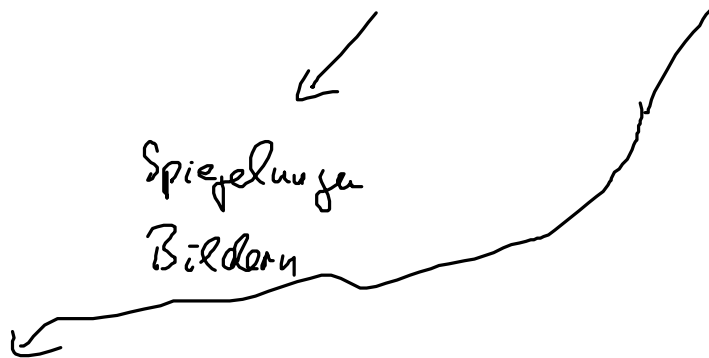


Kapitel 8 Rekursion

rekursiv $\hat{=}$ etwas ist durch sich selber definiert oder enthält sich selbst als Teil

Kommt vor im täglichen Leben, Mathematik, Prog. Sprachen



1. Natürliche Zahlen \mathbb{N}

- $0 \in \mathbb{N}$
- Ist $n \in \mathbb{N}$, so hat n einen ^{eindeutigen} Nachfolger, bezeichnet mit $n+1$ und $n+1 \in \mathbb{N}$
- Die Menge \mathbb{N} ist die kleinste Menge mit den obigen Eigenschaften

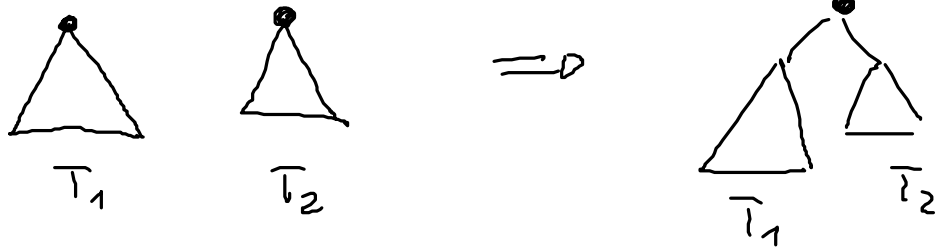
2. Binäre Bäume

Original Def | gerichtete Bäume (im Sinne des Def bei kürzesten Wegen)
bei denen jeder Knoten Anfangspunkt von höchstens 2 Kanten ist

Binäre Bäume können rekursiv wie folgt def werden:

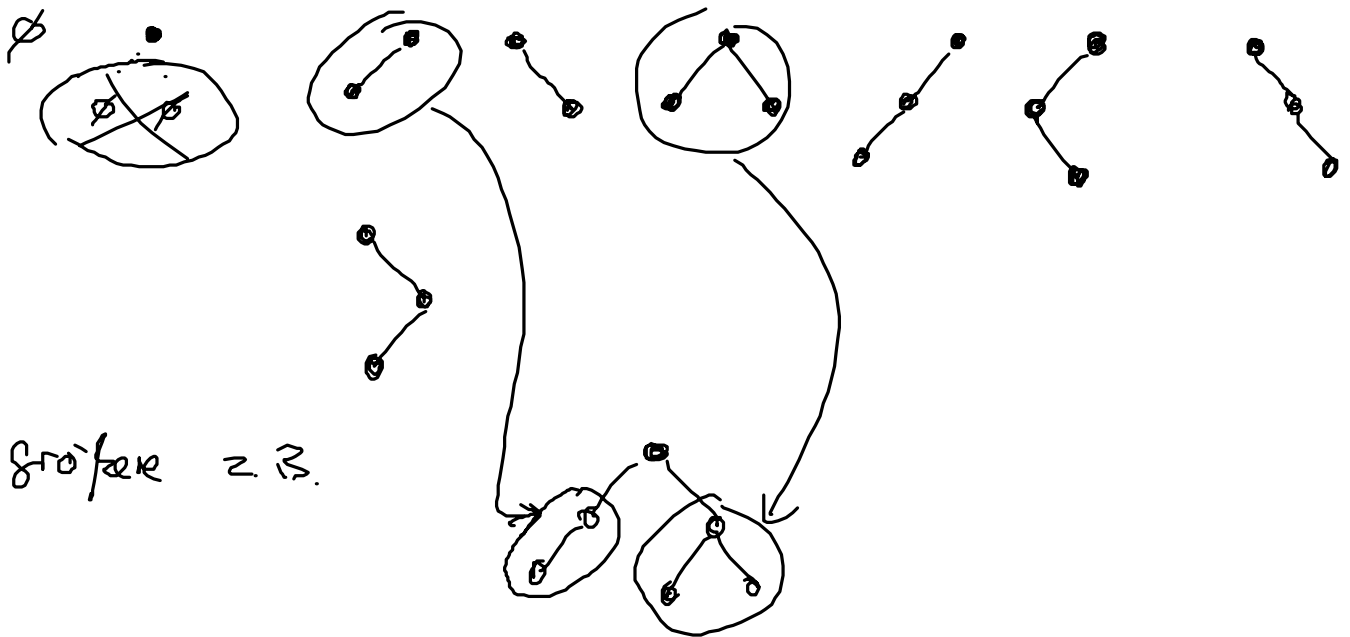
rekursive
Def

- der leere Baum ist ein binärer Baum
 \uparrow
 $G = (V, E)$ mit $V = \emptyset, E = \emptyset$
- Sind T_1, T_2 binäre Bäume, so auch der Graph T bestehend aus einer neuen Wurzel und den Teilbäumen T_1 und T_2



- Dies sind alle binären Bäume

Bsp: alle binären Bäume mit ≤ 3 Knoten



größer z.B.

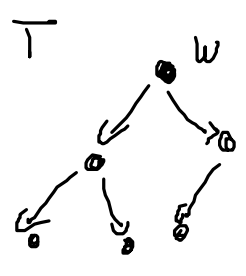
SATZ: beide Definitionen sind äquivalent



Beweis " \Rightarrow "

Sei T ein Baum im Sinne des Orig. Def.

- d.h. 1. T hat eine Wurzel w
- 2. In jedem Knoten $\neq w$ geht genau eine Kante rein
- 3. Jeder Knoten $v \neq w$ ist einem gerichteten Weg von w aus erreichbar
- 4. aus jedem Knoten gehen höchstens 2 Kanten raus



Zeige: T kann auch im Sinne des rekursiven Def erhalten werden

Induktion nach # Knoten n

$$n = 0 \quad T = \emptyset \quad \Rightarrow \quad T \text{ erfüllt rek. Def}$$

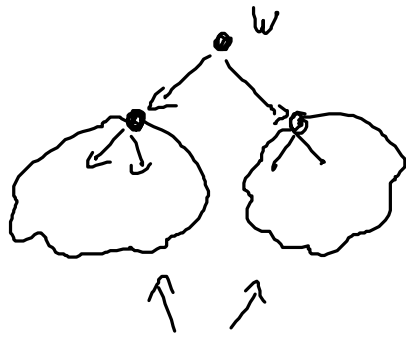
IV. " \Rightarrow " gelte für alle Bäume mit $< n$ Knoten

Schluss auf n

Sei T Baum, der Orig. Def erfüllt mit n Knoten



T hat Wurzel w



disjunkte Knotenmengen
mit $< n$ Knoten

← sind gerichtete Bäume T_1, T_2
im Sinne der Original Def

\Rightarrow (IV) T_1, T_2
erfüllen rekursive Def

$\Rightarrow T$ erfüllt rekursive Def

" \Leftarrow " analog der Induktion \square

3. Fakultät einer natürlichen Zahl $n \geq 0$

$$n! := \begin{cases} 1 & \text{falls } n = 0 \\ n(n-1)! & \text{falls } n > 0 \end{cases}$$

↑ rekursiver Teil des Def.

Anwendungen in Prog. Sprachen

Methoden können sich selbst im Rekursiv aufrufen

"direkte Rekursion"

$f()$ ruft $g()$ auf und $g()$ ruft $f()$ auf

"indirekte Rekursion"

8.1. Beispiele von rekursiven Algorithmen

8.1.1 Berechnung des ggT

$$a > b > 0 \quad \Rightarrow \quad \text{ggT}(a, b) = \text{ggT}(a-b, b)$$

siehe Applet ggT

Bsp: $\text{ggT}(80, 25) = \text{ggT}(55, 25) = \text{ggT}(30, 25)$

$$5 = 80 \bmod 25$$

$$= \text{ggT}(5, 25) = \text{ggT}(20, 5)$$

$$= \text{ggT}(15, 5) = \text{ggT}(10, 5)$$

$$25 \bmod 5 = 0$$

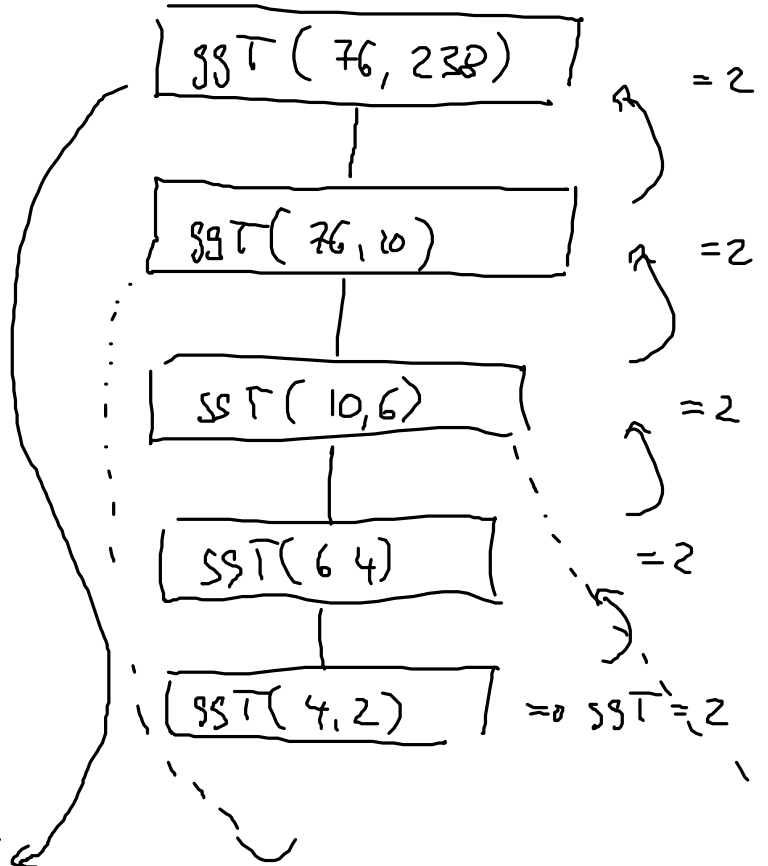
$$= \text{ggT}(5, 5) = 5$$

\Rightarrow

$$\text{ggT}(25, 5) = 5$$

Programm basierend auf modulo Bedingung \rightarrow siehe Skript

Rekursionsbaum = Aufrufbaum (für 76, 238)



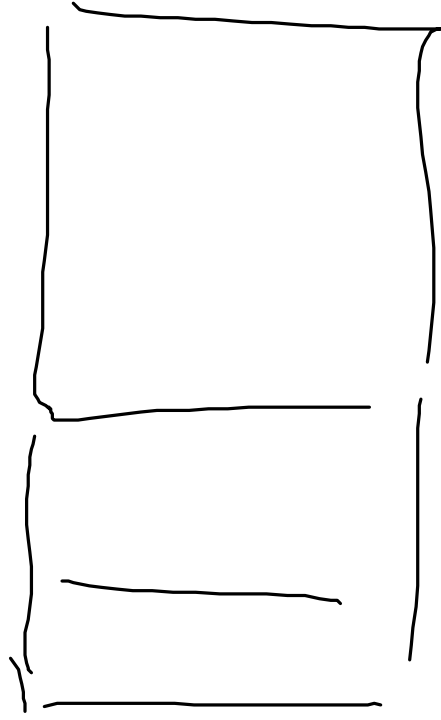
Run Time Stack

(1)

a	76
b	238
min	76
max	238
remainder	10
Rücksprung	
Static pointer HEAP	

a	76
b	10
min	10
max	76
remainder	6
...	
...	

(1)



Bei Rekursion interessiert

Rekursionstiefe $\hat{=}$ Kap für den Speicherplatz
 Anzahl der rekursiven Aufrufe $\hat{=}$ Kap für Laufzeit
 Höhe Aufrufbaum + 1 $\hat{=}$ # Knoten im Aufrufbaum
 = max. Anzahl von
 Aktivierungsrecords
 auf Laufzeitstack

Bei ggT ist Aufrufbaum eine Liste
 \Rightarrow Rekursionstiefe = # rekurs. Aufrufe

SATZ Bei ggT gilt für die Rekursionstiefe $R_{\text{ggT}}(a, b)$

$$R_{\text{ggT}}(a, b) \leq 1 + 2 \cdot \log M \quad M = \max\{a, b\}$$

\nearrow
 sehr schön!

Beweis:

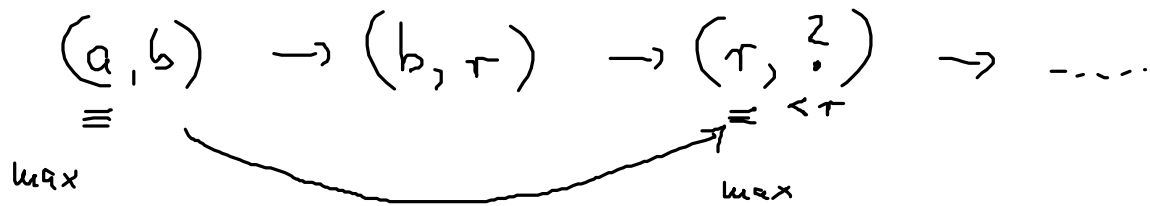
Sei $a > b$

$$\underbrace{a \text{ div } b}_{=: t} \quad \underbrace{a \text{ mod } b}_{=: r}$$

$$\Rightarrow a = t \cdot b + r \geq \underset{t \geq 1}{b} + r > \underset{b > r}{r + r} = 2r$$

$$\Rightarrow \boxed{a > 2r} \Rightarrow r < \frac{a}{2}$$

Betrachten Folge in HSG



in 2 auf einander folgenden rekursiven Aufrufen
hat sich die größere der beiden Zahlen mehr als halbiert

$$\left(r < \frac{a}{2} \right)$$

$$\Rightarrow R_{\text{HSG}}(a, b) \leq 1 + 2 \log M \quad \square$$

8.1.2 Türme von Hanoi



Ausgangsstapel Zielstapel Hilfsstapel

Scheiben auf Zielstapel bringen

eine Scheibe pro "Zug"

wie eine größere auf eine kleinere legen

Gesucht: minimale Folge von Zügen, um alle Scheiben zum Zielstapel zu bewegen

rekursiver Ansatz zur Ermittlung des Züge:

wenn unterste Scheibe auf Zielstapel bewegt wird
müssen die anderen $n-1$ Scheiben auf Hilfsstapel sein

↓

	Start	Hilf
→ Bewege zunächst $n-1$ Scheiben von 1 nach 3		
→ Bewege unterste Scheibe n von 1 nach 2		Ziel
→ Bewege $n-1$ Scheiben von 3 nach 2 (jetzt ist 1 der Hilfsstapel)		

↑

Delegation (rekursiver Aufruf)

Java Application → Programme zw VL / Skript