Karlsruher Institut für Technologie

Karlsruher Institut für Technologie

Algorithmen I
Tutorium 32

Eine Lehrveranstaltung im SS 2017 (mit Folien von Christopher Hommel)

Daniel Jungkind (ufesa@kit.edu) | 12. Mai 2017

INSTITUT FÜR THEORETISCHE INFORMATIK

Zum letzten Blatt (#1)

- **Aussage** + **Begründung** irgendwie deutlich machen
- **Form**: Definieren, was ihr benutzt (Wo kommen f und g her??)
- Induktion: Hübsche Rechnung. **Und was heißt das jetzt?** (Zusammenhang Rechnung $\Leftrightarrow$ Code-Geschehen nicht vergessen!)
- Induktion mittels $a \rightsquigarrow a + 1$ gefährlich: Wo wird a erhöht?
  (c genauso: Wird in der *Mitte* erhöht!)
  $\Rightarrow$ Besser: Schleifendurchläufe **nummerieren** (i) und Variablen auch (a_i, b_i, ...)!

Zum letzten Blatt (#1)

Aufgabe 2 c)

function f$(n, m : \mathbb{N}) : (\mathbb{N}_0, \mathbb{N}_0)$

$a = 0 : \mathbb{N}_0$
$b = m : \mathbb{N}_0$
$c = 1 : \mathbb{N}_0$
while $m - c \cdot n \geqslant 0$ **do**
  invariant $m = a \cdot n + b$
  $a := c$
  $c := c + 1$
  $b := m - a \cdot n$
return (a, b)

Lösung Aufgabe 2 c)

Bezeichne i die Nummer des aktuellen Schleifendurchlaufs und a_i, b_i, c_i den Wert von a, b, c zu Beginn von Schleifendurchlauf i.

IA. $(i = 1)$: $\quad a_1 \cdot n + b_1 = 0 \cdot n + m = m. \quad \checkmark$

IV.: Die Invariante galt zu Beginn von Schleifendurchlauf i.

IS. $(i \rightsquigarrow i + 1)$: Es gilt zu Beginn von Schleifendurchlauf $i + 1$:

$$a_{i+1} = c_i,$$

$$b_{i+1} = m - a_{i+1} \cdot n.$$

$$\implies a_{i+1} \cdot n + b_{i+1} = c_i \cdot n + (m - c_i \cdot n) = m. \quad \square$$

Laufzeitabschätzung

Laufzeit?

function doing($n : \mathbb{N}$) : $\mathbb{N}$

```
function doing(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := 1 to n do
                k++
    return k + ℓ
```

Laufzeitabschätzung

- Erster Gedanke:
  Äußere Schleife: n Durchläufe,
  Innere Schleife: n Durchläufe
  $\cancel{\Rightarrow \Theta(n \cdot n) = \Theta(n^2)}$
- **Aber**: Innere Schleife wird nur **max. 4x** erreicht
  (nämlich für $i \in \{n-3, n-2, n-1, n\}$)
  $\Rightarrow \Theta(n + 4n) = \Theta(n)$

Laufzeitabschätzung

Laufzeit?

```
function boing(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        for j := i to n do
            k++
    return k + ℓ
```

Laufzeitabschätzung

- Erster Gedanke: Äußere Schleife macht n-mal „irgendwas“
  $\Rightarrow n \cdot (???)$
  (Klammer? Wie schreiben wir das auf? Nicht n-mal dasselbe, sondern **von i abhängig**!)
- Rettung: Anzahl innere Schleifendurchläufe **einzeln** für jedes $i = 1, \ldots, n$ **aufsummieren**!
- Für ein festes i wird innere Schleife $(n - i + 1)$-mal durchlaufen

$\Rightarrow$ Gesamtanzahl der inneren Schleifendurchläufe:

$$\sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = n^2 - \sum_{i=1}^{n} i + n$$

$$= n^2 - \frac{n \cdot (n+1)}{2} + n = n^2 - \frac{n^2}{2} + n - \frac{n}{2} = \frac{n^2 + n}{2} \in \Theta(n^2).$$

Laufzeitabschätzung

Karlsruher Institut für Technologie

Laufzeit?

```
function going(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := i to n do
                k++
    return k + ℓ
```

Laufzeitabschätzung

Die innere Schleife wird weiterhin (siehe Funktion *doing*) **nur max. vier Mal** erreicht (für $i \in \{n-3, n-2, n-1, n\}$).

$\Rightarrow$ Die innere Schleife wird erst vier-, dann drei-, dann zwei- und dann einmal durchlaufen

$\Rightarrow \Theta(n + 4 + 3 + 2 + 1) = \Theta(n)$

Laufzeitabschätzung

Hinweise für Aufgaben

- „Obere Schranke“ gefordert
  $\Rightarrow$ $O(f(n))$ (potenziell „zu große“ Schranke) ausreichend
- „**Scharfe** asymptotische Schranke“ gefordert
  $\Rightarrow$ $\Theta(f(n))$ benötigt
- Laufzeit eines Algorithmus soll angegeben bzw. bestimmt werden
  $\Rightarrow$ Offiziell $\Theta(f(n))$ erwünscht
  (in VL oder Musterlösungen aber oft auch $O(f(n))$)

Das Master-Theorem (einfache Form)

a, b, c, d positive Konstanten und für $n \in \mathbb{N}$ sei

$$T(n) = \begin{cases} a, & \text{für } n = 1 \\ d \cdot T(\lceil \frac{n}{b} \rceil) + cn, & \text{für } n > 1 \end{cases}.$$

Dann gilt:

$$T(n) \in \begin{cases} \Theta(n), & d < b \\ \Theta(n \log n), & d = b \\ \Theta(n^{\log_b d}), & d > b \end{cases}.$$

Beispiele Master-Theorem

$n = 8^k, k \in \mathbb{N}_0$:

$$A(n) = \begin{cases} 42, & \text{für } n = 1 \\ 8 \cdot A(\frac{n}{8}) + 5n, & \text{für } n > 1 \end{cases}$$

$$\implies A(n) \in \Theta(n \log n)$$

Beispiele Master-Theorem

$n = 4^k, k \in \mathbb{N}_0$:

$$B(n) = \begin{cases} 1337, & \text{für } n = 1 \\ 2 \cdot B(\frac{n}{4}) + 100000n, & \text{für } n > 1 \end{cases}$$

$$\implies B(n) \in \Theta(n)$$

Beispiele Master-Theorem

$n = 2^k, k \in \mathbb{N}_0$:

$$C(n) = \begin{cases} 69, & \text{für } n = 1 \\ 4 \cdot C(\frac{n}{2}) + 3n, & \text{für } n > 1 \end{cases}$$

$$\implies C(n) \in \Theta(n^{\log_2 4}) = \Theta(n^2)$$

Beispiele Master-Theorem

$n = 13^k, k \in \mathbb{N}_0$:

$$D(n) = \begin{cases} 8, & \text{für } n = 1 \\ 11 \cdot D(\frac{n}{13}) + 6n, & \text{für } n > 1 \end{cases}$$

$$\implies D(n) \in \Theta(n)$$

Beispiele Master-Theorem

$n = 3^k, k \in \mathbb{N}_0$:

$$E(n) = \begin{cases} 255, & \text{für } n = 1 \\ 27 \cdot E(\frac{n}{3}) + 3n, & \text{für } n > 1 \end{cases}$$

$$\implies E(n) \in \Theta(n^{\log_3 27}) = \Theta(n^3)$$

Beispiele Master-Theorem

$n = 35767^k, k \in \mathbb{N}_0$:

$$F(n) = \begin{cases} 21, & \text{für } n = 1 \\ 35767 \cdot F(\frac{n}{35767}) + 5n, & \text{für } n > 1 \end{cases}$$

$$\implies F(n) \in \Theta(n \log n)$$

Aufgaben Master-Theorem

Aufgabe 1: Master-Theorem

Die Laufzeit eines Algorithmus A wird beschrieben durch

$$U(n) = \begin{cases} 1, & \text{für } n = 1 \\ 7 \cdot U(\lceil \frac{n}{2} \rceil) + n, & \text{für } n > 1 \end{cases}$$

Ein weiterer Algorithmus B hat die Laufzeit

$$V(n) = \begin{cases} 1, & \text{für } n = 1 \\ a \cdot V(\lceil \frac{n}{4} \rceil) + 5n, & \text{für } n > 1 \end{cases}$$

Was ist der größte Wert $a \in \mathbb{N}$, so dass B asymptotisch schneller als A ist?

Aufgaben Master-Theorem

Lösung zu Aufgabe 1

- Master-Theorem: Algorithmus A hat Laufzeit $\Theta(n^{\log_2 7})$, wächst also stärker als n^2
- Fall $a \leqslant 4$ also uninteressant $\Rightarrow$ $a > 4$, d.h. Algorithmus B läuft in $\Theta(n^{\log_4 a})$
- Also:

$$\log_4 a < \log_2 7 \;\Leftrightarrow\; \frac{\log a}{\log 4} < \frac{\log 7}{\log 2} \;\Leftrightarrow\; \log a < \log 7 \cdot \underbrace{\log 4}_{=2}$$

$$\Leftrightarrow a < 2^{(\log 7)\cdot 2} = \left(2^{\log 7}\right)^2 = 7^2 = 49 \implies a = 48.$$

(mittels $\log_x y = \frac{\log y}{\log x}$ und $\log z = \log_2 z$)

Aufgaben Master-Theorem

Aufgabe 2: Master-Theorem

Gegeben sei folgende Rekurrenz für $n = 4^k, k \in \mathbb{N}_0$

$$T(n) = \begin{cases} 2, & \text{für } n = 1 \\ 2 \cdot T(\frac{n}{4}), & \text{für } n > 1 \end{cases}$$

Findet eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ und Konstanten c_1, c_2, so dass $c_1 \cdot f(n) \leqslant T(n) \leqslant c_2 \cdot f(n)$ und beweist euren Fund.

Aufgaben Master-Theorem

Lösung zu Aufgabe 2

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$
erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n} \right)$$

IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{Def.}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{IV}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{IV}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}. \quad \square$$