Asymptotische Notation (1)

Definition 3.1 Seien $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ Funktionen.

(a) **(Groß-Oh Notation)** $f = O(g)$ $:\Leftrightarrow$ Es gibt eine positive Konstante $c > 0$ und eine natürliche Zahl $n_0 \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt

$$f(n) \leq c \cdot g(n).$$

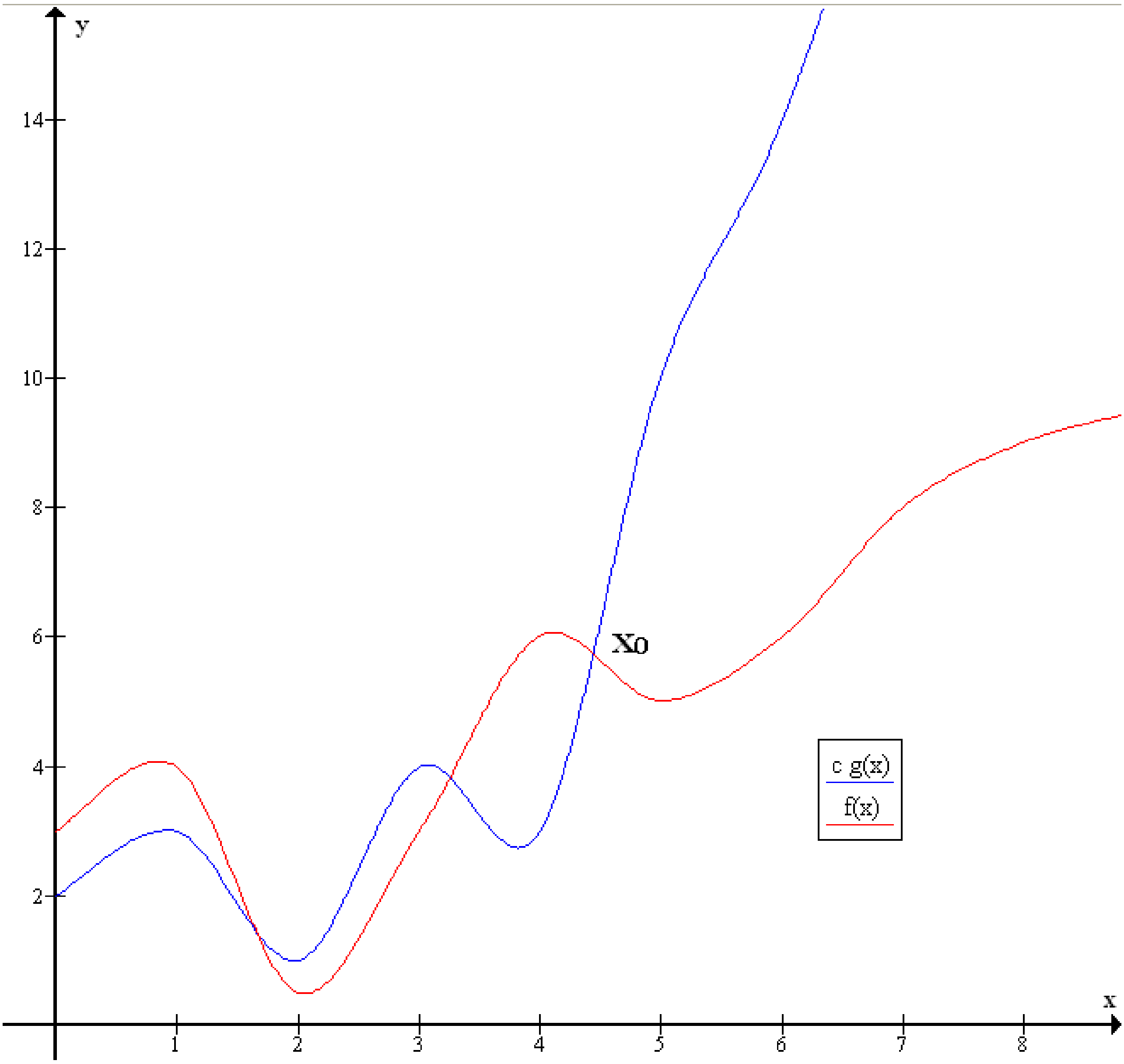
(b) **(Groß-Omega Notation)** $f = \Omega(g) :\Leftrightarrow g = O(f)$.

(c) **(Theta-Notation)** $f = \Theta(g) :\Leftrightarrow f = O(g)$ und $g = O(f)$.

(d) **(Klein-Oh Notation)** $f = o(g) :\Leftrightarrow \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

(e) **(Klein-Omega Notation)** $f = \omega(g) :\Leftrightarrow \lim\limits_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

Veranschaulichung von f = O(g)



Asymptotische Notation (2)

Lemma 3.1 *Der Grenzwert der Folge* $\frac{f(n)}{g(n)}$ *möge existieren und es sei* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$.

(a) Wenn $c = 0$, *dann ist* $f = o(g)$.

(b) Wenn $0 < c < \infty$, *dann ist* $f = \Theta(g)$.

(c) Wenn $c = \infty$, *dann ist* $f = \omega(g)$.

(d) Wenn $0 \leq c < \infty$, *dann ist* $f = O(g)$.

(e) Wenn $0 < c \leq \infty$, *dann ist* $f = \Omega(g)$.

Dieses Kriterium versagt, falls der Grenzwert f(n)/g(n) nicht existiert! Insbesondere gilt die Rückrichtung der Implikation im Allgemeinen nicht!

Beispiel 3.2 Wir definieren $f(n) = \begin{cases} 1 & n \text{ gerade} \\ 0 & \text{sonst.} \end{cases}$ Es sei $g(n) = 1$ für alle $n \in \mathbb{N}$. Dann existiert der Grenzwert von $\frac{f(n)}{g(n)}$ nicht, da die Werte 0 und 1 unendlich oft auftauchen. Es ist aber

$$f = O(g)$$

Die Wachstumshierachie

Beispielaufgabe:
Die nebenstehenden Funktionen „aufsteigend sortieren“

$$\begin{aligned} f_1(n) &= \frac{3^n}{9} \\ f_2(n) &= 9n \cdot \log_3 n \\ f_3(n) &= 3^{\log_9 n} \\ f_4(n) &= 9^{\log_3 n} \\ f_5(n) &= \frac{9n}{3} \\ f_6(n) &= \frac{(n+1)!}{n+1} \end{aligned}$$

Hierachie:

- $f_1(n) = 1$, dann ist $f_1 = o(\log_2 \log_2 n)$,
  haben wir schon eingesehen.
- $\log_2 \log_2 n = o(\log_2 n)$,
  haben wir schon eingesehen.
- $\log_2 n = \Theta(\log_a n)$ für jedes $a > 1$,
  haben wir schon eingesehen.
- $\log_2 n = o(n^b)$ für jedes $b > 0$,
  haben wir schon eingesehen.
- $n^b = o(n)$ und $n = o(n \cdot \log_a n)$ für jedes b mit $0 < b < 1$ und jedes $a > 1$,
  $\lim_{n\to\infty} \frac{n^b}{n} = \lim_{n\to\infty} \frac{1}{n^{1-b}} = 0$ und $\lim_{n\to\infty} \frac{n}{n \cdot \log_a n} = \lim_{n\to\infty} \frac{1}{\log_a n} = 0$.
- $n \cdot \log_a n = o(n^k)$ für jede reelle Zahl $k > 1$ und jedes $a > 1$.
- $n^k = o(a^n)$ für jedes $a > 1$,
  $n^k = a^{k \cdot \log_a n}$ und $\lim_{n\to\infty} \frac{n^k}{a^n} = \lim_{n\to\infty} a^{k \cdot \log_a n - n} = 0$.
- und $a^n = o(n!)$ für jedes $a > 1$.

Baum-Traversierungen

Definition 4.1 Sei T ein geordneter Baum mit Wurzel r und Teilbäumen $T_1, \ldots, T_m$.

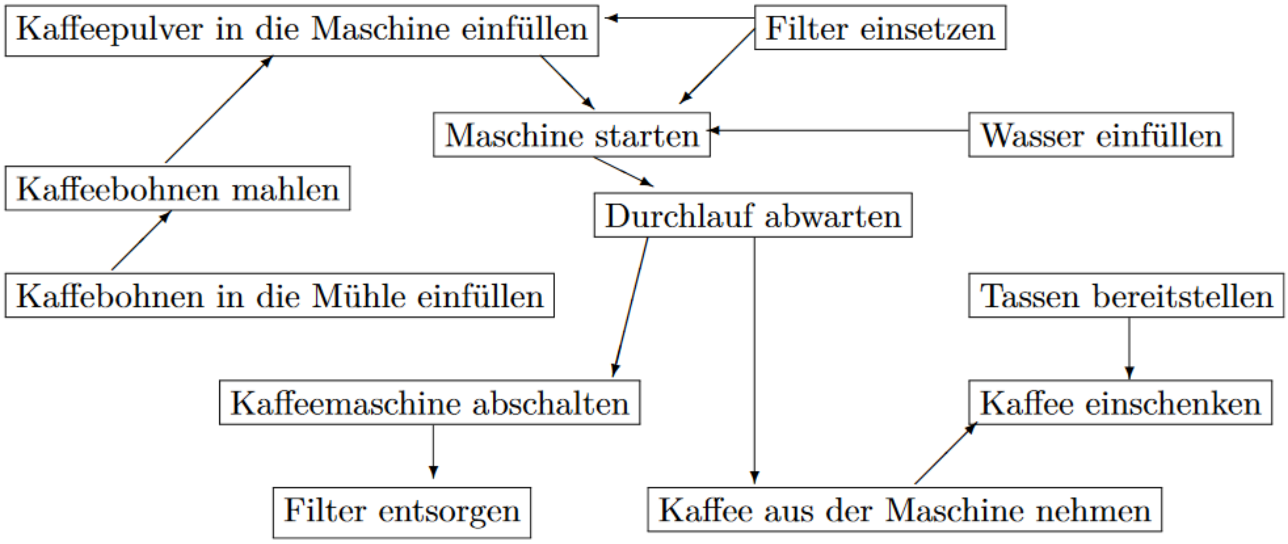
(a) Wenn T in **Postorder** durchlaufen wird, dann werden (rekursiv) die Teilbäume $T_1, \ldots, T_m$ nacheinander durchlaufen und danach wird die Wurzel r besucht.

(b) Wenn T in **Präorder** durchlaufen wird, dann wird zuerst r besucht und dann werden die Teilbäume $T_1, \ldots, T_m$ (rekursiv) durchlaufen.

(c) Wenn T in **Inorder** durchlaufen wird, wird zuerst T_1 (rekursiv) durchlaufen, sodann wird die Wurzel r besucht und letztlich werden die Teilbäume $T_2, \ldots, T_m$ (rekursiv) durchlaufen.

Topologisches Sortieren

Beispiel: Kaffeekochen



Eine gültige Ausführungsreihenfolge der Aufgaben entspricht einer topologischen Sortierung.

Idee: Aufgaben ohne eingehende Kanten können ausgeführt werden. Da die Aufgabe dann erledigt ist, werden ihre ausgehenden Kanten gelöscht.

Algorithmus mit bestmöglicher Laufzeit: O(n + p), wobei n = #Aufgaben, p = #Prioritäten
Bzw. in der Graphdarstellung ist n = #Knoten, p = #Kanten

> Stelle die Prioritäten durch eine Adjazenzliste mit dem Kopf-Array **Priorität** dar.
> Benutze ein Array **In-Grad** mit In-Grad$[v] = k$, falls v Endpunkt von k Kanten ist.

(1) Initialisiere die Adjazenzliste Priorität durch Einlesen aller Prioritäten. (Zeit = $O(n+p)$).

(2) Initialisiere das Array In-Grad. (Zeit = $O(n+p)$).

(3) Alle Knoten v mit In-Grad$[v] = 0$ werden in eine **Schlange** eingefügt. (Zeit = $O(n)$).

(4) Setze Zähler $= 0$; Wiederhole solange, bis *Schlange* leer ist:

  (a) Entferne einen Knoten i aus *Schlange*.
  (b) Setze **Reihenfolge** [Zähler++] $= i$.
  (c) Durchlaufe die Liste Priorität $[i]$ und reduziere In-Grad für jeden Nachfolger j von i um 1. Wenn jetzt In-Grad$[j] = 0$, dann füge j in *Schlange*: Aufgabe a_j ist jetzt ausführbar.

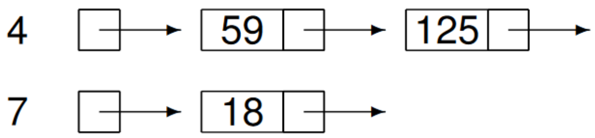
Hashing (1)

Hashing mit Verkettung

Für jede Zelle i wird eine anfänglich leere Liste angelegt.

- Jede Liste wird sortiert gehalten.
- Für **lookup**(**x**): Durchlaufe die Liste von $h(x)$.
- Für **insert**(**x**) und **remove**(**x**): Führe die insert- und remove-Operation für einfach-verkettete Listen aus.

Beispiel: Wähle $\mathbf{h}(\mathbf{x}) = (\mathbf{x} \textbf{ mod } \mathbf{11})$ als Hashfunktion. Die Operationen **insert(59)**, **insert(18)** und **insert**(125) führen auf die Tabelle



lookup (26) benötigt nur einen Suchschritt: Schlüssel 59 wird gefunden und es wird geschlossen, dass 26 nicht präsent ist.

Gute Wahl einer Hashfunktion: h(x) = x mod m (m Primzahl)

Hashing mit offener Adressierung

Wir arbeiten mit einer Folge

$$h_0, \ldots, h_{m-1} : U \to \{0, \ldots, m-1\}$$

von Hashfunktionen. Setze $i = 0$.

(1) Wenn die Zelle $h_i(x)$ frei ist, dann füge x in Zelle $h_i(x)$ ein.

(2) Ansonsten setze $i = i + 1$ und gehe zu Schritt (1).

Lineares Austesten

In der Methode des **linearen Austestens** wird die Folge

$$\mathbf{h_i}(\mathbf{x}) = (\mathbf{x} + \mathbf{i}) \mod \mathbf{m}$$

benutzt: Also wird die jeweils nächste Zelle untersucht.

Hashing (2)

Doppeltes Hashing

Wir benutzen zwei Hashfunktionen f und g und verwenden die Folge

$$\mathbf{h_i(x) = (f(x) + i \cdot g(x)) \mod m}.$$

- Die Klumpenbildung wird vermieden.
- Man erhält gute Ergebnisse bereits für

$$\mathbf{f(x) = x \bmod m} \quad \text{und} \quad \mathbf{g(x) = m^* - (x \mod m^*)}.$$

  - Wähle m als Primzahl und fordere $m^* < m$.