

# Informatik I: Einführung in die Programmierung

## 10. Bäume

Albert-Ludwigs-Universität Freiburg



Prof. Dr. Peter Thiemann

26. November 2024

# 1 Der Baum



- Definition
- Terminologie
- Beispiele

## Der Baum

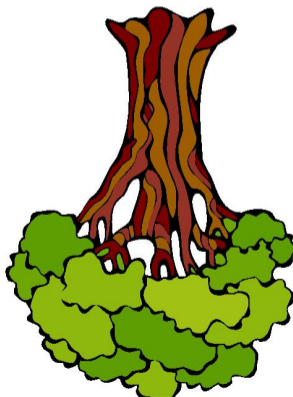
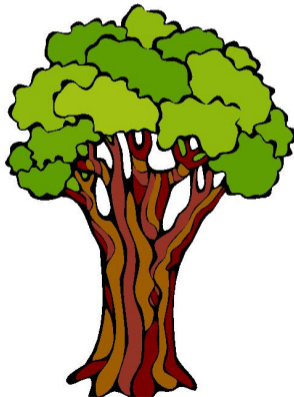
Definition  
Terminologie  
Beispiele

Binärbäume

Suchbäume

Zusammen-  
fassung

- Bäume sind in der Informatik allgegenwärtig.
- Gezeichnet werden sie meistens mit der Wurzel nach oben!



## Der Baum

Definition  
Terminologie  
Beispiele

## Binärbäume

## Suchbäume

## Zusammenfassung

# 1 Der Baum



- Definition
- Terminologie
- Beispiele

Der Baum

**Definition**

Terminologie

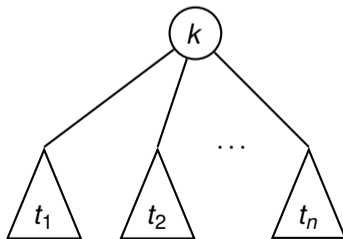
Beispiele

Binärbäume

Suchbäume

Zusammenfassung

- Der **leere Baum** ist ein Baum.
- Wenn  $t_1, \dots, t_n$ ,  $n \geq 0$  disjunkte Bäume sind und  $k$  ein **Knoten**, der nicht in  $t_1, \dots, t_n$  vorkommt, dann ist auch die Struktur bestehend aus der **Wurzel**  $k$  mit **zugeordneten Teilbäumen**  $t_1, \dots, t_n$  ein **Baum**.
- Nichts sonst ist ein Baum.
- Bildlich:



# 1 Der Baum



- Definition
- Terminologie
- Beispiele

Der Baum

Definition

**Terminologie**

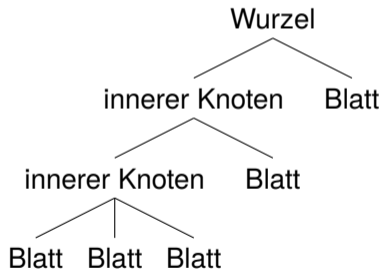
Beispiele

Binärbäume

Suchbäume

Zusammenfassung

- Alle Knoten, denen keine Teilbäume zugeordnet sind, heißen **Blätter**.
- Knoten, die keine Blätter sind, heißen **innere Knoten**.



- Die Wurzel kann also ein Blatt sein (keine weiteren Teilbäume) oder ein innerer Knoten.

Der Baum

Definition

Terminologie

Beispiele

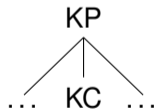
Binärbäume

Suchbäume

Zusammenfassung

## Eltern und Kinder

Wenn KP ein Knoten und KC die Wurzel eines zugeordneten Teilbaums ist, dann gilt:



- KP ist **Elternknoten** von KC (höchstens einer),
- Der Elternknoten von KP, dessen Elternknoten usw. sind **Vorgänger** von KC.
- KC ist **Kind** von KP.
- Kinder von KC, deren Kinder, usw. sind **Nachfolger** von KP.

Der Baum

Definition

Terminologie

Beispiele

Binärbäume

Suchbäume

Zusammenfassung



## Markierte Bäume

- Bäume sind oft **markiert**. Die Markierung weist jedem Knoten eine **Marke** zu.
- Formal: Wenn  $K$  die Knotenmenge eines Baums ist und  $M$  eine Menge von Marken, dann ist die **Markierung eine Abbildung  $\mu : K \rightarrow M$** .

Der Baum

Definition

**Terminologie**

Beispiele

Binärbäume

Suchbäume

Zusammenfassung

# 1 Der Baum



- Definition
- Terminologie
- Beispiele

Der Baum

Definition

Terminologie

**Beispiele**

Binärbäume

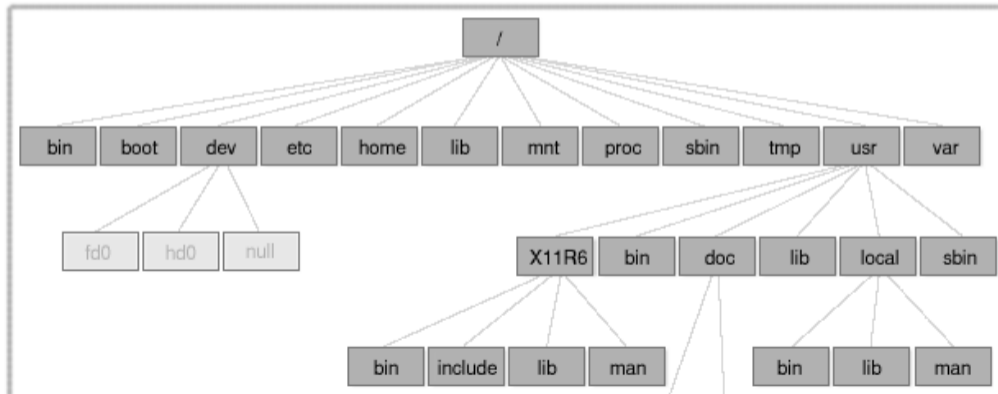
Suchbäume

Zusammenfassung

# Beispiel: Verzeichnisbaum



In vielen Betriebssystemen ist die Verzeichnisstruktur im Wesentlichen baumartig.  
Knotenmarkierung: Dateiname



Der Baum

Definition

Terminologie

Beispiele

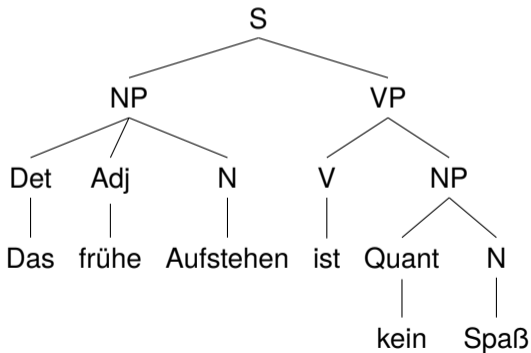
Binärbäume

Suchbäume

Zusammenfassung

# Beispiel: Syntaxbaum

Wenn die Struktur einer Sprache mit Hilfe einer formalen Grammatik spezifiziert ist, dann kann der Satzaufbau durch **Syntaxbäume** beschrieben werden.



Der Baum

Definition

Terminologie

Beispiele

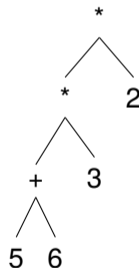
Binärbäume

Suchbäume

Zusammenfassung

# Beispiel: Ausdrucksbaum

- Bäume können Ausdrücke so darstellen, dass ihre Auswertung eindeutig durchführbar ist, ohne dass Klammern notwendig sind.
- Beispiel:  $(5 + 6) * 3 * 2$
- Entspricht:  $((5 + 6) * 3) * 2$
- Operatoren als Markierung innerer Knoten, Zahlen als Markierung der Blätter:



Der Baum

Definition

Terminologie

Beispiele

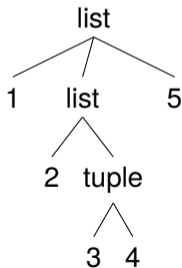
Binärbäume

Suchbäume

Zusammenfassung

# Beispiel: Listen und Tupel als Bäume

- Jede Liste und jedes Tupel kann als Baum angesehen werden, bei dem der Typ die Knotenmarkierung ist und die Elemente die Teilbäume sind.
- Beispiel:  $[1, [2, (3, 4)], 5]$



## 2 Binärbäume



- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Der Baum

**Binärbäume**

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung

- Der Binärbaum ist ein **Spezialfall eines Baumes**.
- Ein Binärbaum ist **entweder** leer **oder** besteht aus einem (Wurzel-) Knoten und zwei Teilbäumen.
- Für viele Anwendungsfälle angemessen.
- Funktionen über solchen Bäumen sind einfach definierbar.

Der Baum

**Binärbäume**

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

**Suchbäume**

**Zusammenfassung**



## 2 Binärbäume



- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Der Baum

Binärbäume

**Repräsentation**

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung

- Der **leere Baum** wird durch `None` repräsentiert.
- Jeder andere **Knoten** wird durch ein `Node`-Objekt repräsentiert.
  - Das Attribut `mark` enthält die **Markierung**.
  - Das Attribut `left` enthält den **linken Teilbaum**.
  - Das Attribut `right` enthält den **rechten Teilbaum**.
- Beispiele:
  - Der Baum bestehend aus dem einzigen Knoten mit der Markierung 8:  
`Node(8, None, None)`
  - Der Baum mit Wurzel '+', linkem Teilbaum mit Blatt 5, rechtem Teilbaum mit Blatt 6:  
`Node('+', Node(5, None, None), Node(6, None, None))`

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

```
from typing import Optional
@dataclass
class Node[T]:
    mark : T
    left : Optional['Node[T]'] = None
    right: Optional['Node[T]'] = None
type BTree[T] = Optional[Node[T]]
```

## Bemerkung zu den Typannotationen

- `Node[T]`: Typ einer *generischen Klasse*  
T ist der Typ der Markierung des Baums
- `Optional[t]`: entweder t oder None (aber nichts anderes)
- Der Typ `Node` existiert erst **nach** Ausführung der `class`-Anweisung. Python ersetzt den String `'Node[T]'` in der Typannotation rückwirkend durch den Typ `Node[T]`.

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

## 2 Binärbäume



- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Der Baum

Binärbäume

Repräsentation

**Beispiel**

Funktionen auf  
Bäumen

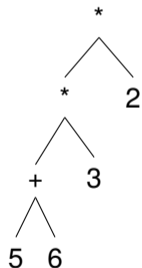
Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung

# Beispiel: Der Ausdrucksbaum



Darstellung mit Node Objekten:

```
Node('*', Node('*', Node('+', Node(5, None, None),  
Node(6, None, None)),  
Node(3, None, None)),  
Node(2, None, None))
```

Der Baum

Binärbäume

Repräsentation

**Beispiel**

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

## 2 Binärbäume



- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Der Baum

Binärbäume

Repräsentation

Beispiel

**Funktionen auf  
Bäumen**

Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung

## Aufgabe

Transformiere einen Baum mit beliebiger Markierung in einen String.

## Signatur

```
def tree_str(tree : BTree[Any]) -> str:
```

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammen-  
fassung

## Präzisierung

- Jeder Knoten des Baums muss in einen String transformiert werden.
  - `BTree[Any] = Optional[Node[Any]]` ist ein Uniontyp (Alternative).
- ⇒ Pattern matching
- Zusätzliches Problem:  
Node-Objekte enthalten selbst Attribute vom Typ `BTree[Any]`.
  - Abhilfe **Wunschdenken**:  
nehme an, dass `tree_str` auf den Teilbäumen schon das Problem löst!
  - D.h. verwende die Funktion in ihrer eigenen Definition (**Rekursion**)!

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung



```
def tree_str(tree : BTree[Any]) -> str:
  match tree:
    case None:
      return "fill in"
    case Node (mark, left, right):
      l_str = tree_str(left)
      r_str = tree_str(right)
      return "fill in"
```

- Node Objekte enthalten selbst wieder Node Objekte (oder None) in den Attributen `left` und `right`.
- Zum Ausdrucken eines Node Objekts müssen auch die Node Objekte in den Attributen ausgedruckt werden.
- `tree_str` ist **rekursiv**, denn es wird in seiner eigenen Definition aufgerufen!

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung

- Die **rekursiven Aufrufe** `tree_str (left)` und `tree_str (right)` erfolgen **nur auf den Kindern des Knotens**.
- Ergibt sich zwangsläufig aus der induktiven Definition!
- **Rekursive Aufrufe auf den Teilbäumen** sind Teil des Funktionsgerüsts, sobald eine baumartige Struktur bearbeitet werden soll.
  
- Die **Alternative** “`case None`” ergibt sich zwangsläufig aus dem Typ `tree:Optional[Node]`: `tree` ist **entweder `None` oder eine `Node`-Instanz**.
  
- Alle Funktionen auf Binärbäumen verwenden dieses Gerüst.

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammen-  
fassung

```
def tree_str(tree : BTree[Any]) -> str:
  match tree:
    case None:
      return "None"
    case Node (mark, left, right):
      return ("Node("
        + repr(m) + ", "
        + tree_str (left) + ", "
        + tree_str (right) + ")")
```

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

## 2 Binärbäume



- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung

# Tiefe von Knoten, Höhe und Größe von (Binär-)Bäumen

induktiv definiert



- Die **Tiefe eines Knotens**  $k$  (Abstand zur Wurzel) ist
  - 0, falls  $k$  die Wurzel ist,
  - $i + 1$ , wenn  $i$  die Tiefe des Elternknotens ist.
- Die **Höhe eines Baumes** ist die maximale Tiefe über alle Blätter:
  - $-1$  für den leeren Baum,
  - $m + 1$ , wenn  $m$  die maximale Höhe aller der Wurzel zugeordneten Teilbäume ist.
- Die **Größe eines Baumes** ist die Anzahl seiner Knoten.
  - 0 für den leeren Baum,
  - $s + 1$ , wenn  $s$  die Summe der Größen der Teilbäume ist.

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

$$\text{height}(tree) = \begin{cases} -1, & \text{if } tree \text{ is empty} \\ 1 + \max(\text{height}(tree.left), \text{height}(tree.right)), & \text{otherwise.} \end{cases}$$

$$\text{size}(tree) = \begin{cases} 0, & \text{if } tree \text{ is empty;} \\ 1 + \text{size}(tree.left) + \text{size}(tree.right), & \text{otherwise.} \end{cases}$$

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

```
def height(tree : BTree[Any]) -> int:
  match tree:
    case None:
      return -1
    case Node (m, l, r):
      return(max(height(l), height(r)) + 1)

def size(tree : BTree[Any]) -> int:
  match tree:
    case None:
      return 0
    case Node (m, l, r):
      return(size(l) + size(r) + 1)
```

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

## 2 Binärbäume



- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaf-  
ten

Traversierung

Suchbäume

Zusammen-  
fassung



- Oft sollen alle Knoten eines Baumes besucht und bearbeitet werden.
- 3 Vorgehensweisen (**Traversierungen**) sind üblich:
  - **Pre-Order** (Hauptreihenfolge): Bearbeite zuerst den Knoten selbst, dann besuche den linken, danach den rechten Teilbaum
  - **Post-Order** (Nebenreihenfolge): Besuche zuerst den linken, danach den rechten Teilbaum, zum Schluss bearbeite den Knoten selbst
  - **In-Order** (symmetrische Reihenfolge): Besuche zuerst den linken Teilbaum, dann bearbeite den Knoten selbst, danach besuche den rechten Teilbaum
- Manchmal auch **Reverse In-Order** (anti-symmetrische Reihenfolge): Rechter Teilbaum, Knoten, dann linker Teilbaum
- Auch das Besuchen nach Tiefenlevel von links nach rechts (**level-order**) ist denkbar

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

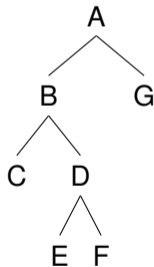
Traversierung

Suchbäume

Zusammenfassung

# Pre-Order Ausgabe eines Baums

- Gebe den Baum *pre-order* aus



- Ausgabe: A B C D E F G

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

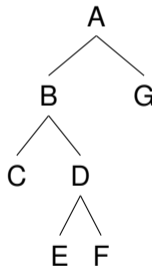
Traversierung

Suchbäume

Zusammenfassung

# Post-Order Ausgabe eines Baums

- Gebe Baum *post-order* aus



- Ausgabe: C E F D B G A

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

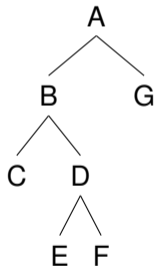
Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

- Gebe Baum *in-order* aus.



- Ausgabe: C B E D F A G

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

```
def postorder[T](tree : BTree[T]) -> list[T]:  
  match tree:  
    case None:  
      return []  
    case Node (m, l, r):  
      return postorder(l) + postorder(r) + [m]
```

```
tree : Node[int|str] = Node('*', Node('+', Node(6), Node(5)),  
                             Node(1))
```

```
postorder(tree)
```

Die *post-order* Ausgabe eines Ausdrucks heißt auch **umgekehrt polnische** oder **Postfix**-Notation (HP-Taschenrechner, Programmiersprachen *Forth* und *PostScript*)

Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

## HP-35



Von Holger Weihe - Eigenes Werk, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=146664>

## Forth

```
: DECADE 10 0 DO
  I .
  LOOP ;
```

## PostScript

```
newpath
100 200 moveto
200 250 lineto
100 300 lineto
2 setlinewidth
stroke
```

Ergebnis:



Der Baum

Binärbäume

Repräsentation

Beispiel

Funktionen auf  
Bäumen

Baumeigenschaften

Traversierung

Suchbäume

Zusammenfassung

# 3 Suchbäume



- Definition
- Suche
- Aufbau

Der Baum

Binärbäume

**Suchbäume**

Definition

Suche

Aufbau

Zusammenfassung

# 3 Suchbäume



- Definition
- Suche
- Aufbau

Der Baum

Binärbäume

Suchbäume

**Definition**

Suche

Aufbau

Zusammenfassung



- *Suchbäume* dienen dazu, Objekte schnell aufzufinden.
- Ein **Suchbaum** ist ein binärer Baum, bei dem jeder Knoten  $k$  die **Suchbaumeigenschaft** erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von  $k$ , alle Markierungen im rechten Teilbaum sind *größer*.
- **Suchen nach einem Objekt  $m$  beginnend beim Knoten  $k$ :**  
Vergleiche  $m$  mit Markierung des aktuellen Knotens  $k$ ,
  - wenn gleich, stoppe und gebe True zurück,
  - wenn  $m$  kleiner ist, suche im linken Teilbaum,
  - wenn  $m$  größer ist, such im rechten Teilbaum.
- Suchzeit ist proportional zur **Höhe des Baums!**  
Im besten Fall *logarithmisch in der Größe des Baums*.

Der Baum

Binärbäume

Suchbäume

Definition

Suche

Aufbau

Zusammenfassung

## Lemma

Ist  $h = h(t)$  die Höhe eines Binärbaums, so gilt für seine Größe  $s(t) \leq 2^{h+1} - 1$ .

## Beweis (Induktion)

IA: Ist der Baum leer, so ist seine Höhe  $-1$  und seine Größe  $0$ .

IS: Besteht ein Baum  $t$  aus einem Knoten und zwei Teilbäumen  $l$  und  $r$  mit Höhen  $h(l)$  und  $h(r)$ , so gilt nach IV  $s(l) \leq 2^{h(l)+1} - 1$  und  $s(r) \leq 2^{h(r)+1} - 1$ .

Wegen  $s(t) = 1 + s(l) + s(r)$  und  $h(t) = 1 + \max(h(l), h(r))$  gilt

$$\blacksquare s(t) = 1 + s(l) + s(r) \leq 1 + (2^{h(l)+1} - 1) + (2^{h(r)+1} - 1) \leq 2 \cdot 2^{\max(h(l)+1, h(r)+1)} - 1 = 2^{h(t)+1} - 1$$

Der Baum

Binärbäume

Suchbäume

Definition

Suche

Aufbau

Zusammenfassung

# 3 Suchbäume



- Definition
- Suche
- Aufbau

Der Baum

Binärbäume

Suchbäume

Definition

**Suche**

Aufbau

Zusammenfassung

```
def search[T : (int, float, str)](tree: BTree[T], item: T) -> bool:
  match tree:
    case None:
      return False
    case Node(m, l, r) if m > item:
      return search(l, item)
    case Node(m, l, r) if m < item:
      return search(r, item)
    case _:      # m == item
      return True
```

```
# smaller values left, bigger values in right subtree
nums = Node(10, Node(5, Node(1), None),
            Node(15, Node(12), Node(20)))
print(search(nums, 12))
```

Der Baum

Binärbäume

Suchbäume

Definition

Suche

Aufbau

Zusammen-  
fassung

# 3 Suchbäume



- Definition
- Suche
- Aufbau

Der Baum

Binärbäume

Suchbäume

Definition

Suche

**Aufbau**

Zusammenfassung

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Ist `tree` leer, so wird der Knoten `Node(item)` zurückgegeben.
- Wenn die Markierung `tree.mark` größer als `item` ist, wird `item` in den linken Teilbaum eingesetzt und der Baum rekonstruiert (das erhält die Suchbaumeigenschaft!).
- Falls `tree.mark` kleiner als `item` ist, entsprechend.
- Falls `tree.mark == item` ist nichts zu tun!

Der Baum

Binärbäume

Suchbäume

Definition

Suche

Aufbau

Zusammenfassung

```
def insert[T : (str, int, float)](
  tree: BTree[T], item: T
) -> Node[T]:
  match tree:
    case None:
      return Node(item)
    case Node(m, l, r) if item < m:
      return Node(m, insert(l, item), r)
    case Node(m, l, r) if m < item:
      return Node(m, l, insert(r, item))
    case _: # m == item
      return tree
```

Der Baum

Binärbäume

Suchbäume

Definition

Suche

Aufbau

Zusammenfassung

```
def insertall[T : (str, int, float)](  
    tree : BTree[T],  
    lst : list[T]  
    ) -> BTree[T]:  
    for key in lst:  
        tree = insert(tree, key)  
    return tree
```

```
bst = insertall(None, [10, 15, 20, 12, 5, 1])
```

Der Baum

Binärbäume

Suchbäume

Definition

Suche

Aufbau

Zusammen-  
fassung



# 4 Zusammenfassung



Der Baum

Binärbäume

Suchbäume

**Zusammenfassung**

- Der **Baum** ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Operationen über Bäumen lassen sich einfach als **rekursive Funktionen** implementieren.
- In einem **Binärbaum** besitzt jeder Knoten genau zwei Teilbäume.
- Es gibt drei Hauptarten der **Traversierung** von Binärbäumen: pre-order, post-order, in-order.
- **Suchbäume** sind Binärbäume, die die Suchbaumeigenschaft besitzen, d.h. im linken Teilbaum sind nur kleinere, im rechten nur größere Markierungen als an der Wurzel.
- Das **Suchen** und **Einfügen** kann durch einfache rekursive Funktionen realisiert werden. **Sortierte Ausgabe** ist auch sehr einfach!

Der Baum

Binärbäume

Suchbäume

Zusammenfassung