Informatik I: Einführung in die Programmierung

9. Bäume

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

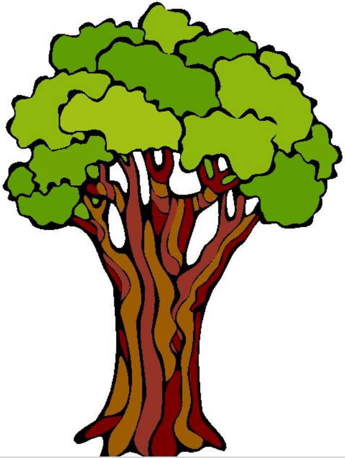
Peter Thiemann

4. Dezember 2019

Der Baum

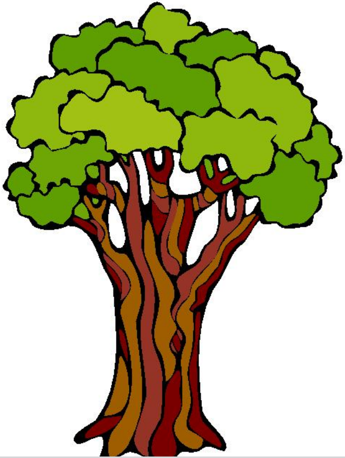
Bäume in der Informatik

- Bäume sind in der Informatik allgegenwärtig.



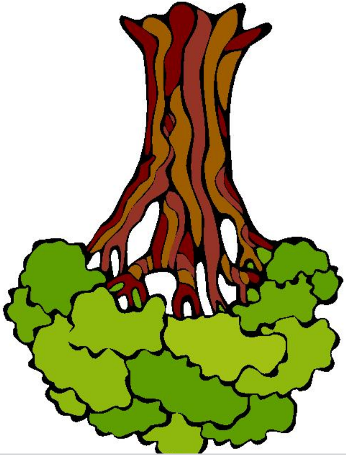
Bäume in der Informatik

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



Bäume in der Informatik

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



Bäume in der Informatik - Definition

- Induktive Definition:

Bäume in der Informatik - Definition

- Induktive Definition:
  - Der leere Baum ist ein Baum.

Bäume in der Informatik - Definition

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

Bäume in der Informatik - Definition

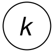
- Induktive Definition:
  - Der leere Baum ist ein Baum.
  - Wenn $t_1, \ldots, t_n$, $n \geq 0$ disjunkte Bäume sind und k ein Knoten, der nicht in $t_1, \ldots, t_n$ vorkommt, dann ist auch die Struktur bestehend aus der Wurzel k mit zugeordneten Teilbäumen $t_1, \ldots, t_n$ ein Baum.
  - Nichts sonst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

Bäume in der Informatik - Definition

- Induktive Definition:
  - 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.
  - Beispiel:

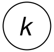
Bäume in der Informatik - Definition

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



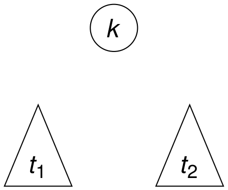
Bäume in der Informatik - Definition

- Induktive Definition:
  - 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.
  - Beispiel:



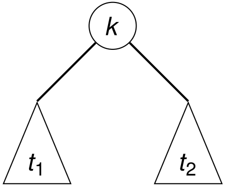
Bäume in der Informatik - Definition

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



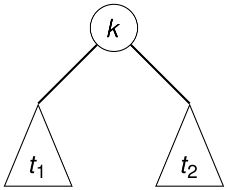
Bäume in der Informatik - Definition

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



Bäume in der Informatik - Definition

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



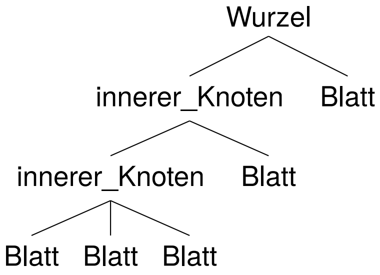
- Beachte: Bäume können auch anders definiert werden und können auch eine andere Gestalt haben (z.B. ungewurzelt).

Terminologie I

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

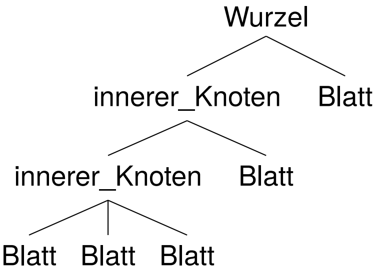
Terminologie I

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



Terminologie I

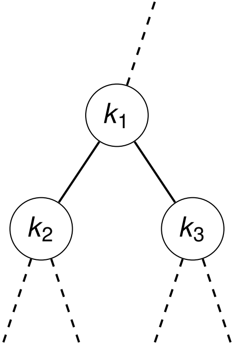
- Alle Knoten, denen keine Teilbäume zugeordnet sind, heißen Blätter.
- Knoten, die keine Blätter sind, heißen innere Knoten.



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

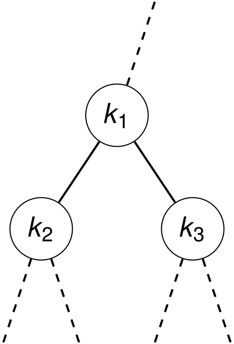
Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:



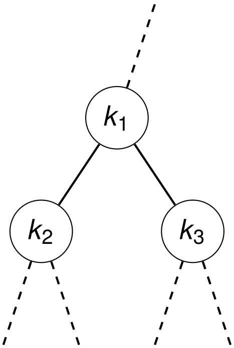
Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,



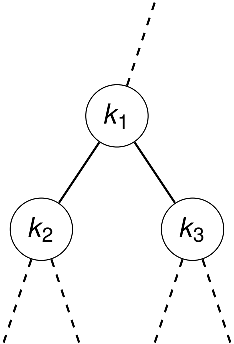
Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,
  - k_1 sowie der Elternknoten von k_1 sowie dessen Elternknoten usw. sind Vorgänger von k_2.



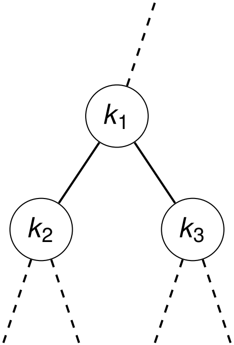
Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,
  - k_1 sowie der Elternknoten von k_1 sowie dessen Elternknoten usw. sind Vorgänger von k_2.
  - k_2 ist Kind von k_1.



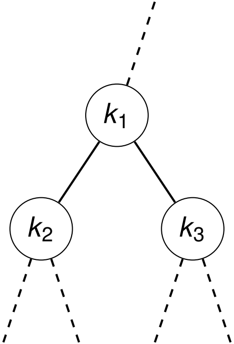
Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,
  - k_1 sowie der Elternknoten von k_1 sowie dessen Elternknoten usw. sind Vorgänger von k_2.
  - k_2 ist Kind von k_1.
  - Alle Kinder von k_1, deren Kinder, usw. sind Nachfolger von k_1.



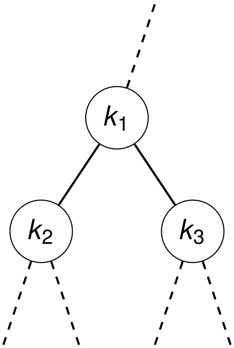
Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,
  - k_1 sowie der Elternknoten von k_1 sowie dessen Elternknoten usw. sind Vorgänger von k_2.
  - k_2 ist Kind von k_1.
  - Alle Kinder von k_1, deren Kinder, usw. sind Nachfolger von k_1.
- Bäume sind oft markiert. Die Markierung weist jedem Knoten eine Marke zu.



Terminologie II

- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,
  - k_1 sowie der Elternknoten von k_1 sowie dessen Elternknoten usw. sind Vorgänger von k_2.
  - k_2 ist Kind von k_1.
  - Alle Kinder von k_1, deren Kinder, usw. sind Nachfolger von k_1.
- 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 $\mu : K \to M$.

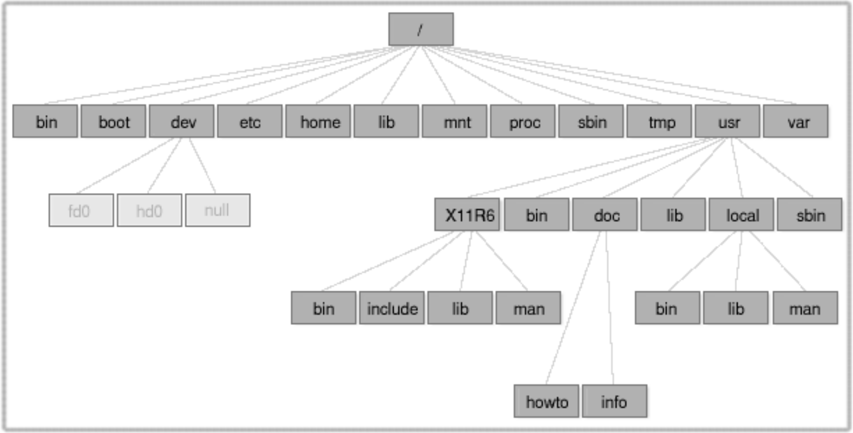


Beispiel: Verzeichnisbaum

In Linux (und anderen Betriebssystemen) ist die Verzeichnisstruktur im Wesentlichen baumartig.

Beispiel: Verzeichnisbaum

In Linux (und anderen Betriebssystemen) ist die Verzeichnisstruktur im Wesentlichen baumartig.

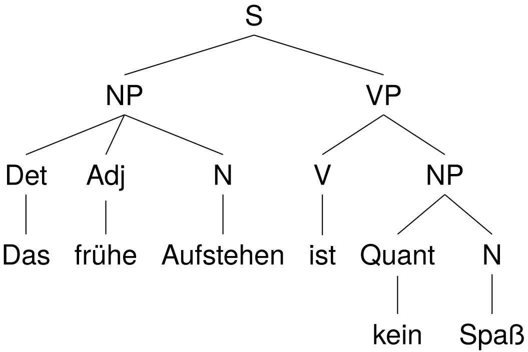


Beispiel: Syntaxbaum

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

Beispiel: Syntaxbaum

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



Beispiel: Ausdrucksbaum

- Bäume können arithmetische (und andere) Ausdrücke so darstellen, dass ihre Auswertung eindeutig (und einfach durchführbar) ist, ohne dass Klammern notwendig sind.

Beispiel: Ausdrucksbaum

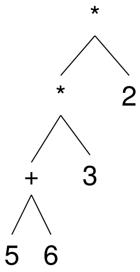
- Bäume können arithmetische (und andere) Ausdrücke so darstellen, dass ihre Auswertung eindeutig (und einfach durchführbar) ist, ohne dass Klammern notwendig sind.
- Beispiel: $(5+6)*3*2$

Beispiel: Ausdrucksbaum

- Bäume können arithmetische (und andere) Ausdrücke so darstellen, dass ihre Auswertung eindeutig (und einfach durchführbar) ist, ohne dass Klammern notwendig sind.
- Beispiel: $(5+6)*3*2$
- Entspricht: $((5+6)*3)*2$

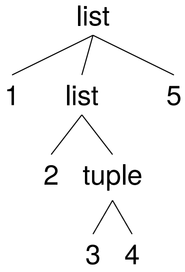
Beispiel: Ausdrucksbaum

- Bäume können arithmetische (und andere) Ausdrücke so darstellen, dass ihre Auswertung eindeutig (und einfach 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:



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]`



Binärbäume

Der Binärbaum

- Der Binärbaum ist ein Spezialfall eines Baumes.

Induktive Definition

Ein Binärbaum ist entweder leer oder besteht aus einem (Wurzel-) Knoten und zwei Teilbäumen, die selbst wieder Binärbäume sind.
Nichts sonst ist ein Binärbaum.

- Im Binärbaum nennen wir einen Knoten
  - ein Blatt, falls beide Teilbäume leer sind
  - einen inneren Knoten, fall mindestens ein Teilbaum vorhanden ist

⇒ i.a. sind von einem Blatt keine weiteren Knoten erreichbar

- Für viele Anwendungsfälle angemessen.
- Funktionen über solchen Bäumen sind einfach definierbar.

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute
  - `mark` enthält die Markierung.

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute
  - `mark` enthält die Markierung.
  - `left` enthält den linken Teilbaum.

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute
  - `mark` enthält die Markierung.
  - `left` enthält den linken Teilbaum.
  - `right` enthält den rechten Teilbaum.

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute
  - `mark` enthält die Markierung.
  - `left` enthält den linken Teilbaum.
  - `right` enthält den rechten Teilbaum.
- Beispiele:

Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute
  - `mark` enthält die Markierung.
  - `left` enthält den linken Teilbaum.
  - `right` enthält den rechten Teilbaum.
- Beispiele:
  - Der Baum bestehend aus dem einzigen Knoten mit der Markierung 8:
    `Node (8, None, None)`

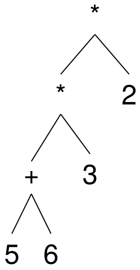
Binärbäume durch Objekte repräsentieren

- Der leere Baum wird durch `None` repräsentiert.
- Jeder andere Knoten wird durch ein `Node`-Objekt repräsentiert.
- Ein `Node`-Objekt besitzt folgende Attribute
  - `mark` enthält die Markierung.
  - `left` enthält den linken Teilbaum.
  - `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))`

Knotenobjekte für Binärbäume


```
class Node:
    def __init__(self, mark, left, right):
        self.mark = mark
        self.left = left
        self.right = right
```

Beispiel: Der Ausdrucksbaum



wird folgendermaßen mit `Node` Objekten dargestellt:

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

Drucken von Bäumen

Funktionsgerüst

```
def tree_str (tree : Node) -> str:
    if tree is None:
        return "fill␣for␣empty"
    else:
        l_str = tree_str (tree.left)
        r_str = tree_str (tree.right)
        return "fill␣for␣node"
```

Funktionsgerüst

```
def tree_str (tree : Node) -> str:
    if tree is None:
        return "fill␣for␣empty"
    else:
        l_str = tree_str (tree.left)
        r_str = tree_str (tree.right)
        return "fill␣for␣node"
```

- `Node`-Objekte enthalten selbst wieder `Node`-Objekte (oder `None`) in den Attributen `left` und `right`.

Drucken von Bäumen

Funktionsgerüst

```
def tree_str (tree : Node) -> str:
    if tree is None:
        return "fill␣for␣empty"
    else:
        l_str = tree_str (tree.left)
        r_str = tree_str (tree.right)
        return "fill␣for␣node"
```

- `Node`-Objekte enthalten selbst wieder `Node`-Objekte (oder `None`) in den Attributen `left` und `right`.
- Zum Ausdrucken eines `Node`-Objekts müssen auch die enthaltenen `Node`-Objekte ausgedruckt werden.

Drucken von Bäumen

Funktionsgerüst

```
def tree_str (tree : Node) -> str:
    if tree is None:
        return "fill␣for␣empty"
    else:
        l_str = tree_str (tree.left)
        r_str = tree_str (tree.right)
        return "fill␣for␣node"
```

- `Node`-Objekte enthalten selbst wieder `Node`-Objekte (oder `None`) in den Attributen `left` und `right`.
- Zum Ausdrucken eines `Node`-Objekts müssen auch die enhaltenen `Node`-Objekte ausgedruckt werden.
- Dafür wird `tree_str` wird induktiv in seiner eigenen Definition verwendet.

Drucken von Bäumen

Funktionsgerüst

```
def tree_str (tree : Node) -> str:
    if tree is None:
        return "fill␣for␣empty"
    else:
        l_str = tree_str (tree.left)
        r_str = tree_str (tree.right)
        return "fill␣for␣node"
```

- `Node`-Objekte enthalten selbst wieder `Node`-Objekte (oder `None`) in den Attributen `left` und `right`.
- Zum Ausdrucken eines `Node`-Objekts müssen auch die enhaltenen `Node`-Objekte ausgedruckt werden.
- Dafür wird `tree_str` wird induktiv in seiner eigenen Definition verwendet.
- Ok, weil `tree_str` induktiv auf den Teilbäumen aufgerufen wird!

Drucken von Bäumen erfolgt induktiv

- Die induktiven Aufrufe `tree_str (tree.left)` und `tree_str (tree.left)` erfolgen auf den Kindern des Knoten.
- Ergibt sich zwangsläufig aus der induktiven Definition!
- Induktive Aufrufe auf den Teilbäumen sind Teil des Funktionsgerüsts, sobald eine baumartige Struktur bearbeitet werden soll.

- Die Alternative "`tree is None`" ergibt sich daraus, dass ein `tree` entweder `None` oder ein `Node`-Objekt ist.

- Alle Funktionen auf Binärbäumen verwenden dieses Gerüst.

```python
def tree_str (tree : Node) -> str:
    if tree is None:
        return "None"
    else:
        return ("Node("
            + repr(tree.mark) + ", "
            + tree_str (tree.left) + ", "
            + tree_str (tree.right) + ")")
```

Visualisierung

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

- Die Tiefe eines Knotens *k*, *depth(k)*, ist sein Abstand von der Wurzel:

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

- Die Tiefe eines Knotens *k*, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls *k* die Wurzel ist,

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.
- Die Höhe eines Baumes t, *height(t)*, ist die maximale Tiefe über alle Blätter:

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.
- Die Höhe eines Baumes t, *height(t)*, ist die maximale Tiefe über alle Blätter:
  - -1 für den leeren Baum,

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.
- Die Höhe eines Baumes t, *height(t)*, 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.

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.
- Die Höhe eines Baumes t, *height(t)*, 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 t, *size(t)*, ist die Anzahl seiner Knoten.

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.
- Die Höhe eines Baumes t, *height(t)*, 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 t, *size(t)*, ist die Anzahl seiner Knoten.
  - 0 für den leeren Baum,

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

- Die Tiefe eines Knotens k, *depth(k)*, ist sein Abstand von der Wurzel:
  - 0, falls k die Wurzel ist,
  - $i+1$, wenn i die Tiefe des Elternknotens ist.
- Die Höhe eines Baumes t, *height(t)*, 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 t, *size(t)*, ist die Anzahl seiner Knoten.
  - 0 für den leeren Baum,
  - $s+1$, wenn s die Summe der Größen der Teilbäume ist.

Induktive Definition von Höhe und Größe von Binärbäumen

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

Induktive Definition von Höhe und Größe von Binärbäumen

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

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

Induktive Definition von Höhe und Größe von Binärbäumen

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

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

Bemerkung

Die Tiefe eines Knotens kann mit dieser Baumrepräsentation nicht bestimmt werden, weil sie keinen Zugriff auf den Elternknoten erlaubt.

Funktionen für Höhe und Größe

Höhe und Größe von Binärbäumen

```
def height(tree : Node) -> int:
    if (tree is None):
        return -1
    else:
        return(max(height(tree.left),
                   height(tree.right)) + 1)

def size(tree : Node) -> int:
    if (tree is None):
        return 0
    else:
        return(size(tree.left)
            + size(tree.right) + 1)

tree = Node('*', Node('+', Node(6, None, None),
                           Node(5, None, None)),
                 Node(1, None, None))
```

Traversierung von Bäumen

- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.

Traversierung von Bäumen

- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.
- Übliche Vorgehensweisen (Traversierungen) für den Besuch eines Knotens:

Traversierung von Bäumen

- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.
- Übliche Vorgehensweisen (Traversierungen) für den Besuch eines Knotens:
  - Pre-Order (Hauptreihenfolge): Zuerst wird der Knoten selbst bearbeitet, dann der linke, danach der rechte Teilbaum besucht

Traversierung von Bäumen

- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.
- Übliche Vorgehensweisen (Traversierungen) für den Besuch eines Knotens:
  - Pre-Order (Hauptreihenfolge): Zuerst wird der Knoten selbst bearbeitet, dann der linke, danach der rechte Teilbaum besucht
  - Post-Order (Nebenreihenfolge): Zuerst wird der linke, danach der rechte Teilbaum besucht, zum Schluss der Knoten selbst bearbeitet

Traversierung von Bäumen

- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.
- Übliche Vorgehensweisen (Traversierungen) für den Besuch eines Knotens:
  - Pre-Order (Hauptreihenfolge): Zuerst wird der Knoten selbst bearbeitet, dann der linke, danach der rechte Teilbaum besucht
  - Post-Order (Nebenreihenfolge): Zuerst wird der linke, danach der rechte Teilbaum besucht, zum Schluss der Knoten selbst bearbeitet
  - In-Order (symmetrische Reihenfolge): Zuerst wird der linke Teilbaum besucht, dann der Knoten bearbeitet, danach der rechte Teilbaum besucht

Traversierung von Bäumen

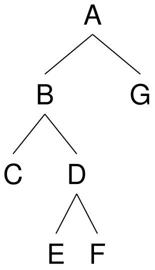
- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.
- Übliche Vorgehensweisen (Traversierungen) für den Besuch eines Knotens:
  - Pre-Order (Hauptreihenfolge): Zuerst wird der Knoten selbst bearbeitet, dann der linke, danach der rechte Teilbaum besucht
  - Post-Order (Nebenreihenfolge): Zuerst wird der linke, danach der rechte Teilbaum besucht, zum Schluss der Knoten selbst bearbeitet
  - In-Order (symmetrische Reihenfolge): Zuerst wird der linke Teilbaum besucht, dann der Knoten bearbeitet, danach der rechte Teilbaum besucht
- Manchmal auch Reverse In-Order (anti-symmetrische Reihenfolge): Rechter Teilbaum, Knoten, dann linker Teilbaum

Traversierung von Bäumen

- Oft sollen alle Knoten eines Baumes besucht werden, wobei an jedem Knoten eine Aufgabe bearbeitet werden muss.
- Übliche Vorgehensweisen (Traversierungen) für den Besuch eines Knotens:
  - Pre-Order (Hauptreihenfolge): Zuerst wird der Knoten selbst bearbeitet, dann der linke, danach der rechte Teilbaum besucht
  - Post-Order (Nebenreihenfolge): Zuerst wird der linke, danach der rechte Teilbaum besucht, zum Schluss der Knoten selbst bearbeitet
  - In-Order (symmetrische Reihenfolge): Zuerst wird der linke Teilbaum besucht, dann der Knoten bearbeitet, danach der rechte Teilbaum besucht
- 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

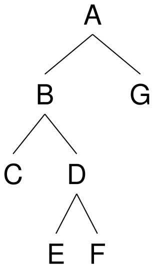
Pre-Order Ausgabe eines Baums

- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



Pre-Order Ausgabe eines Baums

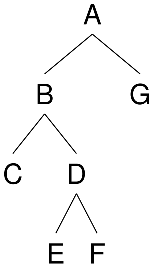
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A

Pre-Order Ausgabe eines Baums

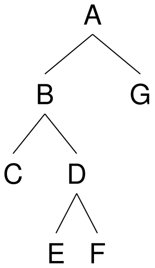
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A

Pre-Order Ausgabe eines Baums

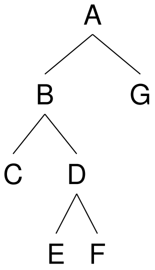
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A B

Pre-Order Ausgabe eines Baums

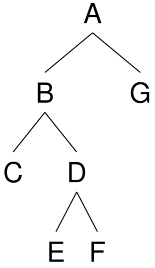
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A B C

Pre-Order Ausgabe eines Baums

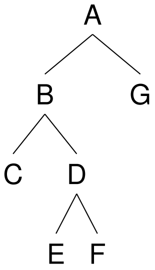
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A B C D

Pre-Order Ausgabe eines Baums

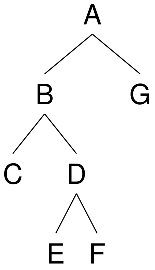
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A B C D E

Pre-Order Ausgabe eines Baums

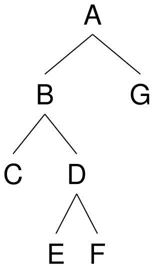
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A B C D E F

Pre-Order Ausgabe eines Baums

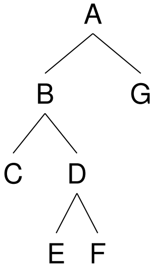
- Gebe Baum *pre-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: A B C D E F G

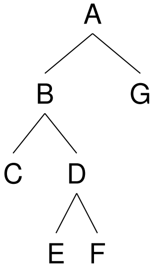
Post-Order Ausgabe eines Baums

- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



Post-Order Ausgabe eines Baums

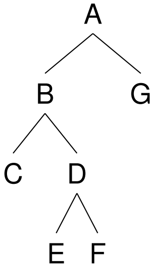
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C

Post-Order Ausgabe eines Baums

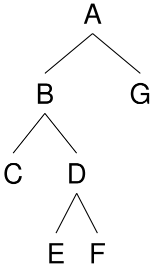
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C

Post-Order Ausgabe eines Baums

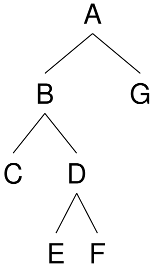
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C E

Post-Order Ausgabe eines Baums

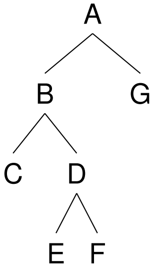
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C E F

Post-Order Ausgabe eines Baums

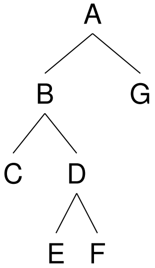
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C E F D

Post-Order Ausgabe eines Baums

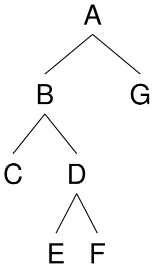
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C E F D B

Post-Order Ausgabe eines Baums

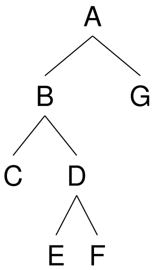
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C E F D B G

Post-Order Ausgabe eines Baums

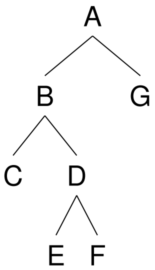
- Gebe Baum *post-order* aus (Aufgabe am Knoten: Ausgabe der Markierung)



- Ausgabe: C E F D B G A

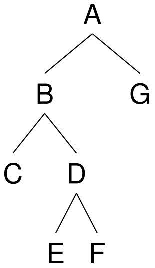
In-Order Ausgabe eines Baums

- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



In-Order Ausgabe eines Baums

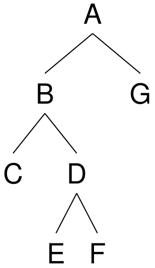
- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C

In-Order Ausgabe eines Baums

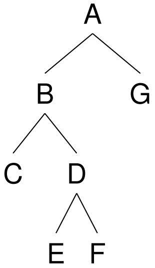
- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C

In-Order Ausgabe eines Baums

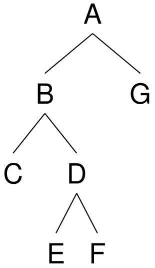
- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C B

In-Order Ausgabe eines Baums

- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).

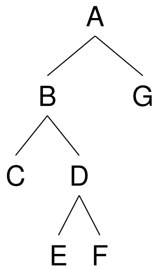


- Ausgabe: C B E

In-Order Ausgabe eines Baums

Der Baum
Binärbäume
Repräsentation
Beispiel
Funktionen auf Bäumen
Baumeigenschaften
Traversierung
Zusammenfassung
Suchbäume

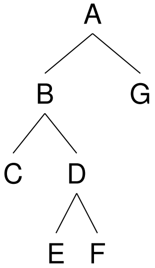
- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C B E D

In-Order Ausgabe eines Baums

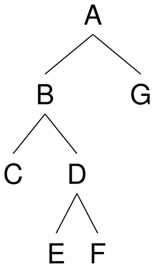
- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C B E D F

In-Order Ausgabe eines Baums

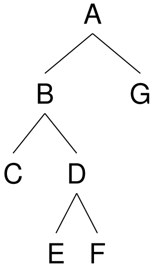
- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C B E D F A

In-Order Ausgabe eines Baums

- Gebe Baum *in-order* aus (Aufgabe am Knoten: Ausgabe der Markierung).



- Ausgabe: C B E D F A G

Post-Order Ausgabe

```
def postorder(tree : Node):
    if tree is None:
        pass
    else:
        postorder(tree.left)
        postorder(tree.right)
        print(tree.mark)

def leaf (m) -> Node:
    return Node (m, None, None)

tree = Node('*', Node('+', leaf(6), leaf(5)),
                 leaf(1))
postorder(tree)
```

Post-Order Ausgabe

```
def postorder(tree : Node):
    if tree is None:
        pass
    else:
        postorder(tree.left)
        postorder(tree.right)
        print(tree.mark)

def leaf (m) -> Node:
    return Node (m, None, None)

tree = Node('*', Node('+', leaf(6), leaf(5)),
                 leaf(1))
postorder(tree)
```

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

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Verschiedene Definitionen:

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Verschiedene Definitionen:
  - induktive Definition (hier; einfach zum Programmieren),

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Verschiedene Definitionen:
  - induktive Definition (hier; einfach zum Programmieren),
  - graphentheoretische Definition (später; komplexere Algorithmen)

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Verschiedene Definitionen:
  - induktive Definition (hier; einfach zum Programmieren),
  - graphentheoretische Definition (später; komplexere Algorithmen)
- Binärbäume sind entweder leer oder bestehen aus einem Wurzelknoten mit genau zwei Teilbäumen, die wieder Binärbäume sind.

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Verschiedene Definitionen:
  - induktive Definition (hier; einfach zum Programmieren),
  - graphentheoretische Definition (später; komplexere Algorithmen)
- Binärbäume sind entweder leer oder bestehen aus einem Wurzelknoten mit genau zwei Teilbäumen, die wieder Binärbäume sind.
- Operationen über (Binär-)Bäumen lassen sich als induktive Funktionen implementieren.

- Der Baum ist eine Struktur, die in der Informatik allgegenwärtig ist.
- Verschiedene Definitionen:
  - induktive Definition (hier; einfach zum Programmieren),
  - graphentheoretische Definition (später; komplexere Algorithmen)
- Binärbäume sind entweder leer oder bestehen aus einem Wurzelknoten mit genau zwei Teilbäumen, die wieder Binärbäume sind.
- Operationen über (Binär-)Bäumen lassen sich als induktive Funktionen implementieren.
- Hauptarten der Traversierung von Binärbäumen: Pre-order, In-order, Post-order.

Suchbäume

Key-Value-Store

- Ein Key-Value-Store ist eine endliche Abbildung $W : K \hookrightarrow V$ mit
  - K eine Menge von Schlüsseln (*key*)
  - V eine Menge von Werten (*value*)
- Endlich heißt, dass die Abbildung nur für endlich viele Schlüssel $k \in K$ definiert ist (angedeutet durch $\hookrightarrow$ für partielle Abbildung)
- Oft auch Wörterbuch (dictionary) genannt

ugs Bedeutung/Beispiel

- Ein Wörterbuch ordnet einem Wort seine Bedeutung bzw. seine Übersetzung in eine andere Sprache zu
- Schlüsselmenge = Wertemenge = Menge der Strings
- Das deutsche Wörterbuch hat nur Einträge für die (endlich vielen) Wörter der deutschen Sprache

Operationen eines Key-Value-Store

- Einfügen eines Schlüssels und des zugehörigen Werts
- Suchen nach einem Schlüssel; Rückgabe des zugehörigen Werts oder Fehlanzeige
- Weitere Operationen: Löschen, Werte bearbeiten, etc

Operationen eines Key-Value-Store

- Einfügen eines Schlüssels und des zugehörigen Werts
- Suchen nach einem Schlüssel; Rückgabe des zugehörigen Werts oder Fehlanzeige
- Weitere Operationen: Löschen, Werte bearbeiten, etc

Einfachste Realisierung

- Repräsentation der Abbildung als Liste von Paaren aus Schlüssel und Wert

```
words = [("eins", "one"), ("zwei", "two"), ("drei", "tree"), ...]
```

- Einfügen: neues Paar anfügen
- Suchen: Durchlaufen der Liste; Vergleich mit erster Komponente der Paare
- Nachteil: Dauer der Suche proportional zur Anzahl der Einträge

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von k, alle Markierungen im rechten Teilbaum sind *größer*.

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von k, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel m

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von k, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel m
  - Beginne an der Wurzel

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von k, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel m
  - Beginne an der Wurzel
  - Vergleiche m mit Markierung am aktuellem Knoten,

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von k, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel m
  - Beginne an der Wurzel
  - Vergleiche m mit Markierung am aktuellem Knoten,
    - wenn gleich, stoppe und gebe True zurück,

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten *k* die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von *k*, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel *m*
  - Beginne an der Wurzel
  - Vergleiche *m* mit Markierung am aktuellem Knoten,
    - wenn gleich, stoppe und gebe True zurück,
    - wenn *m* kleiner ist, suche im linken Teilbaum,

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten *k* die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von *k*, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel *m*
  - Beginne an der Wurzel
  - Vergleiche *m* mit Markierung am aktuellem Knoten,
    - wenn gleich, stoppe und gebe True zurück,
    - wenn *m* kleiner ist, suche im linken Teilbaum,
    - wenn *m* größer ist, such im rechten Teilbaum.

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten *k* die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von *k*, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel *m*
  - Beginne an der Wurzel
  - Vergleiche *m* mit Markierung am aktuellem Knoten,
    - 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!

Suchbäume

- Suchbäume realisieren Wörterbücher und dienen dazu, Schlüssel schnell wieder zu finden.
- Ein Suchbaum ist ein binärer Baum, bei dem jeder Knoten k die Suchbaumeigenschaften erfüllt:
  - Alle Markierungen im linken Teilbaum sind *kleiner* als die Markierung von k, alle Markierungen im rechten Teilbaum sind *größer*.
- Suchen nach Schlüssel m
  - Beginne an der Wurzel
  - Vergleiche m mit Markierung am aktuellem Knoten,
    - 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*.

```
def search(tree : Node, item) -> bool:
    if tree is None:
        return False
    elif tree.mark == item:
        return True
    elif tree.mark > item:
        return search(tree.left, item)
    else:
        return search(tree.right, item)

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

Visualisierung

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Laut Definition: `tree` ist entweder leer oder ein `Node`-Objekt

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Laut Definition: `tree` ist entweder leer oder ein `Node`-Objekt
- Ist `tree` leer, so wird der Knoten `leaf(item)` zurückgegeben.

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Laut Definition: `tree` ist entweder leer oder ein `Node`-Objekt
- Ist `tree` leer, so wird der Knoten `leaf(item)` zurückgegeben.
- Ist `tree` ein `Node`-Objekt, so betrachte die Markierung `tree.mark`

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Laut Definition: `tree` ist entweder leer oder ein `Node`-Objekt
- Ist `tree` leer, so wird der Knoten `leaf(item)` zurückgegeben.
- Ist `tree` ein `Node`-Objekt, so betrachte die Markierung `tree.mark`
    - Wenn die Marke größer als `item` ist, wird `item` in den linken Teilbaum eingesetzt und der Baum rekonstruiert (das erhält die Suchbaumeigenschaft!).

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Laut Definition: `tree` ist entweder leer oder ein `Node`-Objekt
- Ist `tree` leer, so wird der Knoten `leaf(item)` zurückgegeben.
- Ist `tree` ein `Node`-Objekt, so betrachte die Markierung `tree.mark`
  - Wenn die Marke 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 mit rechtem Teilbaum.

Aufbauen eines Suchbaums

Immutable — Vorhandene Struktur bleibt unverändert

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Laut Definition: `tree` ist entweder leer oder ein `Node`-Objekt
- Ist `tree` leer, so wird der Knoten `leaf(item)` zurückgegeben.
- Ist `tree` ein `Node`-Objekt, so betrachte die Markierung `tree.mark`
  - Wenn die Marke 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 mit rechtem Teilbaum.
  - Falls `tree.mark == item` müssen wir nichts machen.

```
def insert(tree : Node, item) -> Node:
    if tree is None:
        return leaf(item)
    elif tree.mark > item:
        return Node (tree.mark,
                     insert (tree.left, item),
                     tree.right)
    elif tree.mark < item:
        return Node (tree.mark,
                     tree.left,
                     insert(tree.right, item))
    else:
        return tree
```

Aufbauen eines Suchbaums


```
def insertall (tree : Node, lst : list) -> Node:
    for key in lst
        tree = insert (tree, key)
    return tree

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

Aufbauen eines Suchbaums

Mutable — Vorhandene Struktur wird verändert

```
1 def insertm(tree : Node, item) -> Node:
2     if tree is None:
3         return leaf(item)
4     if tree.mark > item:
5         tree.left = insertm(tree.left, item)
6     elif tree.mark < item:
7         tree.right = insertm(tree.right, item)
8     return tree
```

Unterschied zur immutable Version

- In Zeile 5 wird der linke Teilbaum durch einen neuen Suchbaum ersetzt.
- In Zeile 7 wird der rechte Teilbaum durch einen neuen Suchbaum ersetzt.
- Diese Änderungen sind auch im Argument von `insertm` sichtbar.

Aufbauen eines Suchbaums

Mutable — Vorhandene Struktur wird verändert


```
def insertmall (tree : Node, lst : list) -> Node:
    for key in lst
        tree = insertm (tree, key)
    return tree

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

Treesort

- Ein einfaches Verfahren zum Sortieren einer Liste
- Einlesen der Liste in einen Suchbaum mit `insertall`
- In-order Traversierung des Suchbaums; dabei Aufbau der Ausgabeliste von links nach rechts gemäß der Traversierung
- Die Ausgabe ist sortiert aufgrund der Suchbaumeigenschaft!

Zusammenfassung

- Ein Key-Value-Store funktioniert wie ein Wörterbuch. Er realisiert eine endliche Abbildung von Schlüsseln auf Werte.

Zusammenfassung

- Ein Key-Value-Store funktioniert wie ein Wörterbuch. Er realisiert eine endliche Abbildung von Schlüsseln auf Werte.
- Ein Key-Value-Store unterstützt mindestens die Operationen Einfügen, Suchen und Löschen.

Zusammenfassung

- Ein Key-Value-Store funktioniert wie ein Wörterbuch. Er realisiert eine endliche Abbildung von Schlüsseln auf Werte.
- Ein Key-Value-Store unterstützt mindestens die Operationen Einfügen, Suchen und Löschen.
- Suchbäume realisieren Key-Value-Stores.

Zusammenfassung

- Ein Key-Value-Store funktioniert wie ein Wörterbuch. Er realisiert eine endliche Abbildung von Schlüsseln auf Werte.
- Ein Key-Value-Store unterstützt mindestens die Operationen Einfügen, Suchen und Löschen.
- Suchbäume realisieren Key-Value-Stores.
- Suchbäume sind Binärbäume, die die Suchbaumeigenschaft besitzen, d.h. für jeden Knoten k befinden sich im linken Teilbaum nur kleinere, im rechten nur größere Markierungen als an k.

Zusammenfassung

- Ein Key-Value-Store funktioniert wie ein Wörterbuch. Er realisiert eine endliche Abbildung von Schlüsseln auf Werte.
- Ein Key-Value-Store unterstützt mindestens die Operationen Einfügen, Suchen und Löschen.
- Suchbäume realisieren Key-Value-Stores.
- Suchbäume sind Binärbäume, die die Suchbaumeigenschaft besitzen, d.h. für jeden Knoten k befinden sich im linken Teilbaum nur kleinere, im rechten nur größere Markierungen als an k.
- Das Suchen und Einfügen kann durch einfache rekursive Funktionen realisiert werden.

Zusammenfassung

- Ein Key-Value-Store funktioniert wie ein Wörterbuch. Er realisiert eine endliche Abbildung von Schlüsseln auf Werte.
- Ein Key-Value-Store unterstützt mindestens die Operationen Einfügen, Suchen und Löschen.
- Suchbäume realisieren Key-Value-Stores.
- Suchbäume sind Binärbäume, die die Suchbaumeigenschaft besitzen, d.h. für jeden Knoten *k* befinden sich im linken Teilbaum nur kleinere, im rechten nur größere Markierungen als an *k*.
- Das Suchen und Einfügen kann durch einfache rekursive Funktionen realisiert werden.
- Sortierte Ausgabe ist auch sehr einfach!