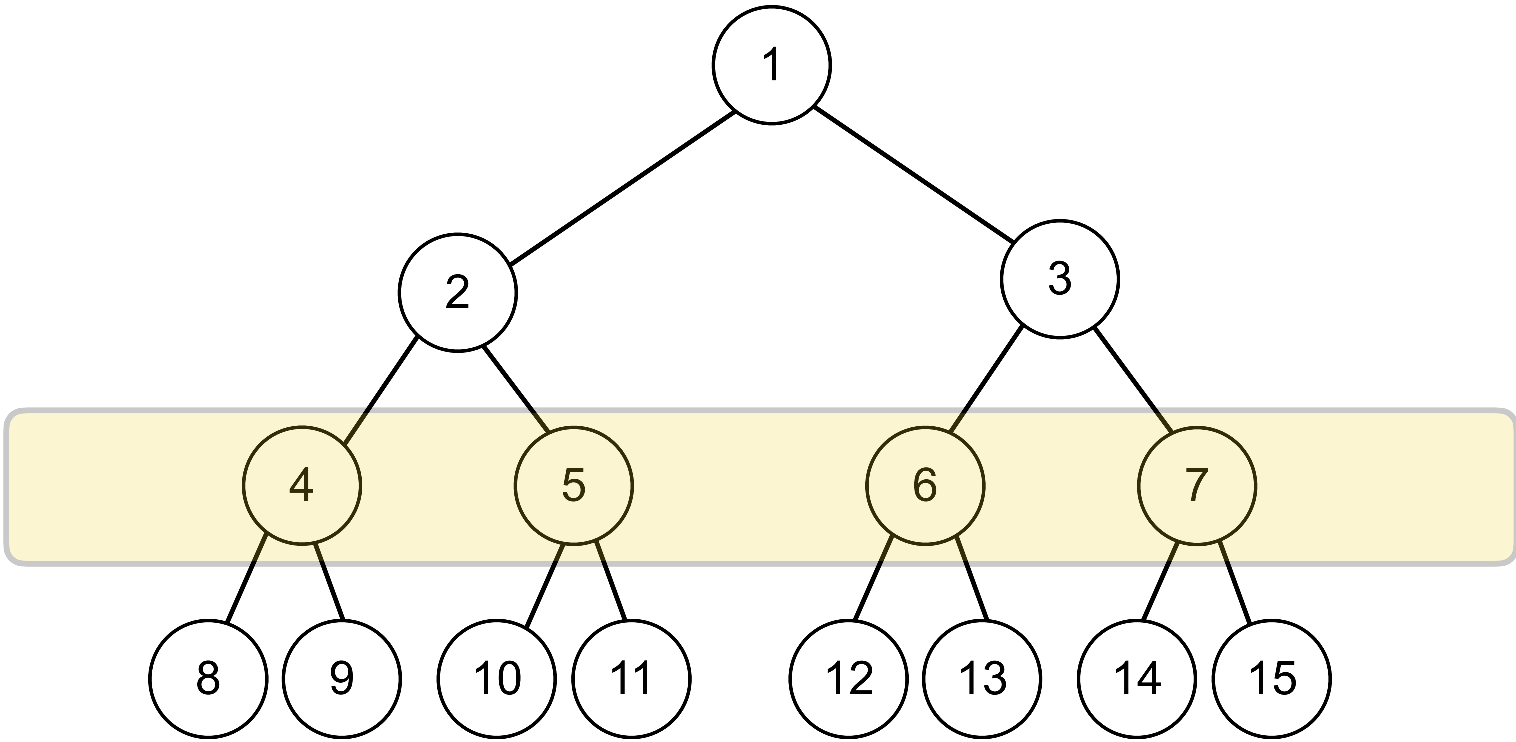


Hochschule Fulda
University of Applied Sciences

Iterative Tiefensuche (Pseudo-Java)

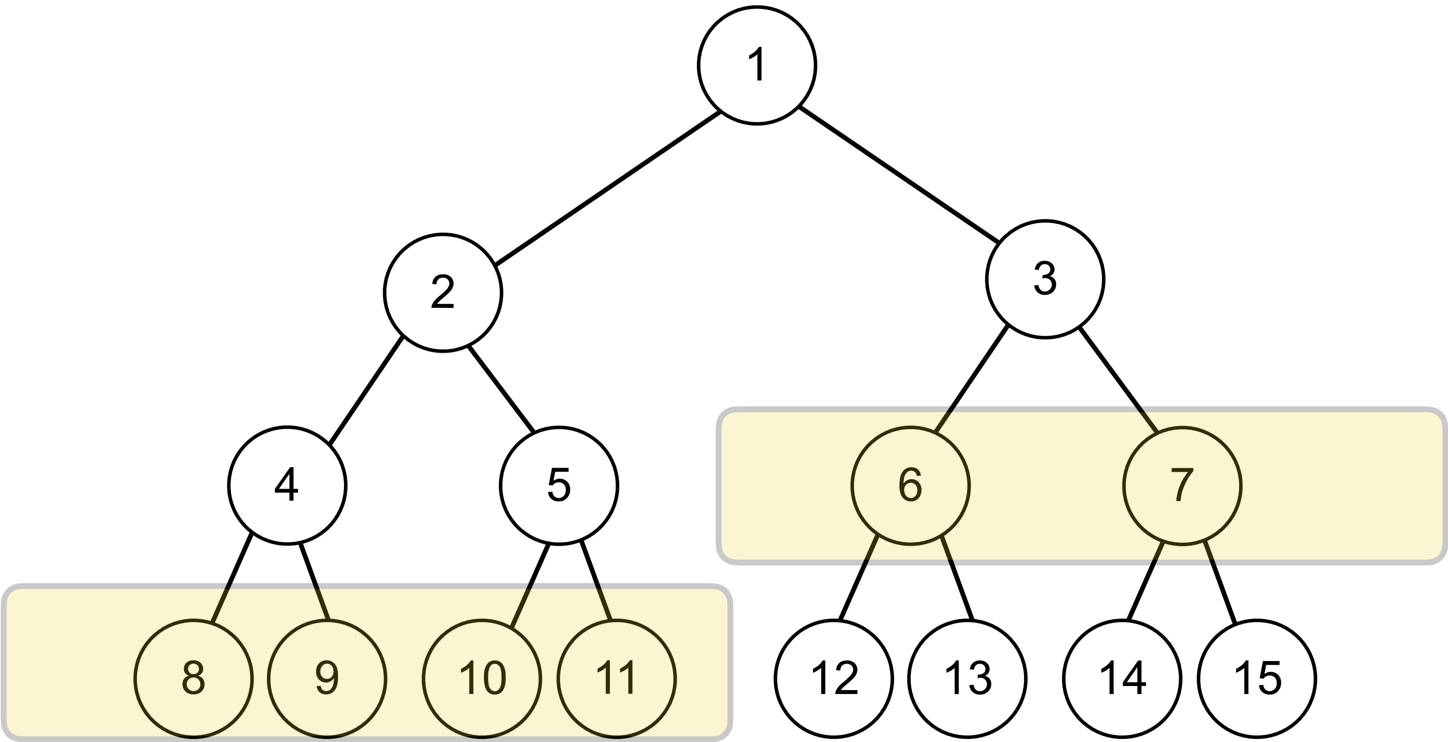
```
public static void DFS(Node w) {
    Node t;
    Stack s = new Stack();
    if(!w.empty()) s.push(w);
    while(!s.empty())
    {
        t = s.top(); s.pop();
        do {
            System.out.println(t.data.toString());
            if(!t.right.empty()) s.push(t.right);
            t = t.left;
        } while (!t.empty())
    }
}
```

Breitensuche



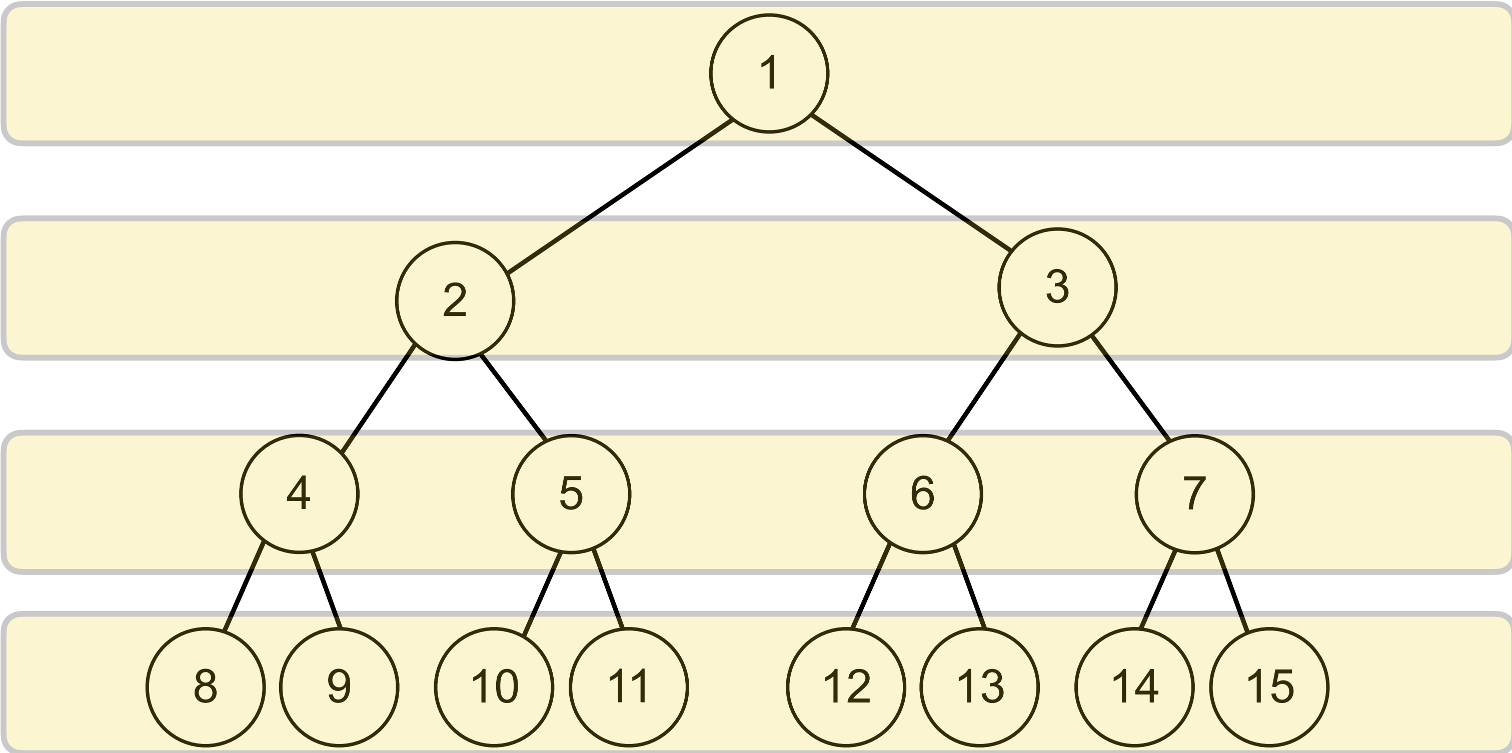
Umsetzung mittels Queue

Breitensuche



Umsetzung mittels Queue

Breitensuche



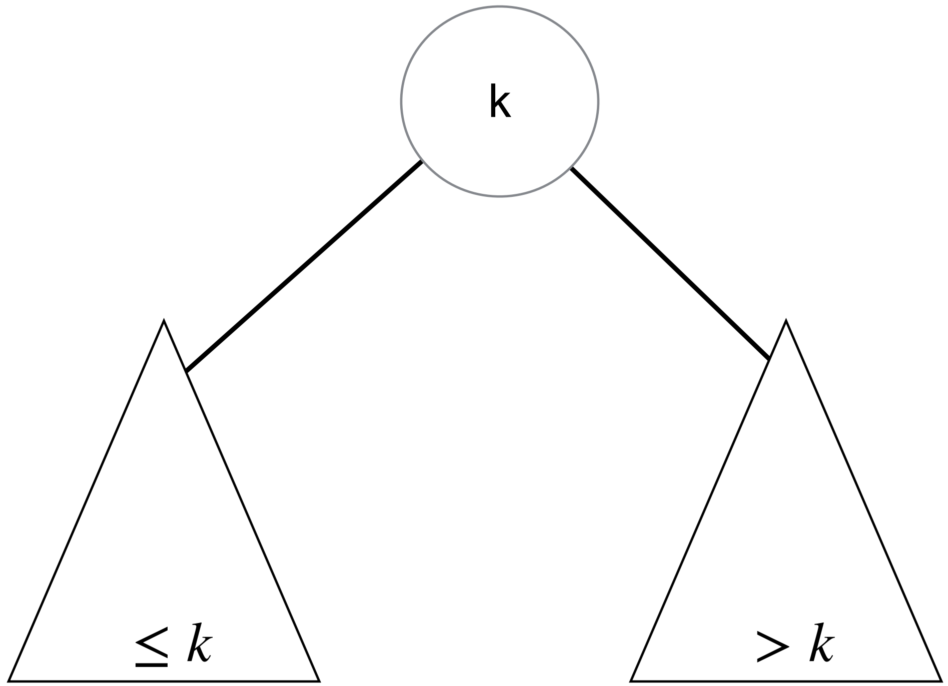
Reihenfolge der Traversierung: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Breitensuche in Pseudo-Java

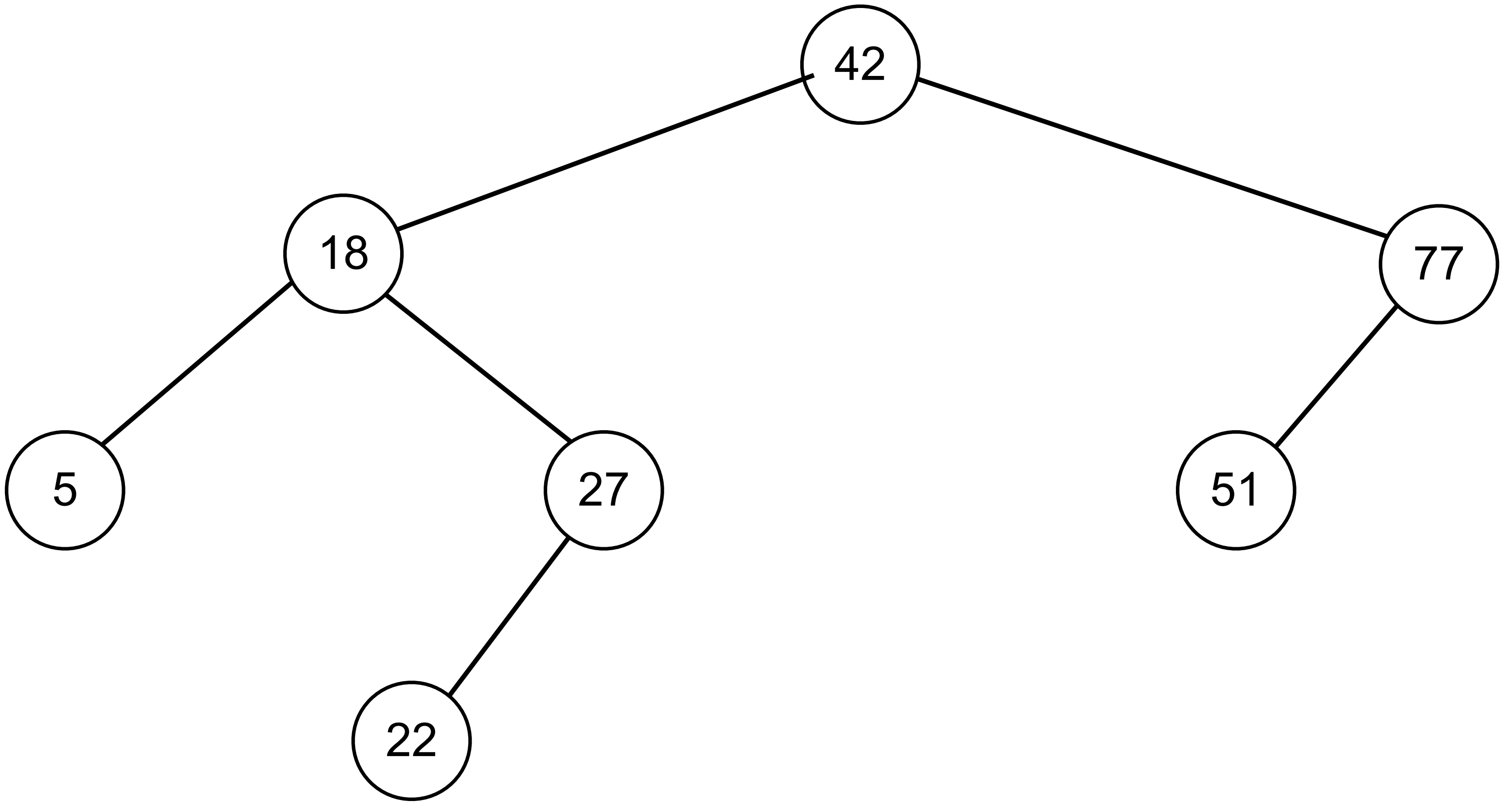
```
public static void BFS(Node w) {
    Node t;
    Queue q = new Queue();
    if(!w.empty()) q.enq(w);
    while(!q.empty()) {
        t = q.front(); q.deq();
        System.out.println(t.data.toString());
        if(!t.left.empty()) q.enq(t.left);
        if(!t.right.empty()) q.enq(t.right);
    }
}
```

Binärer Suchbaum

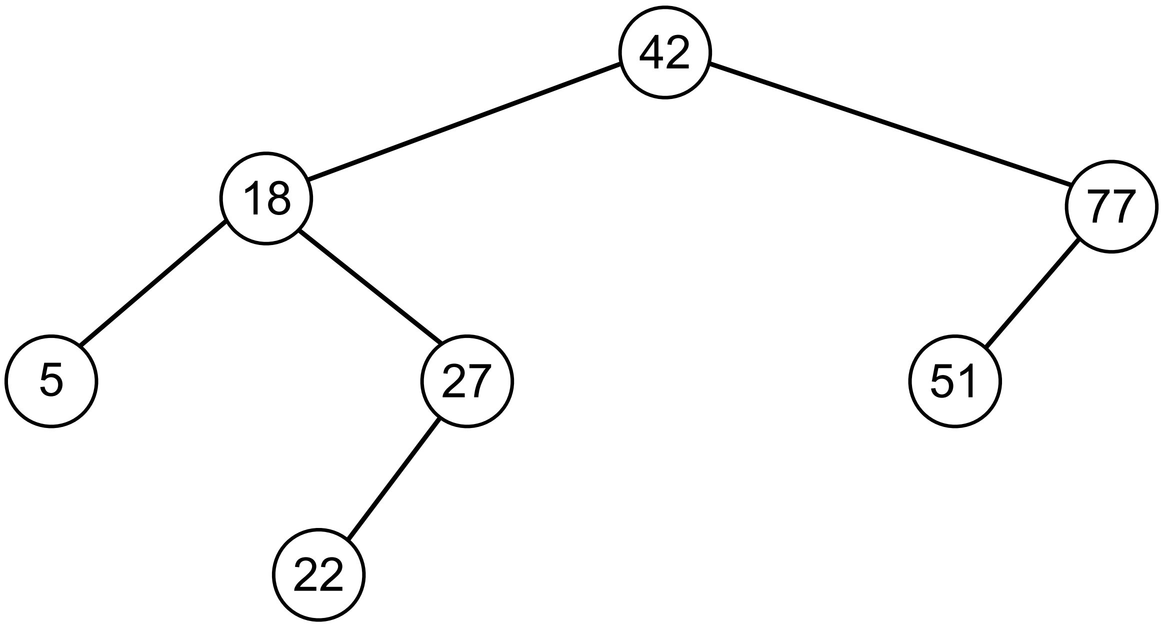
- Ein binärer Suchbaum ist ein binärer Baum mit vergleichbaren Objekten
- Alle Schlüssel im linken Teilbaum unter Knoten k sind kleiner gleich der Schlüssel in k
- Alle Schlüssel im rechten Teilbaum sind größer als k



Binärer Suchbaum



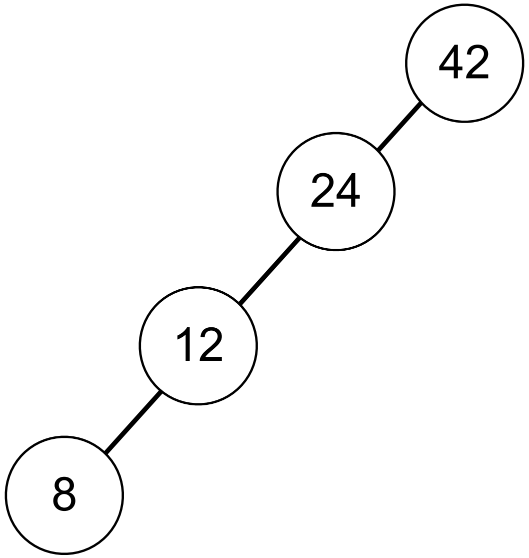
Interface Suchbaum

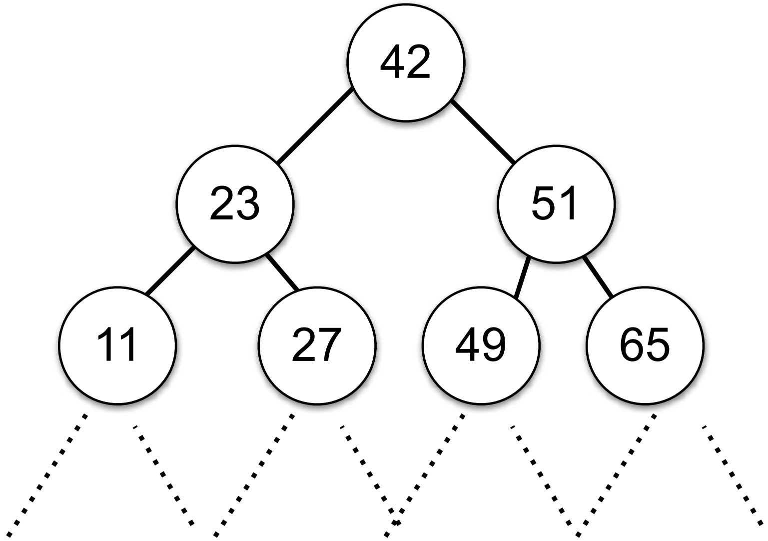


- empty()
- lookup(x)
  - Zielgerichtet nach dem Schlüssel suchen und das Objekt (oder null) zurückgeben
- insert(x)
  - Zielgerichtet absteigen. Falls gefunden, Fehlanzeige. Falls nicht, einhängen
- delete(x)
  - Zielgerichtet absteigen. Falls gefunden löschen. Sonst Fehlanzeige

Analyse Laufzeiten

- Aufwand





Worst Case $\mathcal{O}(n)$

Bei guter Balanciertheit $\mathcal{O}(\log n)$

Suchbaum - Lookup

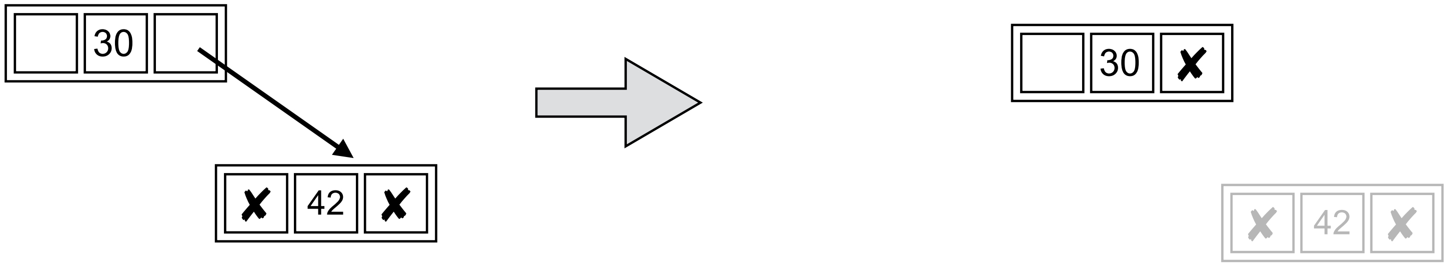
```
public Comparable lookup(Comparable x)
{
    Node k = root;
    while(k != null) {
        if(x.compareTo(k.data) < 0) {
            k = k.left;
        } else if(x.compareTo(k.data) > 0){
            k = k.right;
        } else if(x.compareTo(k.data) == 0) {
            return k.data;
        }
    }
    return null;
}
```

Suchbaum - Insert

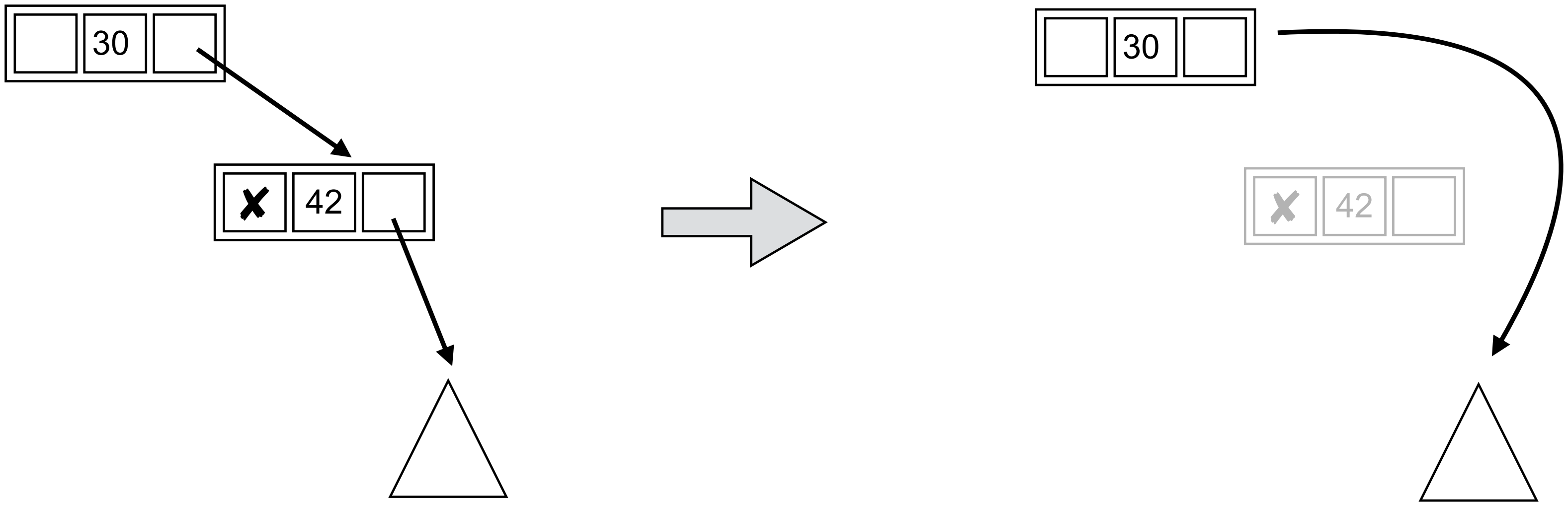
```
public boolean insert(Comparable x) {
    if(root == null)
    {
        root = new node(x);
        return true;
    } else {
        Node k = root;
        Node parent = null;
        while(k != null) {
            parent = k;
            if(x.compareTo(k.data) < 0)
            {
                k = k.left;
            } else if (x.compareTo(k.data) > 0) {
                k = k.right;
            } else {
                return false;
            }
        }
        if(x.compareTo(parent.data) < 0) {
            parent.left = new Node(x);
        } else {
            parent.right = new Node(x);
        }
        return true;
    }
}
```

Binärer Suchbaum - Delete (1)

1.) Schlüssel ist im Blatt



1.) Schlüssel in einem Knoten mit nur einem Sohn



Binärer Suchbaum - Delete (2)

3.) Schlüssel in Knoten mit zwei Söhnen → “Inorder Successor”