Hochschule Offenburg
University of Applied Sciences

Algorithmen und Datenstrukturen
6. Bäume

Prof. Dr. Klaus Dorer

Übersicht

- Bäume
  - Natürliche Bäume
  - Balancierte Binärbäume
    - AVL Bäume

Ziele

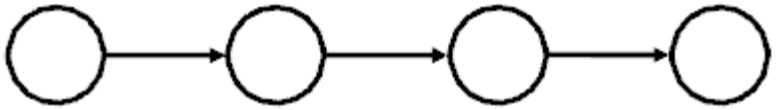
- Datenstruktur-Operationen implementieren können
- Komplexität der Operationen kennen

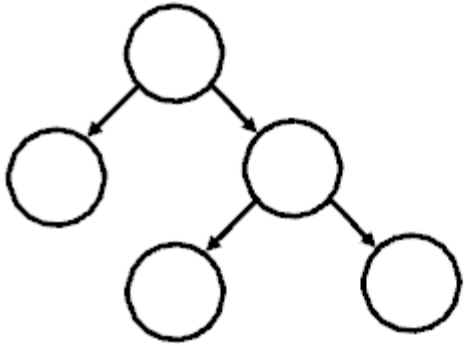
Quellen

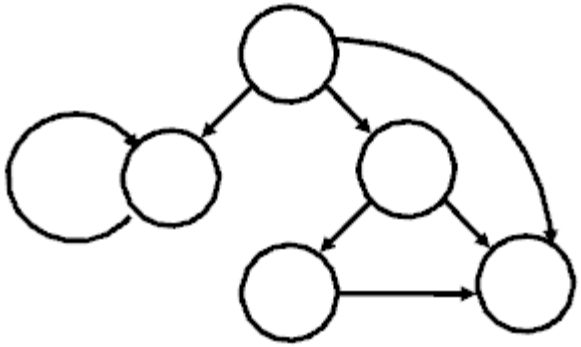
- Thomas Ottmann und Peter Widmayer, Algorithmen und Datenstrukturen, 4. Auflage, Spektrum, Berlin, 2002
- www.wikipedia.de
- Daniel Fischer, Algorithmen und Datenstrukturen, 2006
- Stefan Trahasch, Vorlesungsfolien Algorithmen und Datenstrukturen, Hochschule Offenburg

Datenstrukturen

- Liste (Kapitel 2)
  - Jeder Knoten hat einen Nachfolger
- Baum (Kapitel 6)
  - Jeder Knoten (außer Wurzel) hat einen Vorgänger
  - Ein Knoten kann mehrere Nachfolger haben
- Graph (Kapitel 8)
  - Ein Knoten kann mehrere „Vorgänger“ haben, Zyklen sind möglich





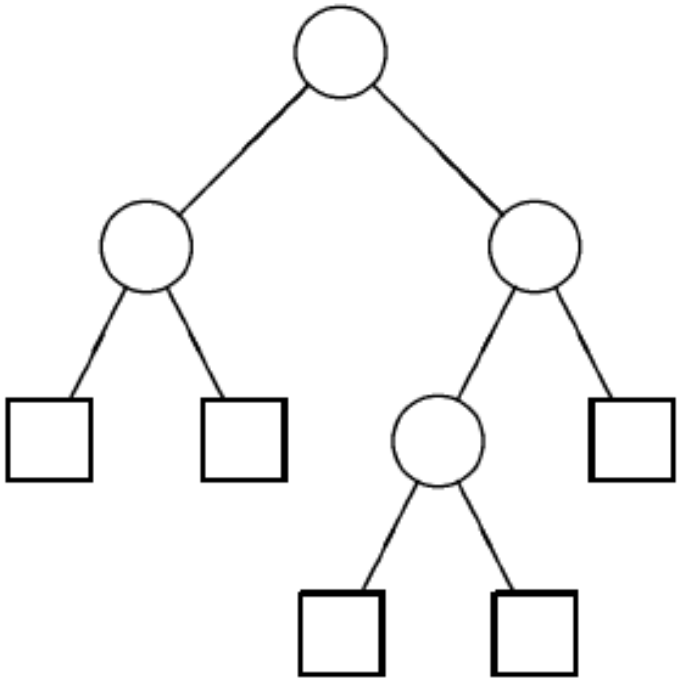


Anwendungen

- Listen
  - Rechnungspositionen
  - Highscores
- Bäume
  - Familienstammbuch
  - Ergebnisse eines Sportturniers („KO-System“)
  - Organigramm
  - Gliederung eines Buches
  - Datei-Struktur im Rechner
  - Syntaxbäume
- Graphen
  - Straßennetz

Begriffe

- Baum
  - Liste deren Elemente mehr als einen Nachfolger haben können
- Knoten
  - Element mit einem Vorgänger (Vater, parent) und >= 0 Nachfolgern (Kind, child)
- Wurzel
  - Einziger Knoten ohne Vorgänger
- Blatt
  - Knoten ohne Nachfolger
- Innerer Knoten
  - Knoten, der kein Blatt ist
- Pfad
  - Folge von Knoten $p_0..p_k$, wobei jeder p_i Vorgänger von p_{i+1} ist
  - Pfadlänge: k
- Tiefe (eines Knotens)
  - Pfadlänge eines Knotens bis zur Wurzel
- Ordnung (Breite) b
  - Maximale Anzahl Nachfolger eines Knotens (Rang)
  - Binärbaum: b = 2

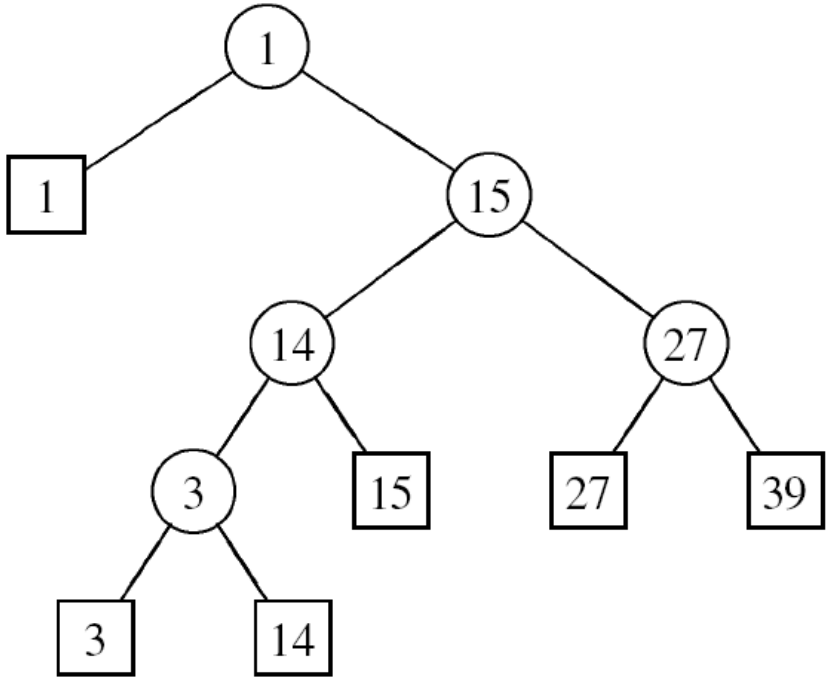


Bäume

- Zentrale Operationen
  - Einfügen(element) fügt Element im Baum ein
  - Entfernen(element) löscht Element aus dem Baum
  - Suchen(element) sucht das Element im Baum
  - Durchlaufen des Baums in bestimmter Reihenfolge
- Weitere Operationen
  - Aufspalten eines Baums in mehrere
  - Zusammenfügen mehrerer Bäume
  - Konstruieren eines Baums mit bestimmten Eigenschaften

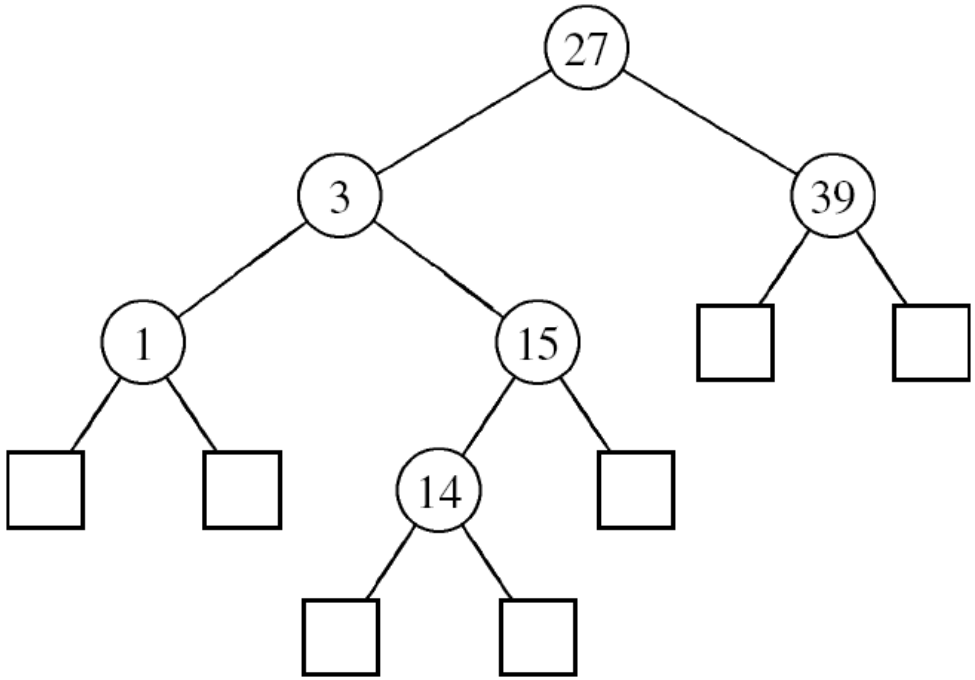
Blattsuchbäume

- Blattknoten enthalten Schlüssel
- Innere Knoten enthalten Wegweiser
  - Beispiel: maximaler Schlüssel im linken Teilbaum
  - Alle Schlüssel im linken Teilbaum sind kleiner oder gleich
  - Alle Schlüssel im rechten Teilbaum sind größer
- Suchen(p, x)
  - p ist Blatt
    - if x == p.key
      gefunden;
      else
      nicht gefunden;
  - p ist innerer Knoten
    - if x <= p.key
      Suchen(p_l, x);
      else
      Suchen(p_r, x);



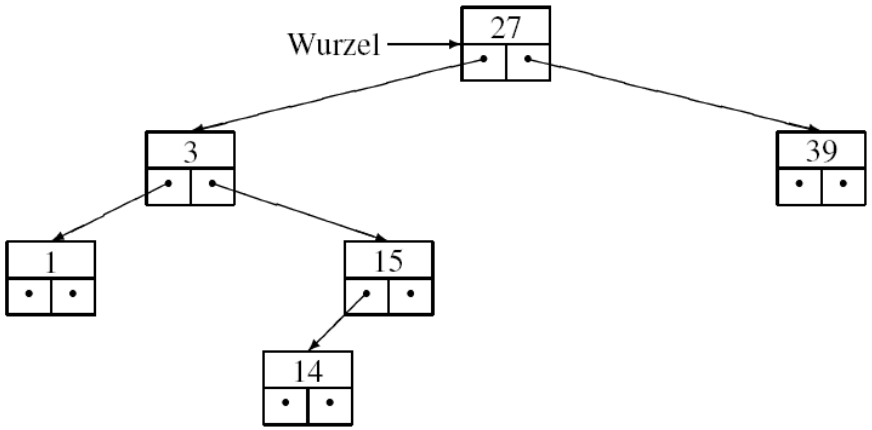
Suchbäume

- Innere Knoten enthalten Schlüssel
  - Alle Schlüssel im linken Teilbaum sind kleiner
  - Alle Schlüssel im rechten Teilbaum sind größer
- Suchen(p, x)
  - p ist Blatt: nicht gefunden
  - p ist innerer Knoten
    - if x < p.key
      Suchen(p_l, x);
      else if x > p.key
      Suchen(p_r, x);
      else
      gefunden;
  - Komplexität?



Suchbäume

- Elemente eines Binärbaums
  - ```
    class Entry<K, E>
    {
        K key;                  // key
        E element;              // data
        Entry<K, E> left;       // Reference to the left subtree
        Entry<K, E> right;      // Reference to the right subtree
    …}
    ```
- Blattknoten können als null Referenz repräsentiert werden
- Der Baum besteht aus einer Referenz auf die Wurzel
  - ```
    public class Tree<K, E>
    {
        private Entry<K, E> root;
        private int size;
        …
    }
    ```

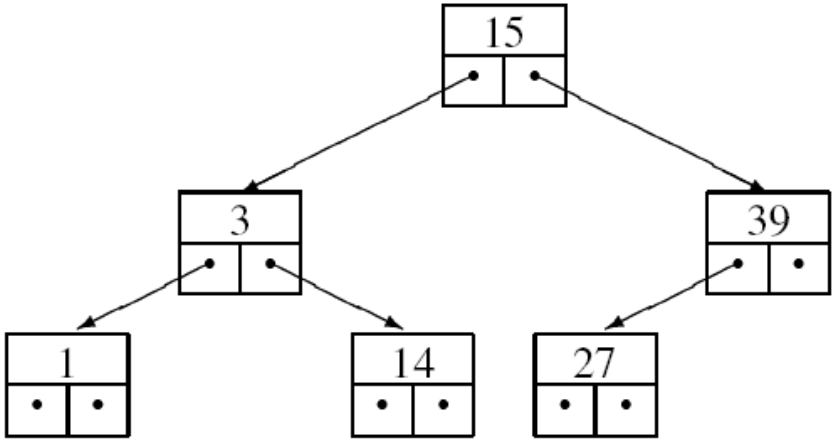


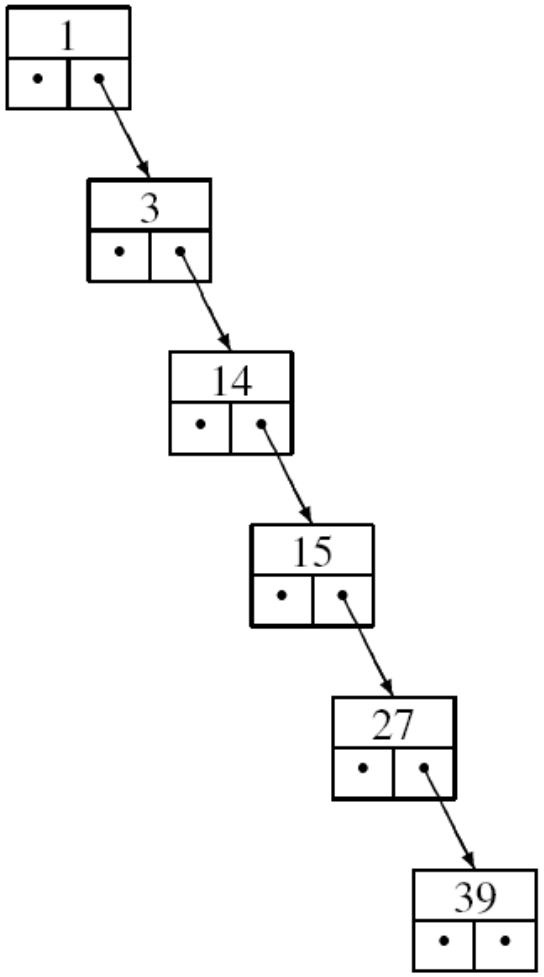
Suchbäume

- Einfügen(key, element)
  - Starte beim Wurzelknoten
  - Falls Wurzelknoten null, neuer Knoten ist Wurzelknoten, sonst
  - if key == p.key {
    Schlüssel bereits im Baum; //fertig
    } else if key < p.key {
    if p.left == null
    p.left = neuer Knoten; //fertig
    else
    p = p.left;
    } else {
    if p.right == null
    p.right = neuer Knoten; //fertig
    else
    p = p.right;
    } solange bis fertig

Suchbäume

- Ergebnis des Einfügens hängt von der Reihenfolge ab
  - 15, 39, 3, 27, 1, 14
  - 1, 3, 14, 15, 27, 39
  - „Natürliche Bäume“



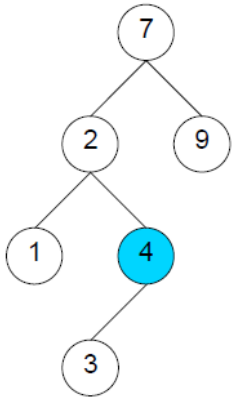


Suchbäume

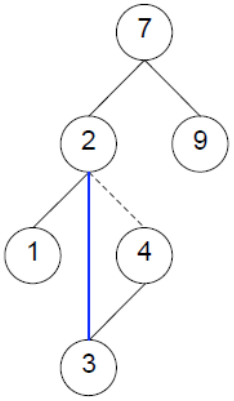
- Entfernen(key)
  - p = Suchen(root, key)
  - Falls p == null
    - Schlüssel nicht im Baum
  - Falls p keinen inneren Knoten als Nachfolger hat
    - p löschen
  - Falls p einen inneren Knoten als Nachfolger hat
    - p durch einzigen Nachfolger ersetzen
  - Falls p zwei innere Knoten als Nachfolger hat
    - Suche im rechten Teilbaum Knoten q mit kleinstem Schlüssel
      - Steht ‚am weitesten links' und hat keinen linken Nachfolger
      - Symmetrischer Nachfolger
    - Ersetze p.key mit q.key
    - Entfernen(q)

Beispiele

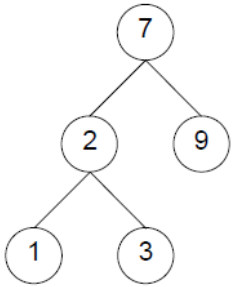
- Ein Nachfolger
  - Lösche 4



Suche 4

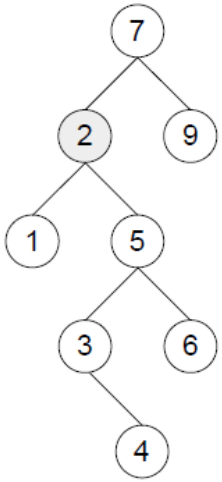


Knoten 4 wird überbrückt und gelöscht

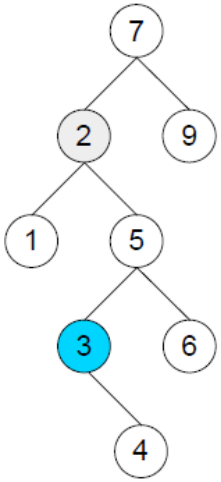


Knoten 4 wurde gelöscht

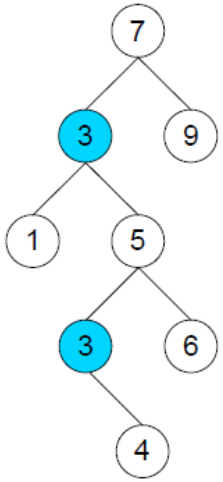
- Zwei Nachfolger
  - Lösche 2



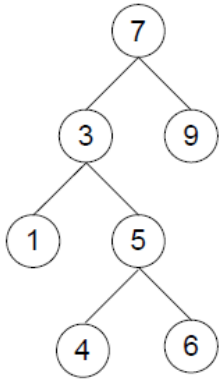
Suche 2



Suche kleinsten Knoten in rechten Teilbaum von 2



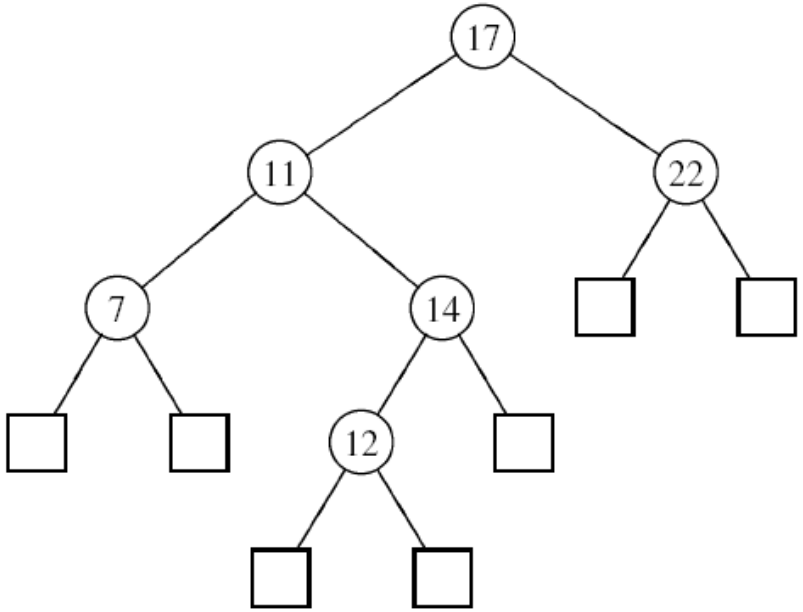
Knoten 2 wurde durch kleinsten Knoten ersetzt



kleinster Knoten wurde gelöscht

Suchbäume

- Durchlaufen
  - Hauptreihenfolge (Preorder)
    - Besuche Wurzel
    - Durchlaufe linken Teilbaum
    - Durchlaufe rechten Teilbaum
  - Nebenreihenfolge (Postorder)
    - Durchlaufe linken Teilbaum
    - Durchlaufe rechten Teilbaum
    - Besuche Wurzel
  - Symmetrische Reihenfolge (Inorder)
    - Durchlaufe linken Teilbaum
    - Besuche Wurzel
    - Durchlaufe rechten Teilbaum



Suchbäume

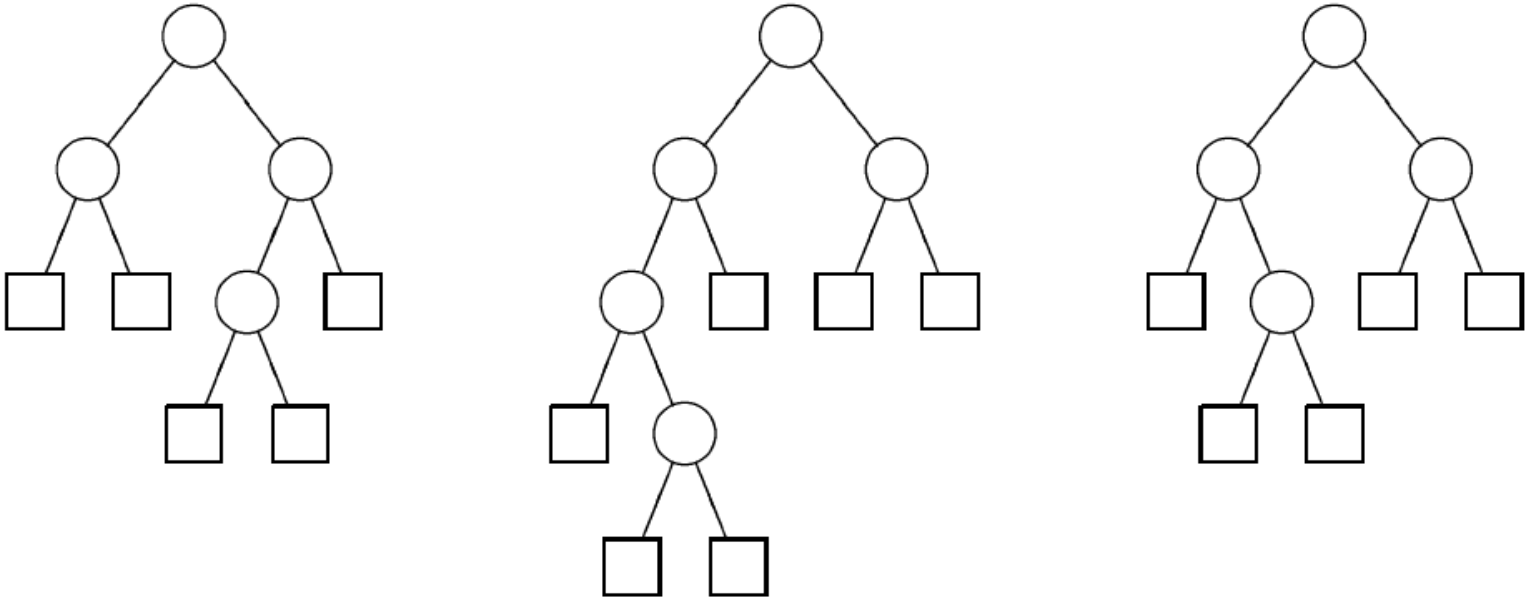
- Ein Suchbaum mit n Schlüsseln
  - Innere Knoten:
  - Blattknoten:
  - Minimale Höhe:
  - Maximale Höhe:
- Problem
  - Im schlechtesten Fall sind alle wichtigen Operationen von der Größenordnung O(n)
- Lösung
  - Vermeidung des schlechtesten Falls

Balancierte Binärbäume

- Statt Bäume natürlich wachsen zu lassen sorgt man dafür, dass ausgeglichene Bäume entstehen
  - Durch zusätzliche Bedingungen für die Knoten
  - Einfügen und Entfernen wird dadurch schwieriger
  - Höhe des Baums liegt dann in O(log n)
  - Suchen, Einfügen und Entfernen in O(log n)

AVL Bäume

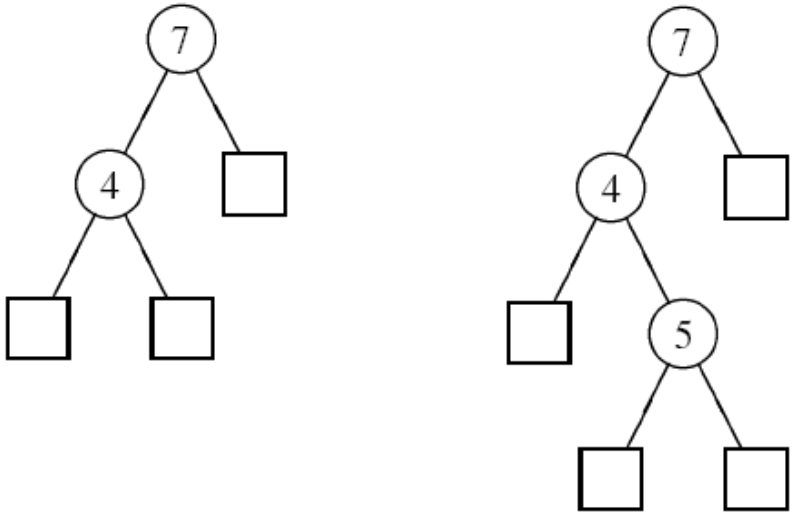
- Von Adelson-Velskij und Landis 1962 eingeführt
- Binärbaum ist ein AVL Baum, wenn für jeden Knoten gilt
  - Höhe des linken und rechten Teilbaums unterscheidet sich um maximal 1
  - „AVL ausgeglichen“ oder „höhenbalanciert“



(a) (b) (c)

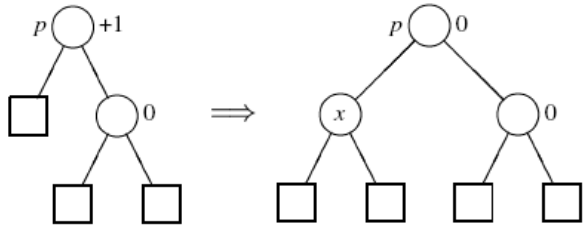
AVL Bäume

- Suchen(element)
  - Identisch zu natürlichen Bäumen
  - Da Höhe des AVL Baums in O(log n) ist Suchen auch in O(log n)
- Einfügen(element), Entfernen(element)
  - Wie bei natürlichen Bäumen
  - Anschließend eventuell Rebalancierung notwendig
  - O(log n)

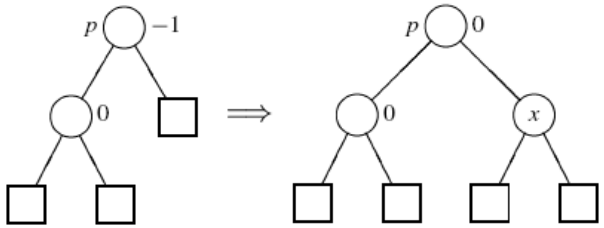


AVL Bäume

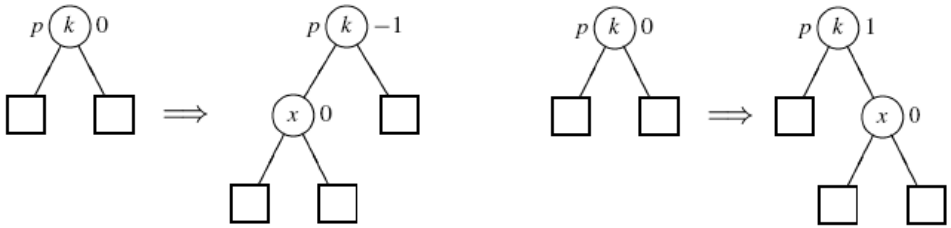
- Knoten müssen um Balancefaktor bal(p) erweitert werden
  - bal(p) = Höhe rechter Teilbaum von p – Höhe linker Teilbaum
  - bal(p) in {-1, 0, +1}
- Einfügen: 3 Fälle (p ist Vater des Blatts, bei dem die Suche endet)
  - bal(p) = +1
    - Links einfügen
    - Danach bal(p) = 0



  - bal(p) = -1
    - Rechts einfügen
    - Danach bal(p) = 0

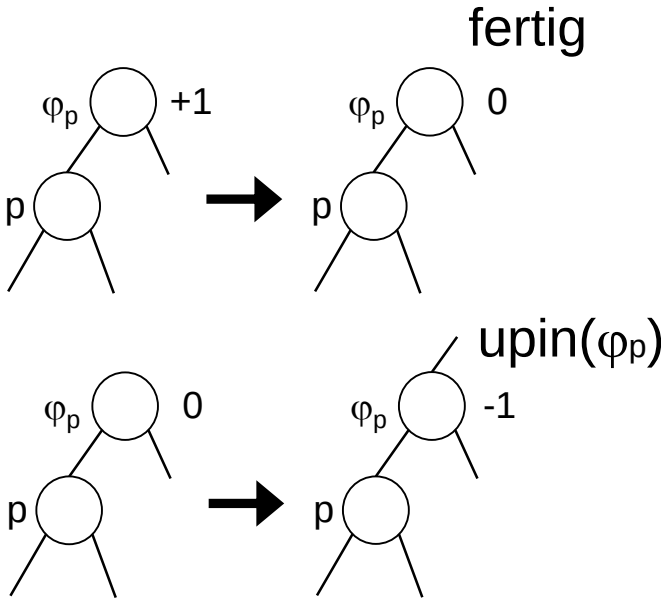


  - bal(p) = 0
    - Höhe ändert sich
    - Balance ändert sich



AVL Bäume

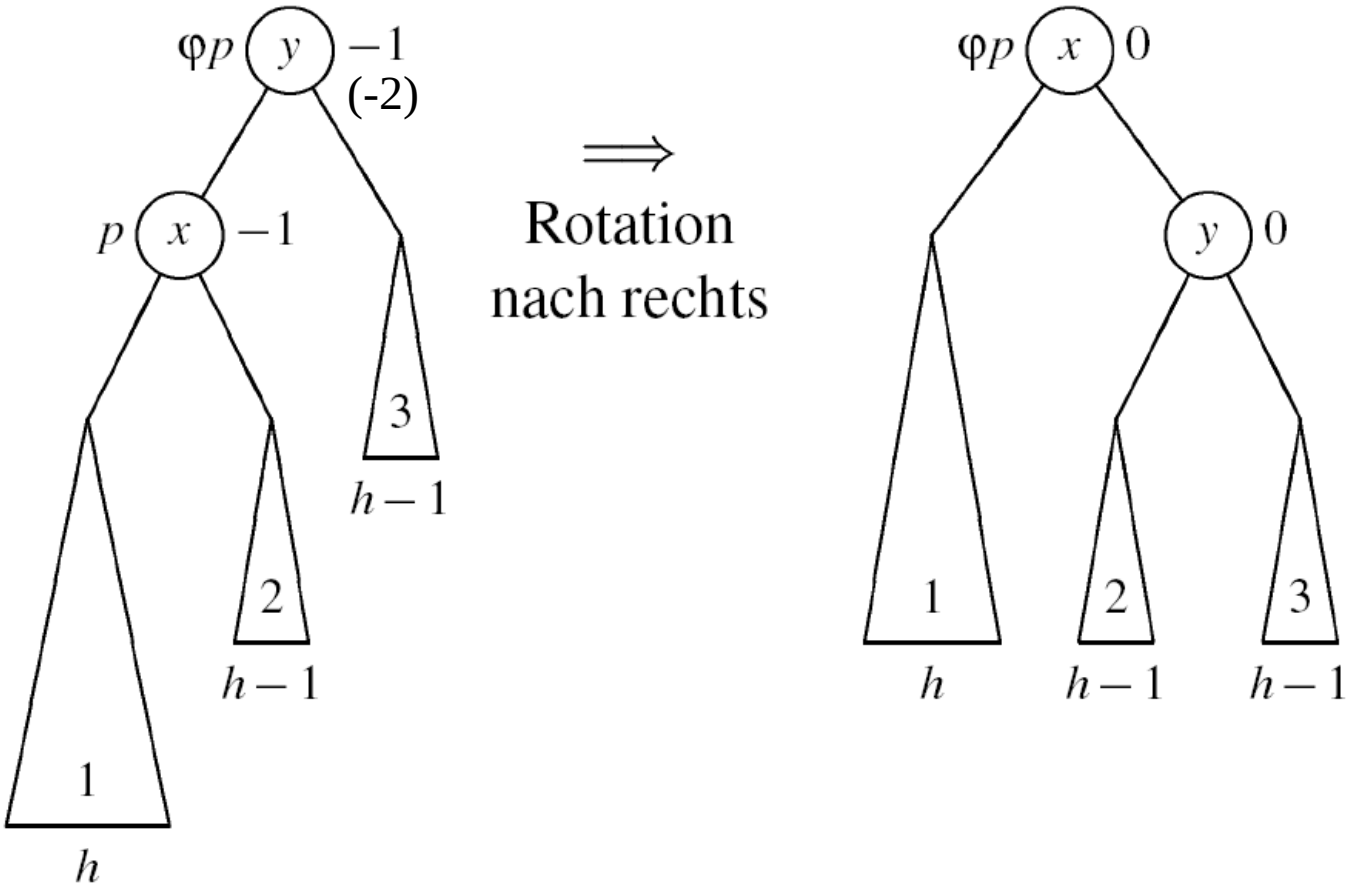
- Höhenänderung propagieren: upin(p) (φp ist Vater von p)
  - p ist linker Sohn von φp
    - bal(φp) = +1
      - danach bal(φp) = 0
      - fertig
    - bal(φp) = 0
      - danach bal(φp) = -1
      - Höhenänderung weiter an Vater von φp propagieren: upin(φp)
    - bal(φp) = -1
      - Rotation notwendig
      - fertig
  - p ist rechter Sohn
    - analog



Nächste Folie

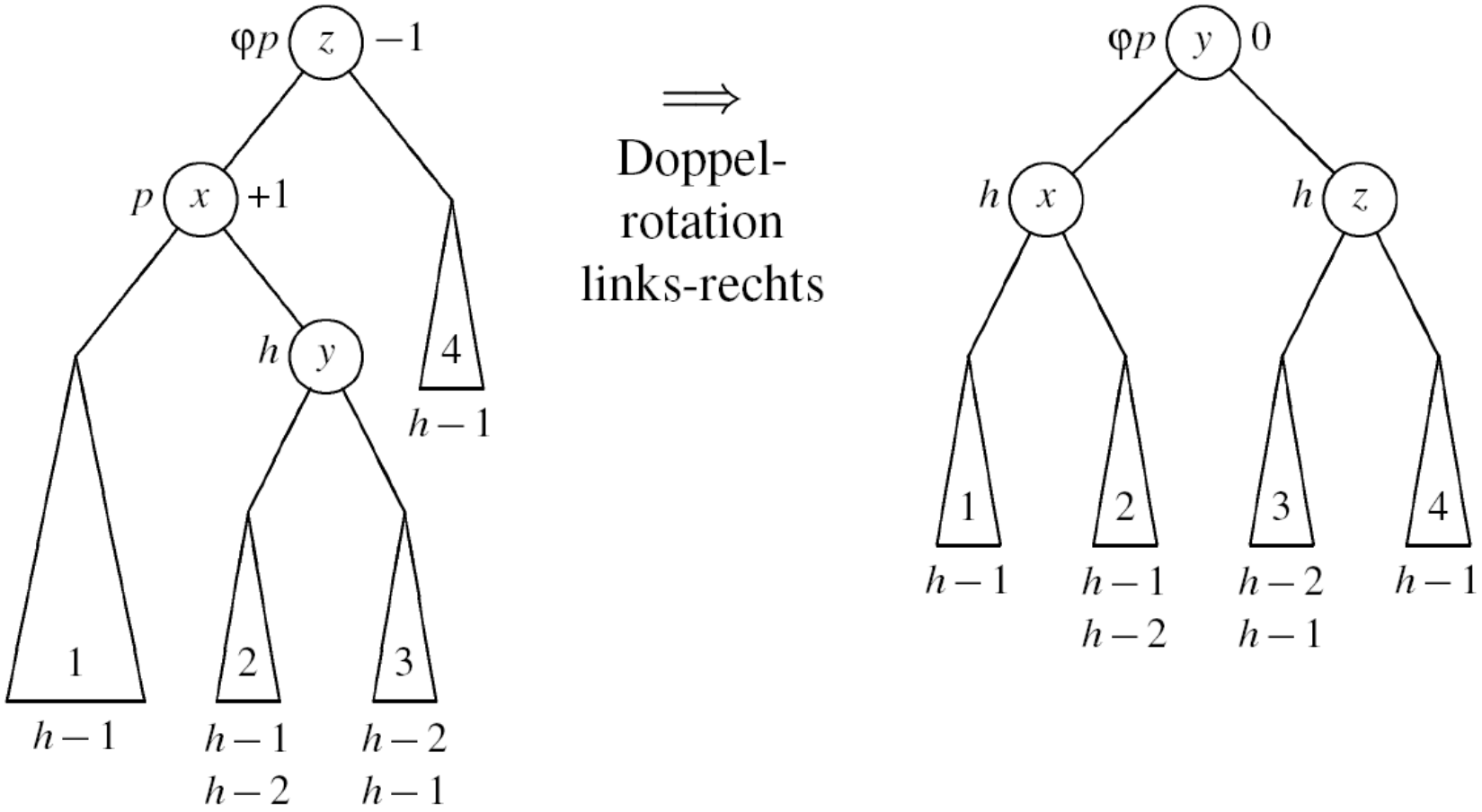
AVL Bäume

- Einfache Rotation
  - bal(φp) = -1
  - 1. Fall: bal(p) = -1



AVL Bäume

- Doppelrotation
  - bal(φp) = -1
  - 2. Fall: bal(p) = +1



AVL Bäume

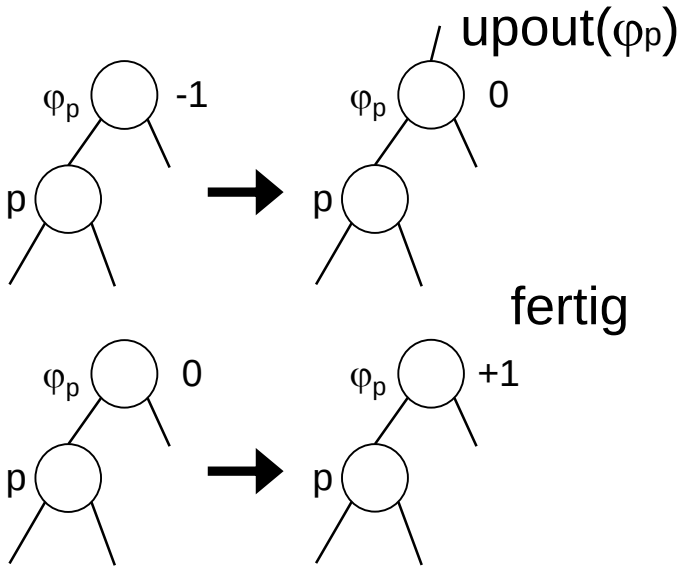
- Einfügen — Komplexität
  - Suche des Vaters der Einfügeposition :
  - Einfügen :
  - Höhenänderung bis maximal Wurzel propagieren :
  - Maximal eine Rotation oder Doppelrotation durchführen :
- Summe :

AVL Bäume

- Entfernen(q)
  - Beide Kinder sind Blätter
    - Lösche Knoten (p sei der Vater des gelöschten Knotens q, falls der Baum nicht leer wird)
    - Anderer Teilbaum hat Höhe 0, 1 oder 2
      - 0: bal(p) = 0 (vorher +1 oder -1)
        Höhe hat sich verkleinert: Höhenänderung nach oben propagieren (upout(p) aufrufen)
      - 1: bal(p) = +1 oder -1 (vorher 0), fertig
      - 2: Baum ist nicht mehr ausgeglichen, upout(q) aufrufen, das zuerst eine Rotation oder Doppelrotation bei p durchführt
  - Ein Kind ist Blatt
    - Ersetze Schlüssel von q mit Schlüssel von einzigem Kindknoten
    - Lösche Kindknoten, bal(q) = 0
    - Propagiere Höhenänderung (von -1) nach oben mit upout(q)
  - Beide Kinder sind innere Knoten
    - Wie bei natürlichen Bäumen
      - Schlüssel(q) = Schlüssel symmetrischer Nachfolger
      - Symmetrischer Nachfolger löschen

AVL Bäume

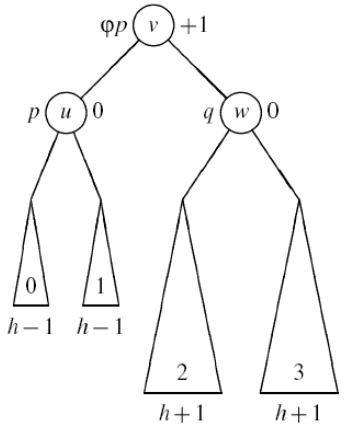
- Höhenänderung propagieren: upout(p) (φp ist Vater von p)
  - bal(p) = 0, Höhe des Teilbaums p wurde um eins kleiner
  - p ist linker Sohn von φp
    - bal(φp) = -1
      - danach bal(φp) = 0
      - upout(φp)
    - bal(φp) = 0
      - danach bal(φp) = +1
      - fertig
    - bal(φp) = +1
      - Rotation notwendig
  - p ist rechter Sohn
    - analog

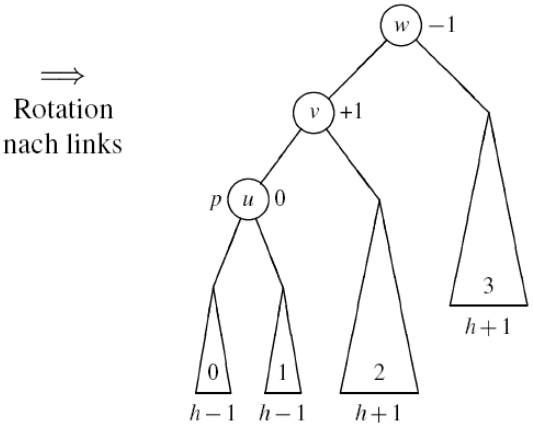


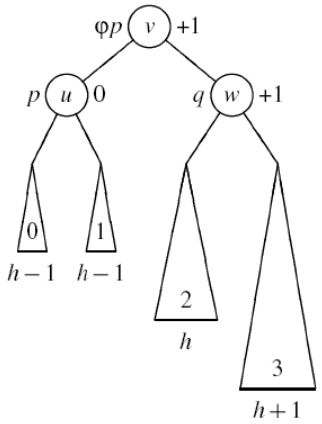
Nächste Folie

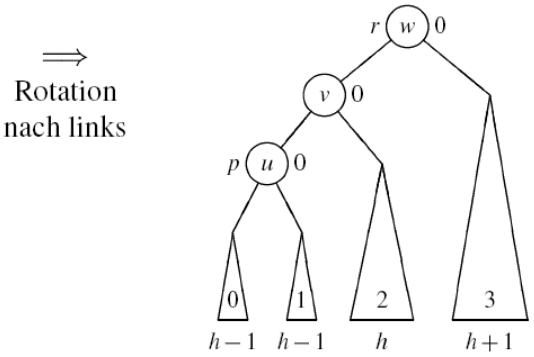
AVL Bäume

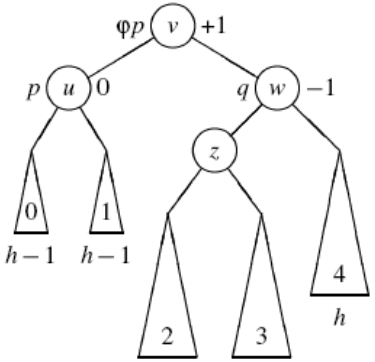
- Rotation beim Entfernen
  - bal(q) = 0
    - Rotation nach links
    - fertig
  - bal(q) = 1
    - Rotation nach links
    - upout(r)
  - bal(q) = -1
    - Doppelrotation rechts-links
    - upout(r)

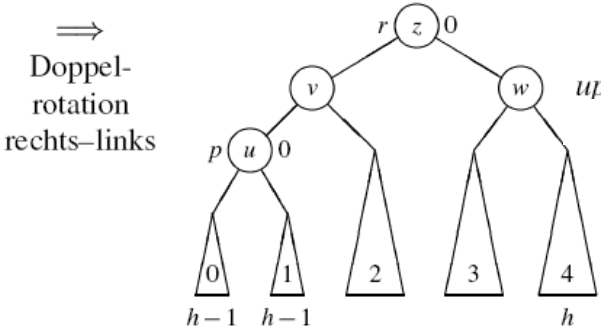










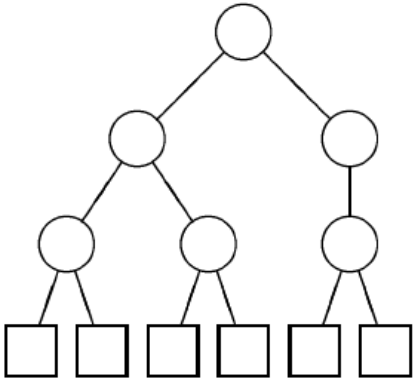


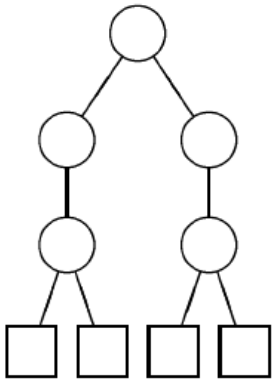
AVL Bäume

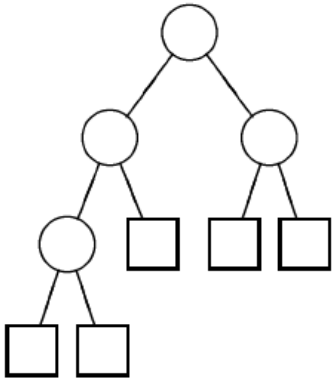
- Entfernen — Komplexität
  - Suche des zu löschenden Knotens :
  - Löschen :
  - Höhenänderung bis maximal Wurzel propagieren :
  - Maximal an jedem Knoten im Pfad eine Rotation oder Doppelrotation durchführen :
- Summe :

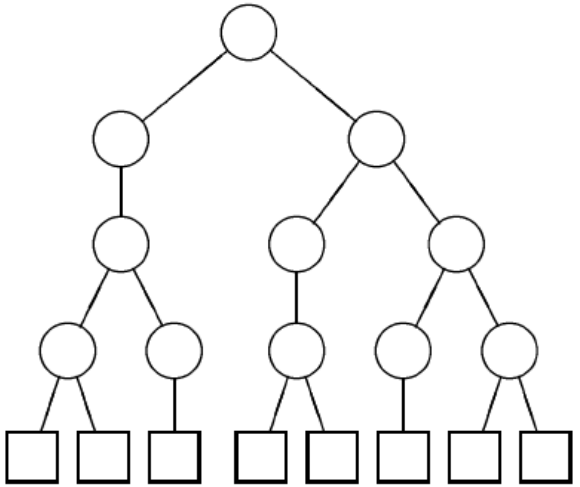
Bruder-Bäume

- Jeder innere Knoten hat einen oder 2 Kinder
- Jeder unäre Knoten hat einen binären Bruderknoten
- Alle Blätter haben dieselbe Tiefe
- (Gegen)Beispiele



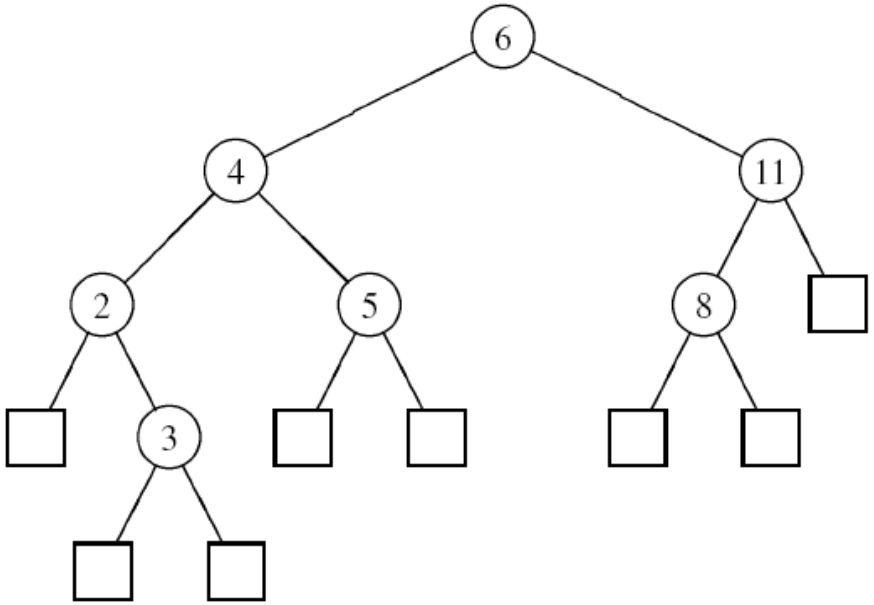






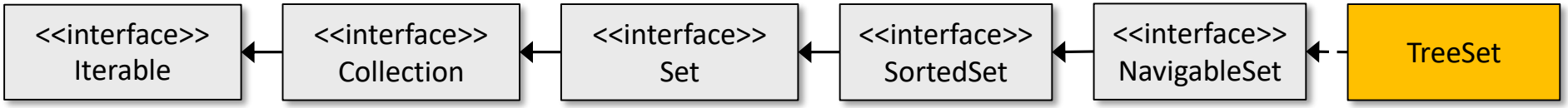
Gewichtsbalancierte Bäume

- Anzahl der Knoten bzw. Blätter im linken und rechten Teilbaum dürfen sich nicht zu stark unterscheiden
  - $W(T_p)$: Anzahl Blätter im Teilbaum T mit Wurzel p
  - $p(T) = W(T_l)/W(T)$: Wurzelbalance
- Gewichtsbalancierter Baum mit Balance α (BB[α]-Baum)
  - Für jeden Teilbaum gilt: $\alpha <= p(T) <= (1- \alpha)$
    - Mit $0<= \alpha <= 1/2$



Bäume in Java

- Bäume in Java sind red black trees
  - eine andere Art balancierter Binärbäume
- Implementiert durch die Klassen TreeSet und TreeMap




```
Set<String> set = new TreeSet<>();

// add sorted in O(log n)
set.add("Neuer");
set.add("Hummels");
set.add("Boateng");

System.out.println("Set: " + set);
```

Zusammenfassung

- Balancierte Binärbäume sind Datenstrukturen die die grundlegenden Operationen in O(log(n)) Zeit erledigen
  - Einfügen
  - Entfernen
  - Suchen
- AVL Bäume gewährleisten dies durch Höhenbalance