Informatik I: Einführung in die Programmierung

10. Bäume

Albert-Ludwigs-Universität Freiburg

Prof. Dr. Peter Thiemann

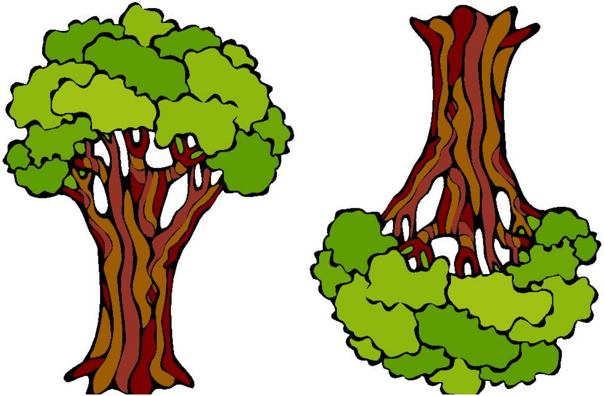
29. November 2022

1 Der Baum

- Definition
- Terminologie
- Beispiele

Bäume in der Informatik

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



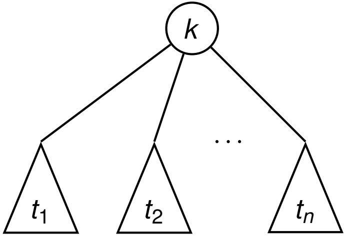
1 Der Baum

- Definition
- Terminologie
- Beispiele

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: $\square$

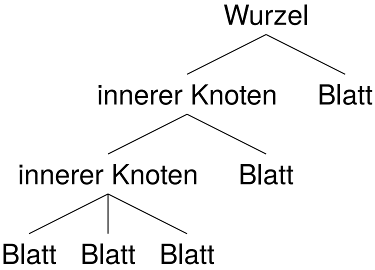


1 Der Baum

- Definition
- Terminologie
- Beispiele

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

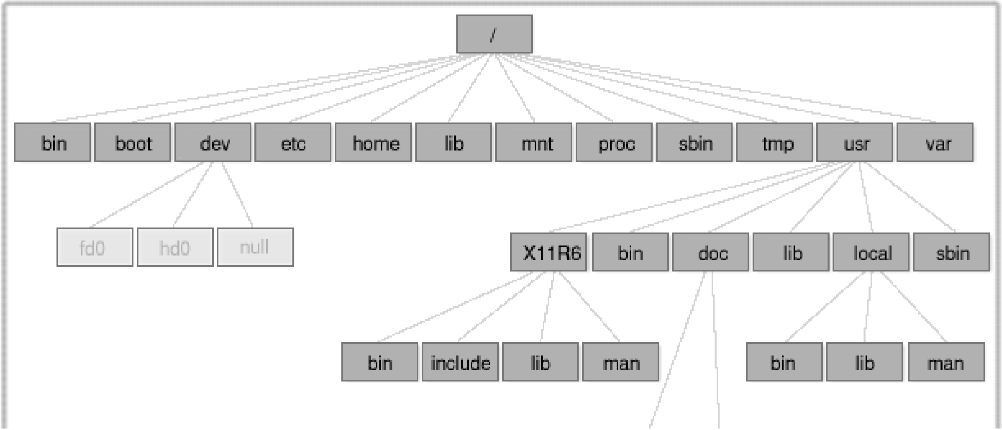
- Wenn k_1 ein Knoten und k_2 die Wurzel eines zugeordneten Teilbaums ist, dann gilt:
  - k_1 ist Elternknoten von k_2,
  - Alle Elternknoten von k_2, deren Elternknoten usw. sind Vorgänger von k_2.
  - k_2 ist Kind von k_1.
  - Alle Kinder von k_1, deren Kinder, usw. sind Nachfolger von k_1.
- Bäume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

1 Der Baum

- Definition
- Terminologie
- Beispiele

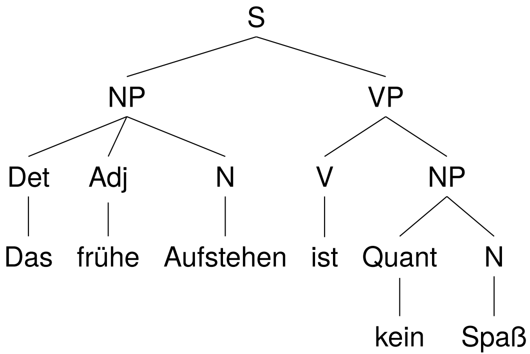
Beispiel: Verzeichnisbaum

In vielen Betriebssystemen ist die Verzeichnisstruktur im Wesentlichen baumartig.



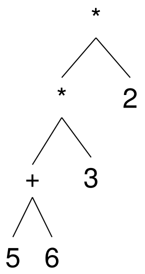
Beispiel: Syntaxbaum

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



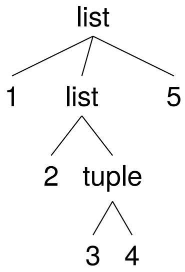
Beispiel: Ausdrucksbaum

- 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:



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 Binärbaum

- 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.

2 Binärbäume

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

Binärbäume durch Objekte repräsentieren

- 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))`

Baumobjekte

```python
from typing import Any, Optional
@dataclass
class Node:
    mark  : Any
    left  : Optional['Node']
    right : Optional['Node']
```

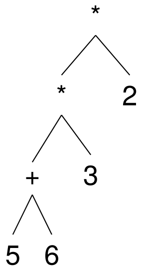
Bemerkung zu den Typannotationen

- `Any`: ein Objekt von beliebigem Typ
- `Optional[t]`: entweder `t` oder `None` (aber nichts anderes)
- Der Typ `Node` existiert erst **nach** Ausführung der `class`-Anweisung. Der String `'Node'` in der Typannotation wird rückwirkend durch den Typ `Node` ersetzt.

2 Binärbäume

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

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))
```

2 Binärbäume

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

Drucken von Bäumen

Funktionsgerüst

```
def tree_str(tree : Optional[Node]) -> str:
    match tree:
        case None:
            return "fill in"
        case Node (m, l, r):
            l_str = tree_str(l)
            r_str = tree_str(r)
            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 enhaltenen `Node` Objekte ausgedruckt werden.
- `tree_str` ist rekursiv, es wird in seiner eigenen Definition aufgerufen!

Drucken von Bäumen erfolgt rekursiv

- Die rekursiven Aufrufe `tree_str (tree.left)` und `tree_str (tree.left)` 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.

Drucken von Bäumen

Funktionsdefinition

```
def tree_str(tree : Optional[Node]) -> string:
    match tree:
        case None:
            return "None"
        case Node (m, l, r):
            return ("Node("
                + repr(m) + ", "
                + tree_str (l) + ", "
                + tree_str (r) + ")")
```

2 Binärbäume

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

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.

Induktive Definition von Höhe und Größe von Binärbäumen

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

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

Funktionen für Höhe und Größe

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

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

2 Binärbäume

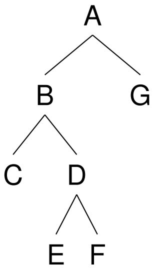
- Repräsentation
- Beispiel
- Funktionen auf Bäumen
- Baumeigenschaften
- Traversierung

Traversierung von Bäumen

- 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

Pre-Order Ausgabe eines Baums

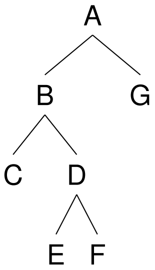
- Gebe den Baum *pre-order* aus



- Ausgabe: A B C D E F G

Post-Order Ausgabe eines Baums

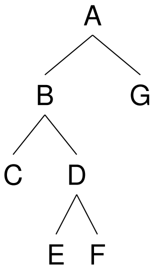
- Gebe Baum *post-order* aus



- Ausgabe: C E F D B G A

In-Order Ausgabe eines Baums

- Gebe Baum *in-order* aus.



- Ausgabe: C B E D F A G

Ausdrucken in post-order

```
def postorder(tree : Optional[Node]):
    match tree:
        case Node (m, l, r):
            postorder(l)
            postorder(r)
            print(m)
def leaf(m : Any) -> Node:
    return Node(m, None, None)
tree = Node('*', Node('+', leaf(6), leaf(5)),
                 leaf(1))
postorder(tree)
```

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

3 Suchbäume

- Definition
- Suche
- Aufbau

3 Suchbäume

- Definition
- Suche
- Aufbau

Suchbäume

- *Suchbäume* dienen dazu, Objekte schnell wieder zu finden.
- 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*.

Höhe und Größe eines Binärbaums

Lemma

Ist h die Höhe eines Binärbaums, so ist seine Größe kleiner gleich $2^{h+1} - 1$.

Beweis (Induktion)

Ist der Baum leer, so ist seine Höhe -1 und seine Größe 0.
Besteht ein Baum t aus einem Knoten und zwei Teilbäumen l und r mit Höhen $h(l)$ und $h(r)$, so gilt nach Induktionsvoraussetzung $s(l) \leq 2^{h(l)+1} - 1$ 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

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

3 Suchbäume

- Definition
- Suche
- Aufbau

Suche im Suchbaum

```
def search(tree : Optional[Node], item : Any) -> 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

3 Suchbäume

- Definition
- Suche
- Aufbau

Aufbauen eines Suchbaums

Immutable — unveränderlich

- Aufruf `insert(tree, item)` für das Einsortieren von `item` in `tree`
- Ist `tree` leer, so wird der Knoten `leaf(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!

Suchbaumaufbau

Immutable — unveränderlich

```
def insert(
        tree : Optional[Node], item : Any
          ) -> 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
```

Suchbaumaufbau

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

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

4 Zusammenfassung

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!