



Informatik II
Bäume



G. Zachmann
Clausthal University, Germa
zach@in.tu-clausthal.de

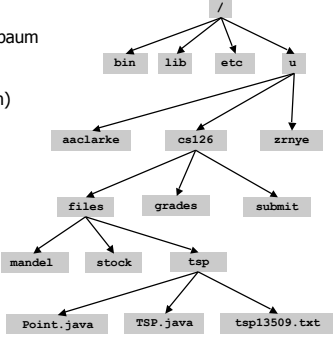
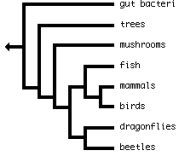
Beispiele

- Stammbaum



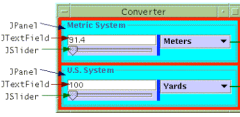
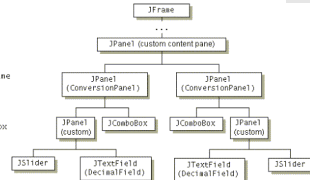
G. Zachmann Informatik 2 - SS 06 BBume 2

- Stammbaum
- Parse tree, Rekursionsbaum
- Unix file hierarchy
- Stammbaum (Evolution)

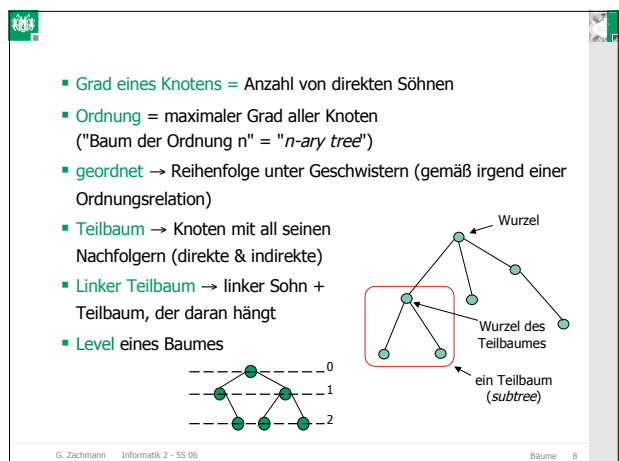
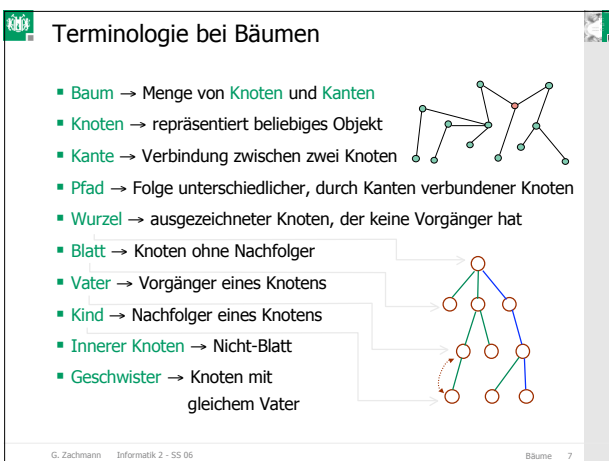
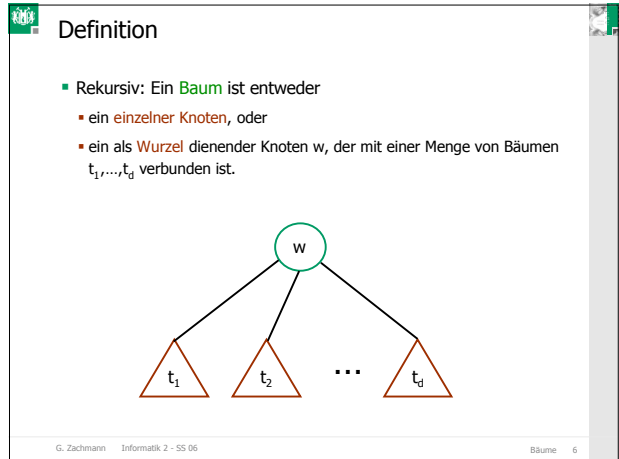
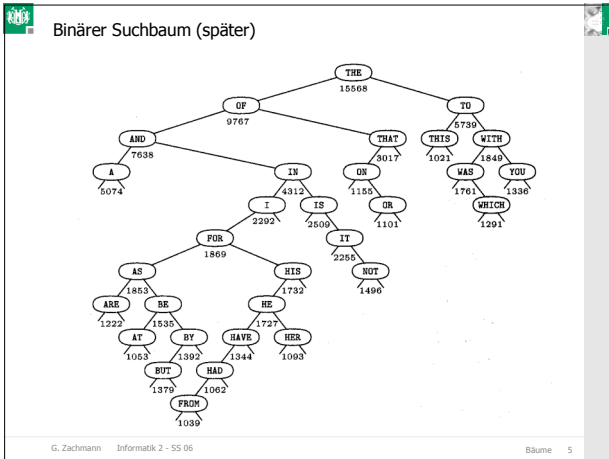
G. Zachmann Informatik 2 - SS 06 Bäume 3

- Stammbaum
- Parse tree, Rekursionsbaum
- Unix file hierarchy
- Stammbaum (Evolution)
- GUI containment hierarchy

Reference: <http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html>

G. Zachmann Informatik 2 - SS 06 BBume 4



Baumtiefe

- Definition: **Tiefe eines Knotens**
 - Länge des Pfades von der Wurzel zu dem Knoten
 - eindeutig, da es nur einen Pfad bei Bäumen gibt
 - dabei zählt man die Knoten entlang des Pfades
 - Wurzel: Tiefe 1 (manchmal auch Tiefe 0)
 - 1. Schicht: Tiefe 2, etc.
- Definition: **Tiefe eines Baumes**
 - leerer Baum: Tiefe 0
 - ansonsten: Maximum der Tiefe seiner Knoten

G. Zachmann Informatik 2 - SS 06 Bäume 9

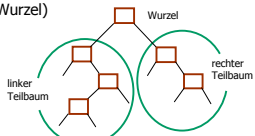
Eigenschaften

- Von der Wurzel gibt es zu jedem Knoten genau einen Pfad
- Für je zwei verschiedene Knoten existiert genau ein Pfad, der sie verbindet.
 - ⇒ jeder beliebige Knoten kann Wurzel sein
- Ein Baum mit n Knoten hat $n-1$ Kanten
 - Beweis von Eigenschaft 3 durch Induktion:
 - Induktionsanfang: $n=1 \rightarrow 0$ Kanten
 - Induktionsschritt: $n>1$
 - Baum hat k Kinder, mit je n_i Knoten und $n_1 + \dots + n_k = n-1$
 - Per Induktionsannahme hat jeder Teilbaum n_i-1 Kanten
 - Zusammen also $n-1-k$ Kanten
 - Dazu k Kanten von den Wurzeln der Teilbäume zur Wurzel \rightarrow Behauptung

G. Zachmann Informatik 2 - SS 06 Bäume 10

Binärbäume

- Wichtiger Spezialfall: jeder Knoten hat höchstens zwei Kinder
 - = Baum der Ordnung 2 = Binärbaum
- alternative **rekursive** Definition:
 - ein Binärbaum ist entweder leer
 - oder besteht aus einem Knoten (Wurzel) und zwei Binärbäumen (linker und rechter Teilbaum)



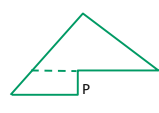
- Wichtige (einfache) Eigenschaft:
In einem Baum, in dem jeder Knoten entweder genau 2 Kinder hat oder keines (Blatt), gilt:

$$\# \text{ Blätter} = \# \text{ innerer Knoten} + 1$$

G. Zachmann Informatik 2 - SS 06 Bäume 11

Vollständige Bäume

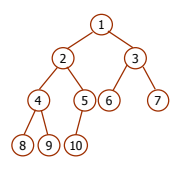
- Definition:
Ein **vollständiger Baum** ist ein binärer Baum B mit folgenden Eigenschaften:
 - für jedes k mit $k < \text{Tiefe}(B)$ gilt, die k -te Schicht ist voll besetzt; und,
 - die letzte Schicht ist von links nach rechts bis zu einem Knoten P besetzt
- Achtung: manchmal abweichende Def.: jede Schicht muß besetzt sein!
- Die Höhe eines vollständigen binären Baumes mit n Knoten ist $\lceil \log_2(n+1) \rceil$



G. Zachmann Informatik 2 - SS 06 Bäume 12

Nummerierung der Knoten

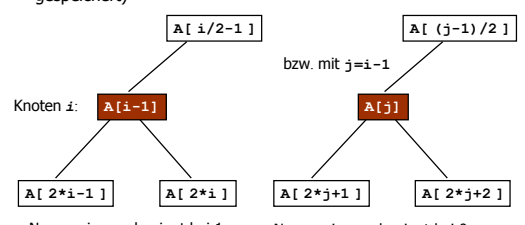
- Von oben nach unten, von links nach rechts, beginnend bei 1
- Beobachtung:
 - ein Knoten i hat immer die Nachfolger $2i$ und $2i+1$
 - Vater ist immer Knoten Nummer $\lfloor i/2 \rfloor$
- Fazit: Knoten können in einem Array abgelegt werden
- Achtung: Indizierung beginnt bei 0
 - Knoten i in Array-Element $A[i-1]$
- Frage: funktioniert ein ähnliches Schema auch, wenn man die Knoten selbst mit 0 beginnend nummeriert?



G. Zachmann Informatik 2 - SS 06 Bäume 13

Speichern eines vollständigen Baumes im Array



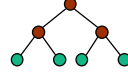

- aus Sicht des Knotens i (Adresse ist **nicht** als Referenz im Knoten gespeichert)
 - Knoten i : $A[i-1]$
 - Linkschild: $A[2*i-1]$
 - Rechtschild: $A[2*i]$
 - Nummerierung beginnt bei 1
- bzw. mit $j=i-1$
 - Knoten j : $A[j]$
 - Linkschild: $A[2*j+1]$
 - Rechtschild: $A[2*j+2]$
 - Nummerierung beginnt bei 0
- Alternative: $A[0]$ frei lassen, $A[1]$ speichert Knoten Nummer 1



G. Zachmann Informatik 2 - SS 06 Bäume 14

Maximale Anzahl Knoten

Wie groß ist die maximale Anzahl der Knoten eines vollständigen Baumes gegebener Höhe?

Baum	Höhe	Anzahl innere Knoten	Anzahl Blätter
	1	0	1
	2	1	2
	3	3	4
	4	7	8
	h	$2^{h-1}-1$	2^{h-1}
		$\Sigma = 2^h - 1$	

G. Zachmann Informatik 2 - SS 06 Bäume 15

Satz: Ein maximal vollständiger binärer Baum der Höhe h enthält 2^{h-1} Blätter und 2^h-1 Knoten und $2^{h-1}-1$ inneren Knoten.

Beweis:

- Induktionsanfang: $h=1$
Der Baum besteht nur aus der Wurzel, die auch das einzige Blatt ist:
 $2^{1-1} = 2^0 = 1$ Blatt
 $2^1 - 1 = 2 - 1 = 1$ Knoten
- Induktionsschritt: $h \rightarrow h' = h + 1$
 Höhe h : 2^{h-1} Blätter, 2^h-1 Knoten
 Höhe $h' = h + 1$: $2 \cdot 2^{h-1} = 2^h = 2^{h'-1}$ Blätter \rightarrow Beh.
 $2^{h'-1}$ innere Knoten + 2^h Blätter = $2^{h'+1}-1 = 2^{h'-1}$

G. Zachmann Informatik 2 - SS 06 Bäume 16

Implementierung in Python

- Nicht mehr 1 Nachfolger, sondern 2, einen **linken** und einen **rechten**
- Knoten hat (mind.) 3 Instanzvar.:
 - Eine Referenz zu **item**
 - Eine Referenz zu **left tree**
 - Eine Referenz zu **right tree**

```

class Tree:
    def __init__( self, item,
                  left = None,
                  right = None):
        self.item = item
        self.left = left
        self.right = right
  
```

G. Zachmann Informatik 2 - SS 06 Bäume 17

Binary Tree Anwendung: Parse-Tree von Ausdrücken

- Abstrakte Repräsentation der Ausdrücke
- Anwendung: Compiler, Computerlinguistik

```

a = Tree(10)
b = Tree(12)
c = Tree('*',a,b)
d = Tree(7)
e = Tree('+',c,d)
  
```

G. Zachmann Informatik 2 - SS 06 Bäume 18

Parse-Tree-Auswertung: Implementierung in Python

- Auswertung eines Parse-Tree:
 - Wenn String ein Integer ist, gebe es aus
 - Sonst, werte rekursiv beide Unterbäume aus und gebe die Summe oder das Produkt aus

$((10 * 12) + (7)) = 127$

```

class ParseTree:
    def eval(self):
        if self.item == "+":
            return self.left.eval() + self.right.eval()
        elif self.item == "*":
            return self.left.eval() * self.right.eval()
        else:
            return self.item
  
```

G. Zachmann Informatik 2 - SS 06 Bäume 19

Preorder Traversierung

- Wie schreiben wir die Information?
 - schreibe den Operator / die Zahl
 - schreibe rekursiv den linken Unterbaum
 - schreibe rekursiv den rechten Unterbaum
- Keine Klammern!

$((4 + 5) + (6 * 7)) * 8$

```

def toString(self):
    if self.item == "+" || self.item == "*":
        return str(self.item) + " " +
            self.left.toString() + " " +
            self.right.toString()
    else:
        return str(self.item)
  
```

Preorder Traversierung: * + + 4 5 * 6 7 8

G. Zachmann Informatik 2 - SS 06 Bäume 20

Konstruktion eines Parse-Tree

- Wie lesen und konstruieren wir den Baum?
 - Lese den String von Standard-Input
 - Falls ein + oder ein * Operator gelesen wird, konstruiere rekursiv den linken und rechten Unterbaum

```

class ParseTree:
    def __init__( self, terminals ):
        # remove next (front) terminal
        self.item = terminals.pop()
        if self.item == "+" or self.item == "*" :
            self.left = ParseTree( terminals )
            self.right = ParseTree( terminals )

a = string.split( sys.stdin.read() )
parsetree = ParseTree( a )

```

```

$ ./parsetree
* + + 4 5 + 6 7 8
108

```

G. Zachmann Informatik 2 - SS 06 Bäume 21

- Beachte: Vorrangregeln (Präzedenzregeln) und Klammern
 - sind in der Baumdarstellung überflüssig
 - nur notwendig in **Infix-Notation**
 - Sog. "**polnische Notation**", erfunden 1920 von Jan Lukasiewicz

HP9100A (1968)
erster Desktop-Computer

```

4.5
7.071 067 811 -01
1.414 213 562 00

```

G. Zachmann Informatik 2 - SS 06 Bäume 22