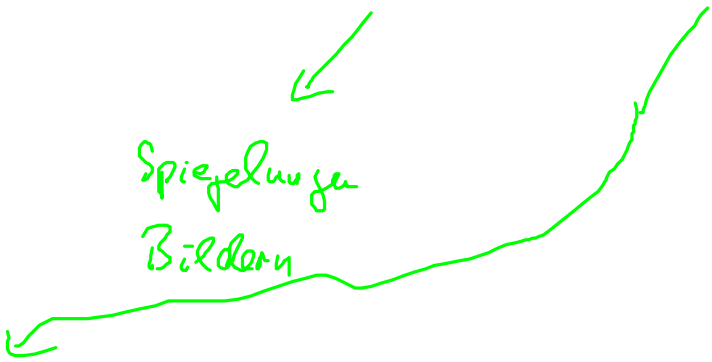
Kapitel 8 Rekursion

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

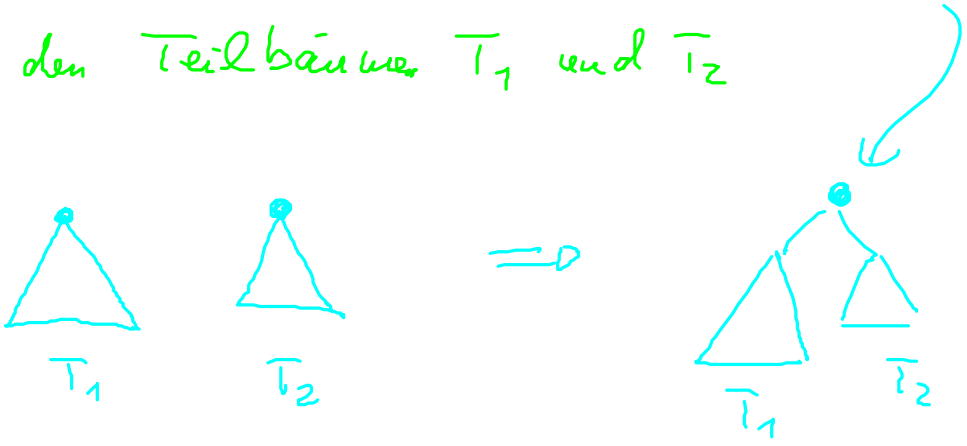
2. Binäre Bäume

Original Def | geordnete Bäume (im Sinne der Def bei Kürzesten Wegen) bei denen jeder Knoten Anfangspunkt von höchstens 2 Kanten ist

Binäre Bäume können rekursiv wie folgt def werden:

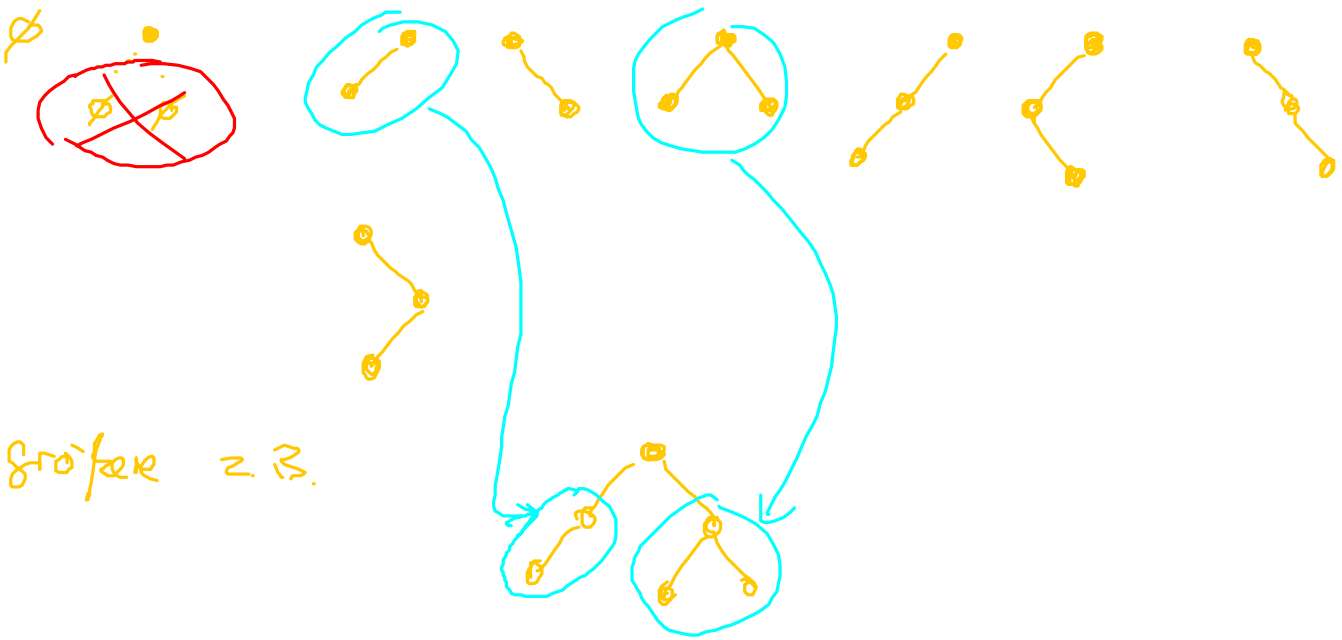
rekursive Def

- der leere Baum ist ein binärer Baum
  ↑ $G = (V, E)$ 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



größere z.B.

SATZ: beide Definitionen sind äquivalent

Original Def $\Leftrightarrow$ rek. Def

Beweis "$\Rightarrow$"

Sei T ein Baum im Sinne der Orig. Def.

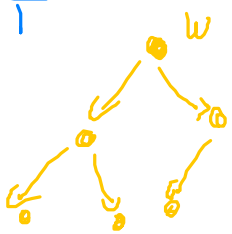
d.h. 1. T hat eine Wurzel w

2. In jeden Knoten $\neq w$ geht genau eine Kante rein

3. Jeder Knoten $v \neq w$ ist auf einem gerichteten Weg von w aus erreichbar

4. aus jedem Knoten gehen höchstens 2 Kanten raus

T



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

Induktion nach # Knoten n

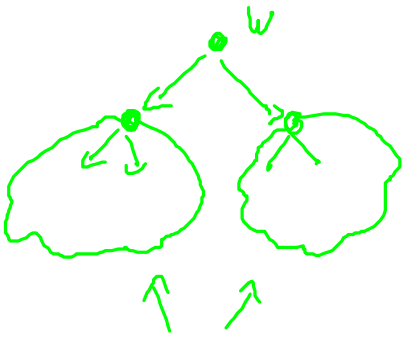
$n = 0$ $T = \emptyset$ $\Rightarrow$ T erfüllt rek. Def

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

$\Leftarrow$ sind geordnete 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 der Def.

Anwendungen in Prog. Sprachen

Methoden können sich selbst im Rumpf 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

$$a > b > 0 \Rightarrow ggT(a,b) = ggT(a-b,b)$$

siehe Applet ggT

Bsp: $ggT(80,25) = ggT(55,25) = ggT(30,25)$

$= ggT(5,25) = ggT(20,5)$

$5 = 80 \bmod 25$

$= ggT(15,5) = ggT(10,5)$

$= ggT(5,5) = 5$

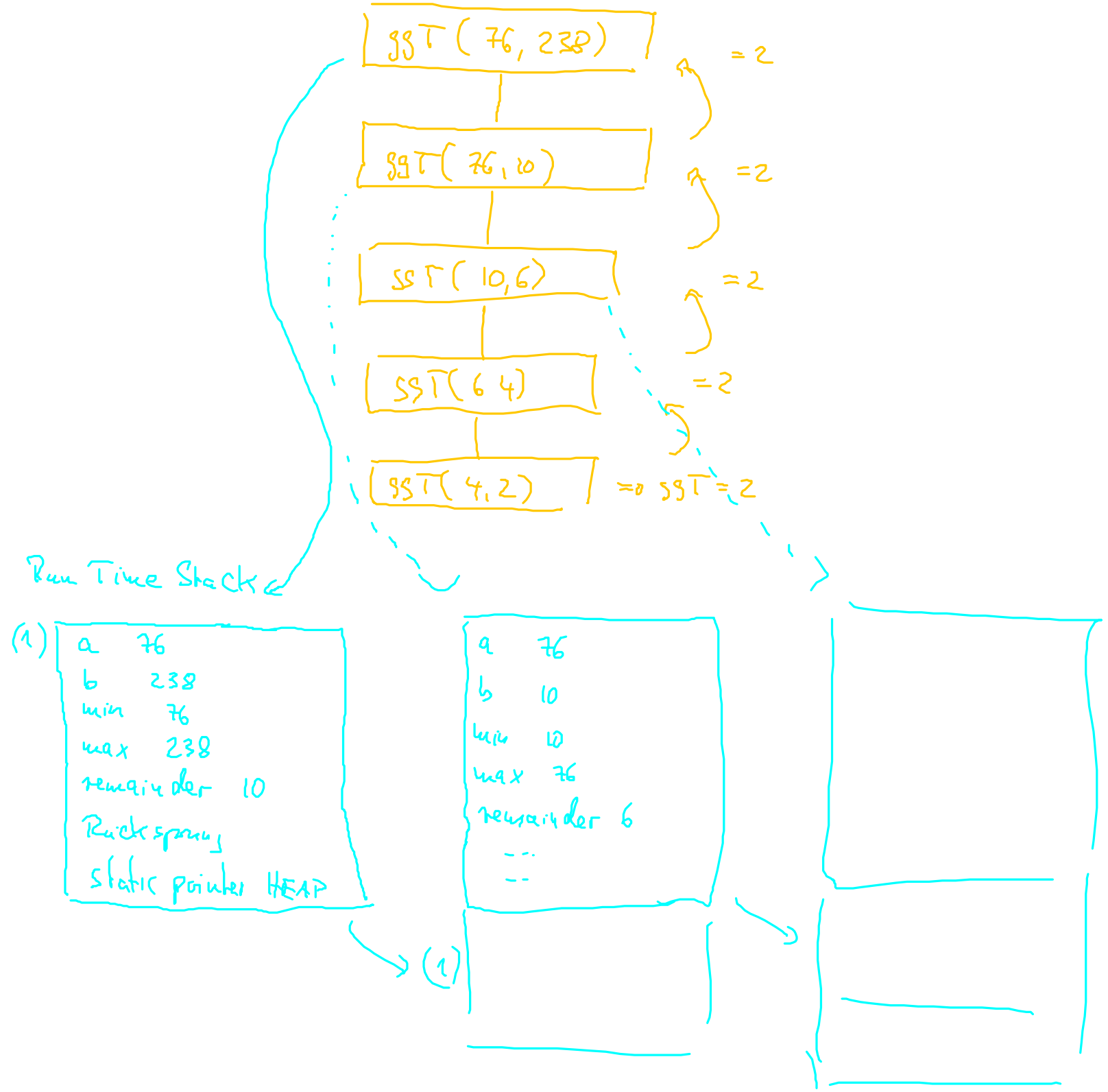
$25 \bmod 5 = 0$

$\Rightarrow$

$ggT(25,5) = 5$

Programm basierend auf modulo Rechnung → siehe Skript

Rekursionsbaum = Aufrufbaum (für 76, 238)



Bei Rekursion ineffizient

Rekursionstiefe $\hat{=}$ Platz für den Speicherplatz

Anzahl der rekursiven Aufrufe $\hat{=}$ Platz für Laufzeit

Höhe Aufrufbaum + 1

Knoten im Aufrufbaum

= max. Anzahl von aktivierten Records auf Laufzeitstack

Bei ggT ist Aufrufbaum eine Liste

$\Rightarrow$ Rekursionstiefe = # rekurs. Aufrufe

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

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

sehr schön!

Beweis:

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

$$\Rightarrow \quad a = t \cdot b + r \geq b + r > r + r = 2r$$

$t \geq 1$ $\qquad b > r$

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

Betrachte Folge in Regel

$$(a,b) \to (b,r) \to (r,?) \to \dots$$

$\equiv$ max

$= <r$ max

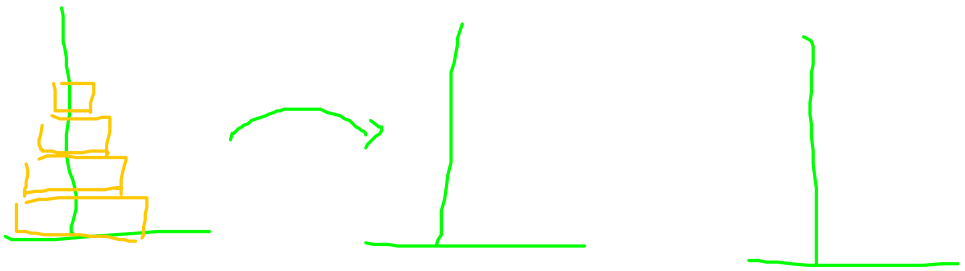
in 2 aufeinanderfolgenden rekursiven Aufrufen

hat sich die größere der beiden Zahlen mehr als halbiert

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

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

8.1.2 Türme von Hanoi



Ausgangsstapel Zielstapel Hilfsstapel

Scheiben auf Zielstapel bringen

eine Scheibe pro "Zug"

nie eine größere auf eine kleinere legen

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

rekursiver Ansatz zur Ermittlung der Züge:

> wenn unterste Scheibe auf Zielstapel bewegt werd
> 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
→ Bewegen $n-1$ Scheiben von	3	nach 2 (jetzt ist 1 der Hilfsstapel)

↑

Delegation (rekursiver Aufruf)

Java Applikation → Programme zu VL / Skript