Algorithmen und Datenstrukturen
Suchbaum

Matthias Teschner
Graphische Datenverarbeitung
Institut für Informatik
Universität Freiburg

SS 09

Motivation

- Datenstruktur zur Repräsentation dynamischer Mengen
- unterstützt `insert, search, delete, minimum, maximum, predecessor, successor`
- Grundoperationen im mittleren Fall in O (log n), was der Baumhöhe entspricht. Schlechtester Fall O (n)
- Wörterbuchoperationen langsamer als bei Hashverfahren, dafür Unterstützung von mehr Operationen, z. B. sortierte Ausgabe

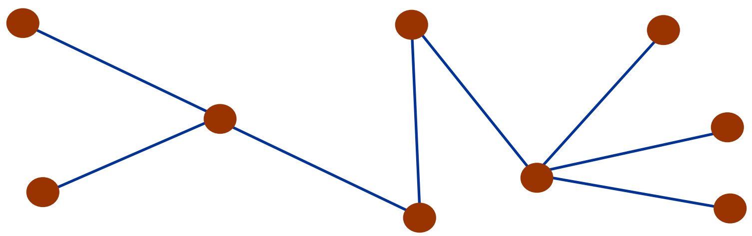
Überblick

- Baum
- Binärer Suchbaum

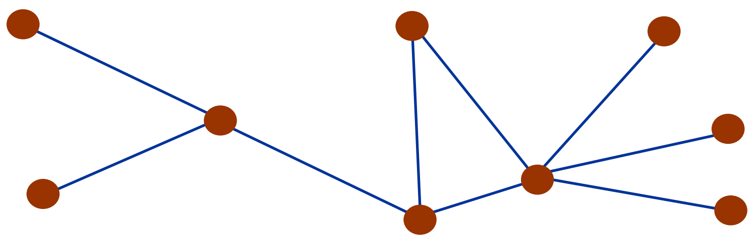
Baum

- verallgemeinerte Listen
  - Elemente können mehr als nur einen Nachfolger haben
- spezieller Graph
  - Graph G besteht aus Knoten V,
    die durch Kanten E verbunden sind G = (V, E)
  - Kanten sind gerichtet oder ungerichtet
  - Zahl der Knoten: |V| = n, Zahl der Kanten: |E| = m
  - G ist ein Baum
    - gdw. zwischen je zwei Knoten genau ein Weg existiert
    - gdw. G zusammenhängend ist und m=n-1
    - gdw. G keinen Zyklus enthält und m=n-1

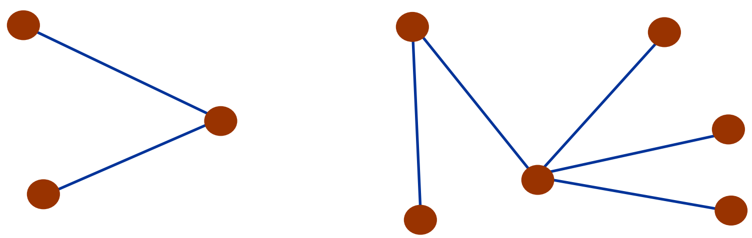
Beispiele



Baum



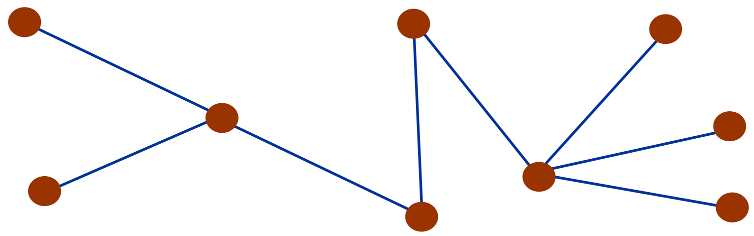
kein Baum



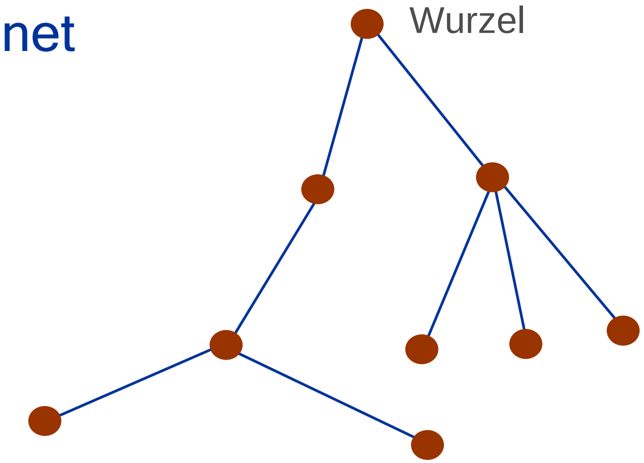
kein Baum

Baum

- ungerichteter Baum
  - kein ausgezeichneter Knoten



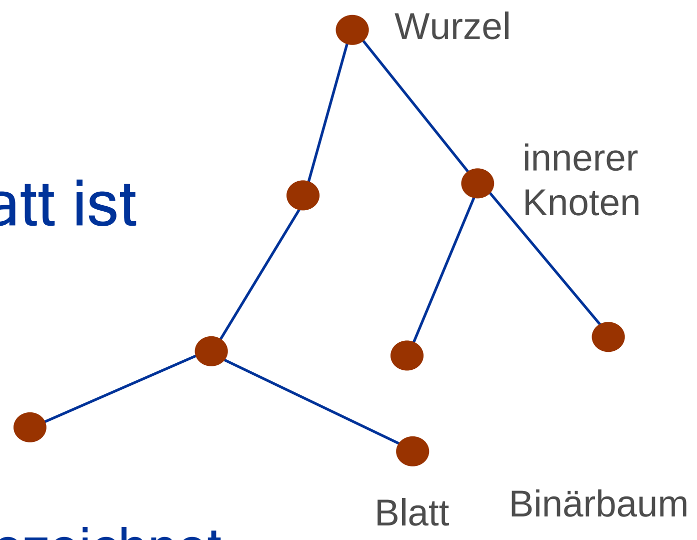
- gewurzelter Baum
  - ein Knoten ist als Wurzel ausgezeichnet
  - Vater eines Knotens:
    sein direkter Vorgänger
    auf dem Weg zur Wurzel
  - Sohn eines Knotens:
    sein direkter Nachfolger
  - Rang eines Knotens:
    die Zahl seiner Kinder



Gewurzelter Baum

- im Folgenden nur als "Baum" bezeichnet

- Wurzel: einziger Knoten ohne Vater
- Blatt: Knoten ohne Söhne
- innerer Knoten: Knoten, der kein Blatt ist
- Ordnung des Baums:
  maximaler Rang aller Knoten
- Binärbaum: Baum der Ordnung 2
  - Söhne werden durch links und rechts bezeichnet
- Vielwegbaum: Baum höherer Ordnung

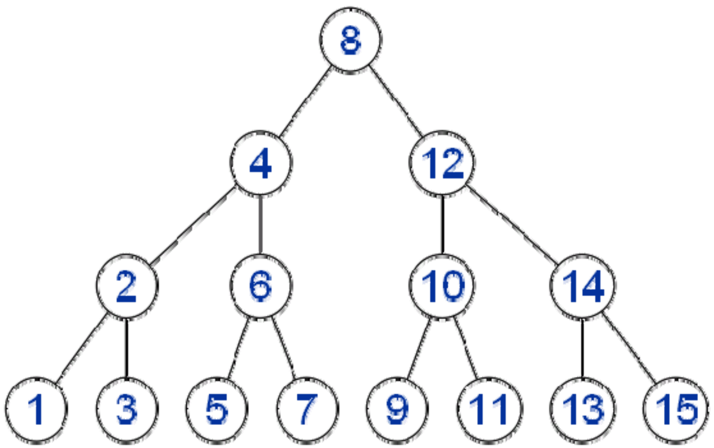


Gewurzelter Baum

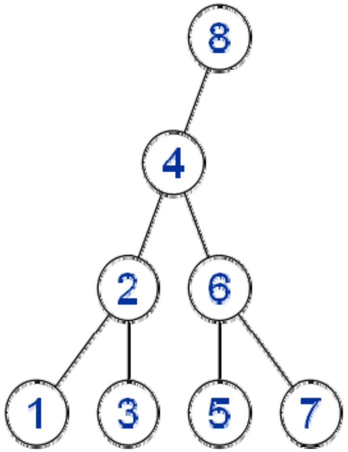
- Tiefe eines Knotens:
  Zahl der Kanten vom Knoten zur Wurzel
- Höhe eines Baums:
  Maximale Tiefe eines Blattes des Baums
- Niveau / Ebene:
  Alle Knoten einer festen Tiefe
- Vollständiger Baum:
  Jedes nichtleere Niveau hat die maximale Knotenzahl.
  Alle Blätter haben die gleiche Tiefe

Binärer Suchbaum

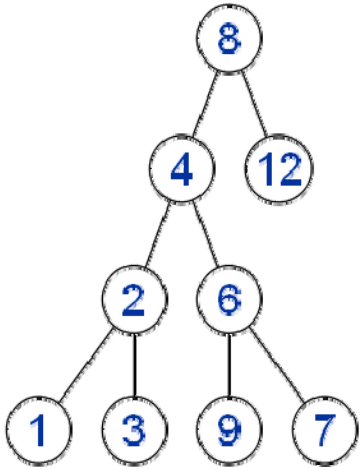
- Binärbaum mit Suchbaum-Eigenschaft:
  - Knoten x mit schlüssel [x]
  - Für alle Knoten y im linken Teilbaum gilt: schlüssel [y] $\leq$ schlüssel [x].
  - Für alle Knoten y im rechten Teilbaum gilt: schlüssel[x] $\leq$ schlüssel[y].



binärer Suchbaum



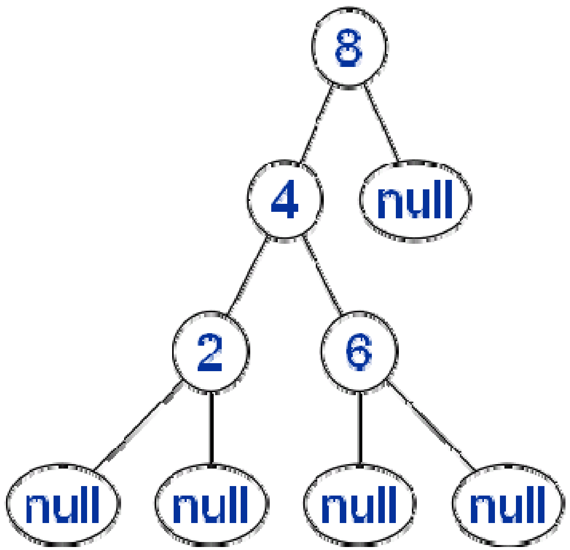
binärer Suchbaum



kein binärer
Suchbaum

Binärer Suchbaum

- Suchbaum:
  (Natürlicher Baum)
  Schlüssel werden in
  inneren Knoten gespeichert.



- Blattsuchbaum:
  Schlüssel werden in Blättern gespeichert.
  Innere Knoten enthalten Wegweiser zum Auffinden
  von Schlüsseln in Blättern.

Sortierte Ausgabe

- sortierte Ausgabe der Elemente eines binären Suchbaums
  - nutze die Suchbaum-Eigenschaft aus
  - traversiere rekursiv: linker Teilbaum, Wurzel, rechter Teilbaum
  - Laufzeit O(n)

```
void inOrderTreeWalk (Node x) {
   if (x!=null) {
      inOrderTreeWalk (x.left);
      print x.key;
      inOrderTreeWalk (x.right);
   }
}
```

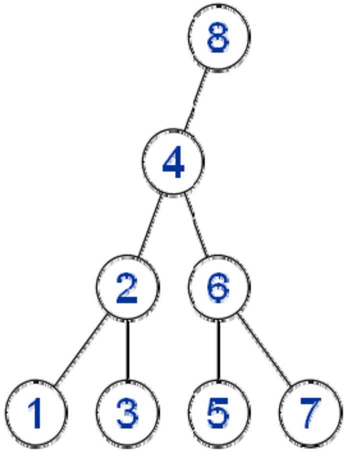
in-order:	links, Wurzel, rechts
pre-order:	Wurzel, links, rechts
post-oder:	links, rechts, Wurzel

x.left	– Zeiger auf linken Sohn
x.right	– Zeiger auf rechten Sohn
x.key	– Schlüssel
x.parent	– Zeiger auf Vater

Sortierte Ausgabe

- alternative Implementierung

- ```
  void inOrderTreeWalk (Node x) {
     if (x.left!=null) inOrderTreeWalk(x.left);
     print x.key;
     if (x.right!=null) inOrderTreeWalk(x.right);
  }
  ```



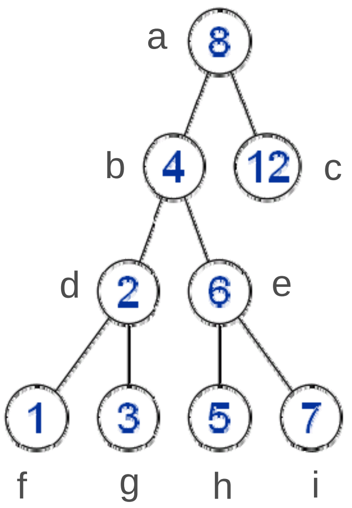
Ausgabe:
1, 2, 3, 4, 5, 6, 7, 8
  ```

Laufzeit

- $T(0) = c$
- $T(n) = T(k) + T(n-k-1) + d$

- Vermutung: $T(n) = (c + d)\, n + c$
- Beweis durch vollständige Induktion:
  $T(0) = c$
  $$\begin{aligned} T(n) &= T(k) + T(n-k-1) + d \\ &= (c + d)\, k + c + (c + d)\,(n - k - 1) + c + d \\ &= (c + d)\, n + c + (c + d)\,(-1) + c + d \\ &= (c + d)\, n + c \end{aligned}$$

Suche

- vergleiche Suchschlüssel mit Schlüssel eines Knotens und entscheide, ob links oder rechts weitergesucht wird
- rekursiv bis Schlüssel gefunden oder Blatt erreicht
- Laufzeit O(log n)

x sollte die Wurzel des Baums sein.

- ```
  Node search (Node x, int key) {
      if (x==null || key == x.key) return x;
      if (key < x.key)
          return search(x.left, key);
      else
          return search (x.right, key);
  }
  ```
  ```

Suche

- iterative Implementierung

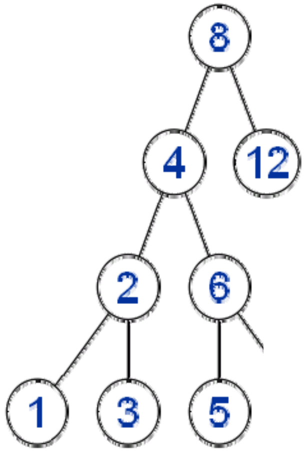
- ```
  Node search (Node x, int key) {
      while (x!=null && key != x.key) {
          if (key < x.key)
              x = x.left;
          else
              x = x.right;
      }
    return x;
  }
  ```

search (a, 6)



6<8 → x = x.left = b

6>4 → x = x.right = e

6==6 → return x=e
  ```

Minimum / Maximum

- traversiere ausgehend vom Knoten den links-Zeiger / rechts-Zeiger bis kein linkes / rechtes Kind mehr existiert
- Laufzeit O (log n)

x sollte die Wurzel des Baums sein.

```
Node minimum (Node x) {
    while (x.left!=null) x = x.left;
    return x;
}
```

```
Node maximum (Node x) {
    while (x.right!=null) x = x.right;
    return x;
}
```

Nachfolger / Vorgänger

- Laufzeit O (log n)

beliebiger Knoten x

- ```
  Node successor (Node x) {
      if (x.right!=null)
          return minimum(x.right);
      Node y = x.parent;
      while (y!=null && x==y.rechts) {
          x=y; y=y.parent;
      }
      return y;
  }
  ```

dichtestes Vorfahre von x, dessen linker Sohn ebenfalls Vorfahre von x ist.

(Für 6 ist das die 8, weil 8 der dichteste Vorfahre von 6 ist, für den der linke Sohn, die 4, ebenfalls Vorfahre von 6 ist.


  ```

Einfügen

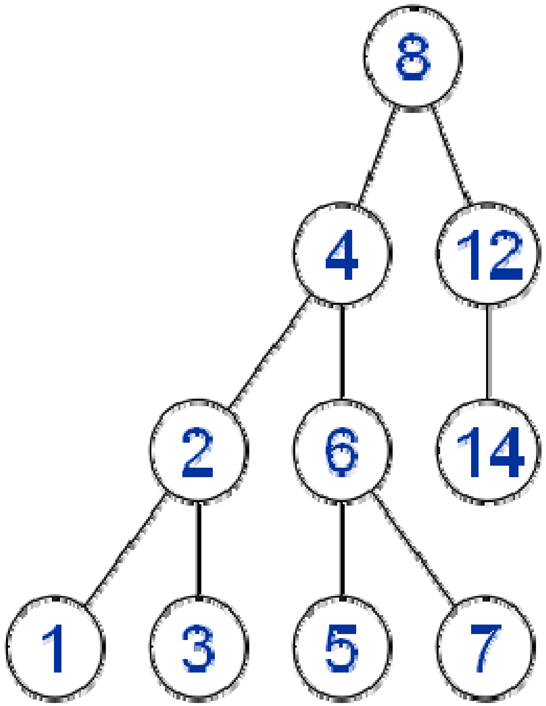
- Laufzeit in O (log n)

Wurzel einzufügender Knoten

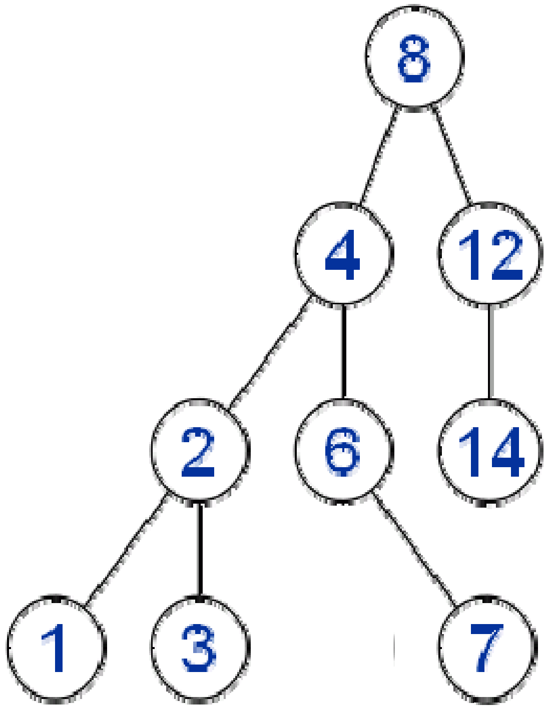
```
void insert (Node root, Node ins) {
    if (root==null) { root = ins; return; }
    Node parent, x=root;
    while (x!=null) {
        parent = x;
        if (ins.key < x.key)
            x=x.left;
        else
            x=x.right;
    }
    if (ins.key < parent.key)
        parent.left = ins;
    else
        parent.right = ins;
    return;
}
```

Löschen

- Fall 1: zu löschender Knoten hat keine Kinder

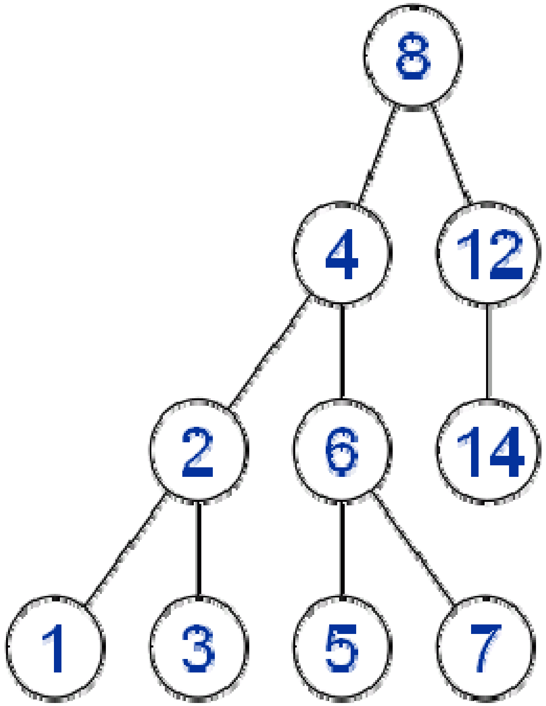


Lösche 5:
ermittle Vater (6)
ermittle, ob 5 linker
oder rechter Sohn ist
setze 6.left = null;

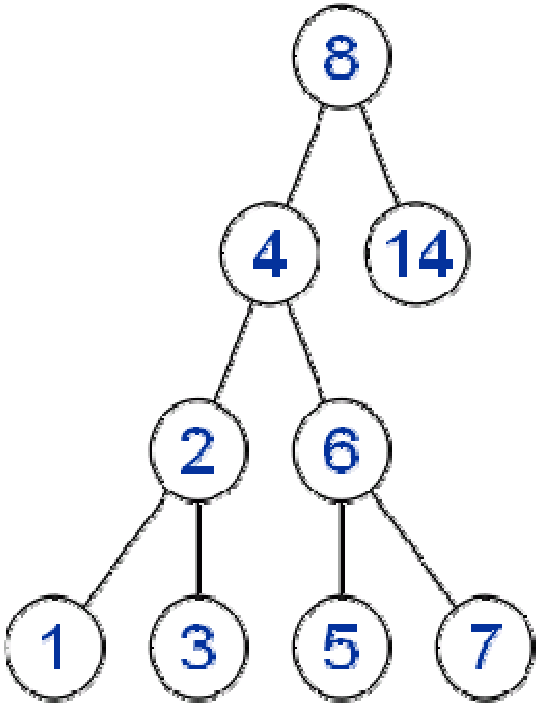


Löschen

- Fall 2: zu löschender Knoten hat ein Kind

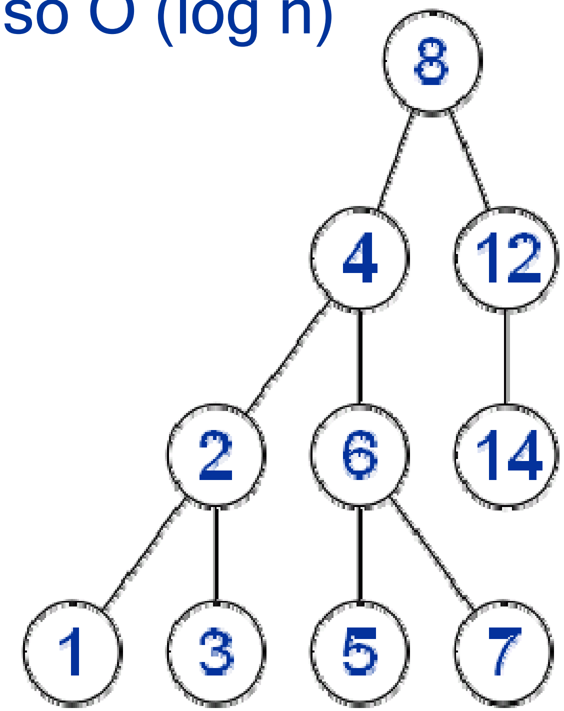


Lösche 12:
ermittle Sohn (14)
ermittle Vater (8)
ermittle, ob 12 linker oder rechter Sohn ist
setze 8.right = 14;

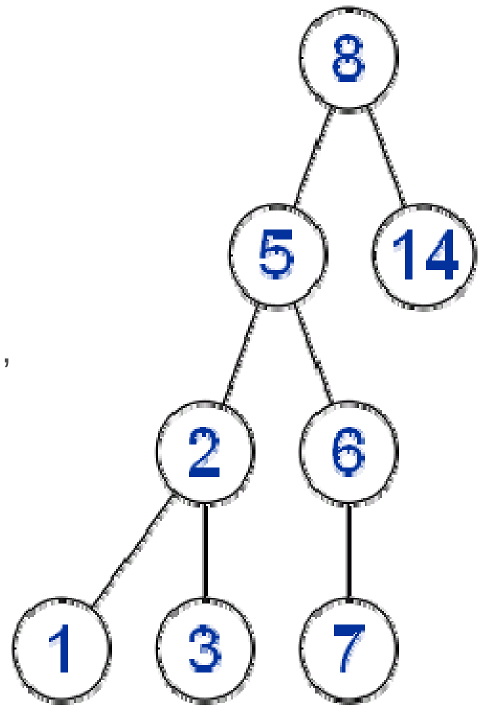


Löschen

- Fall 3: zu löschender Knoten hat zwei Kinder
- Laufzeit ergibt sich aus Laufzeit für `successor`, also O (log n)



Lösche 4:
ermittle successor(4) = 5
ersetze 4 durch 5
lösche 5 (Fall 1 oder Fall 2, da 5 als Nachfolger von 4 keinen linken Sohn haben kann.



Zusammenfassung

- Suchbaum-Eigenschaft
  - Für alle Knoten y im linken Teilbaum von x gilt:
    schlüssel [y] $\leq$ schlüssel [x].
  - Für alle Knoten y im rechten Teilbaum von x gilt:
    schlüssel[x] $\leq$ schlüssel[y].
- Dann können `insert`, `search`, `delete`, `minimum`, `maximum`, `predecessor`, `successor` in O(log n) realisiert werden, sortierte Ausgabe in O(n).

Nächstes Thema

- Algorithmen / Datenstrukturen
  - noch mehr Bäume