RUB

Vorlesung Datenstrukturen

Weitere Grundlagen

Maike Buchin

20.4.2017

Wiederholung

RUB

- wir interessieren uns für effizienten Algorithmen und Datenstrukturen
- Laufzeiten messen wir asymptotisch in der Oh-Notation
- dabei zählen wir die Anzahl elementarer Rechenschritte in der Größe der Eingabe im schlimmsten Fall

2.6 Algorithmenanalyse

2.6.2 Rekurrenzen

- Laufzeiten rekursiver Algorithmen lassen sich als Rekurrenz formulieren
- Z.B. Binäre Suche: $T(n) = T(\lfloor n/2 \rfloor) + O(1)$
- Zum Lösen dieser gibt es verschiedene Methoden
- Häufig lassen sich diese mit dem Mastertheorem lösen

Mastertheorem [einfache Form]

RUB

Satz 2.5 Für positive Konstanten a, b, c, d und $n = b^k$ für eine natürliche Zahl k, sei die folgende Rekurrenz gegeben:

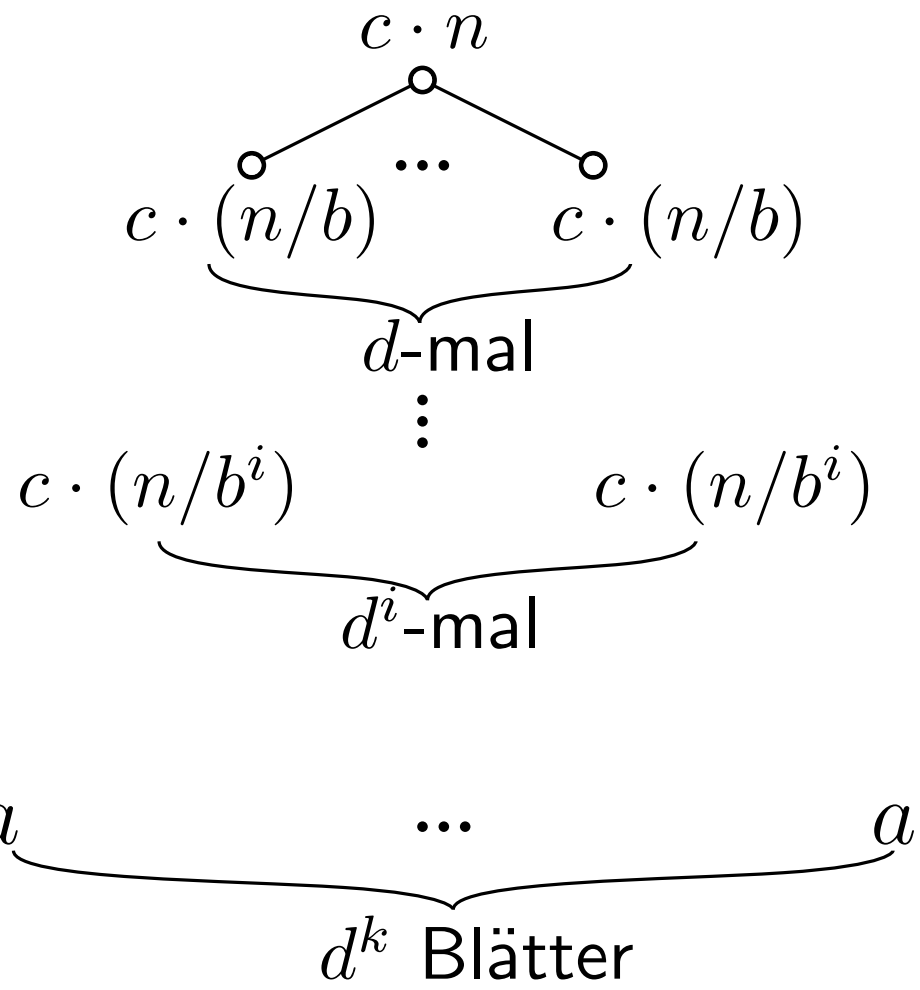
$$r(n) = \begin{cases} a, & \text{für } n = 1 \\ d \cdot r(n/b) + cn, & \text{für } n > 1 \end{cases}$$

Dann gilt:

$$r(n) = \begin{cases} \Theta(n) & \text{für } d < b \\ \Theta(n \log n) & \text{für } d = b \\ \Theta(n^{\log_b d}) & \text{für } d > b \end{cases}$$

Mastertheorem [einfache Form]

Beweis: Rekursionsbaum



Kosten (= Anzahl · Größe)

$c \cdot n$

$d \cdot c \cdot (n/b)$

$\vdots$

$d^i \cdot c \cdot (n/b^i)$

$\vdots$

$d^{k-1} \cdot c \cdot (n/b^{k-1})$

$a \cdot d^{\log_b n}$

$$= \sum_{i=0}^{k-1} d^i c(n/b^i) + an^{\log_b d}$$
$$= cn \sum_{i=0}^{k-1} (d/b)^i + ad^k$$

Tiefe: $k = \log_b n$

Mastertheorem [einfache Form]

RUB

Beweis:

2. Fall $d = b$: $cn \log_b n = \Theta(n)$ für Teile-und-Herrsche und an für unterste Ebene

1. Fall $d < b$: $ad^k < ab^k = O(n)$ für unterste Ebene und $cn \sum_{i=0}^{k-1} (d/b)^i = \Theta(n)$ für Teile-und-Herrsche

3. Fall $d > b$: $\Theta(n^{\log_b d})$ für beides

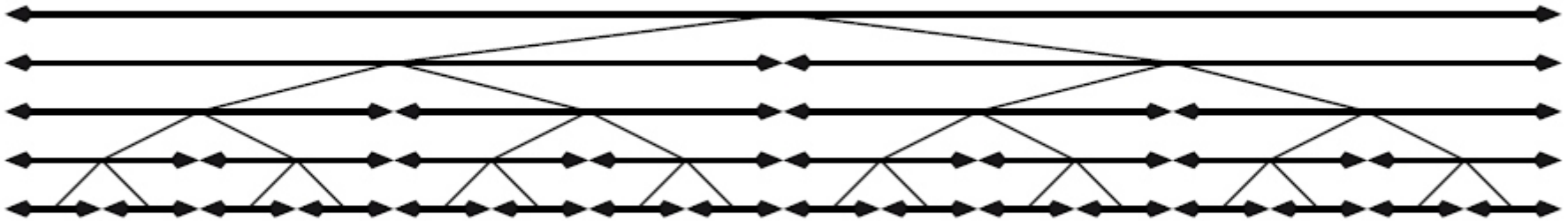
Mastertheorem [einfache Form]

RUB

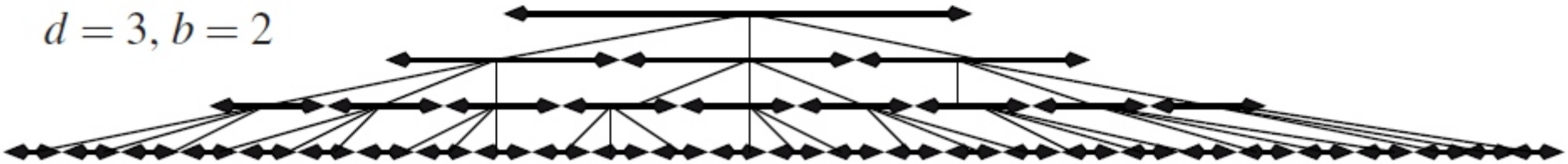
Beispiele:

1. $d = 2, b = 4$

2. $d = b = 2$



3. $d = 3, b = 2$



1. $T(n) = 2T(n/4) + cn = \Theta(n)$
2. $T(n) = 2T(n/2) + cn = \Theta(n \log n)$
3. $T(n) = 4T(n/2) + cn = \Theta(n^2)$

Mastertheorem [allgemeine Form]

RUB

Satz [Thm 4.1 in CLRS] Seien $a \geq 1$ und $b > 1$ Konstanten. Sei $f(n)$ eine Funktion und $T(n)$ durch die Rekurrenz

$$T(n) = aT(n/b) + f(n)$$ gegeben.

Dann besitzt $T(n)$ folgende asymptotische Schranken:

1. Falls $f(n) = O(n^{\log_b a - \epsilon})$ für eine Konstante $\epsilon > 0$, dann gilt $T(n) = \Theta(n^{\log_b a})$.
2. Falls $f(n) = \Theta(n^{\log_b a})$, dann gilt $T(n) = \Theta(n^{\log_b a} \log n)$.
3. Falls $f(n) = \Omega(n^{\log_b a + \epsilon})$ für eine Konstante $\epsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und hinreichend große n, dann gilt $T(n) = \Theta(f(n))$.

Mastertheorem [allgemeine Form]

RUB

Beispiele:

1. $T(n) = 9T(n/3) + n$
   Da $f(n) = n = O(n^{\log_3 9}) = O(n^2)$ folgt mit dem ersten Fall mit $\epsilon = 1$, dass $T(n) = \Theta(n^2)$.

2. $T(n) = T(2n/3) + 1$
   Da $f(n) = 1 = \Theta(n^{\log_{3/2} 1}) = \Theta(1)$ folgt mit dem zweiten Fall, dass $T(n) = \Theta(\log n)$.

3. $T(n) = 3T(n/4) + n \log n$

   Es ist $f(n) = n \log n = \Omega(n^{\log_4 3+\epsilon})$ mit $\epsilon = 0,2$.
   Da ausserdem $3f(n/4) = 3(n/4 \log(n/4)) \leq (3/4)n \log n$ gilt die Regularitätsbedingung für $c = 3/4$.
   Also folgt mit dem dritten Fall, dass $T(n) = \Theta(n \log n)$.

2.7 Analyse im mittleren Fall

RUB

2.7.1 Inkrementieren eines Zählers

Ein Array $a[0 \ldots n-1]$ in dem Nullen und Einsen stehen. Wir wollen die durch die Bits in a dargestellte Zahl um 1 erhöhen (mod 2^n).

$i := 0$
while $(i < n \; and \; a[i] = 1)$ **do** $a[i] = 0$; i++
if $i < n$ **then** $a[i] = 1$

Wie häufig wird die while-Schleife durchlaufen?

- Im besten Fall 0-mal, im schlechtesten Fall n-mal.
- Und im mittleren Fall?

$$T(n) = \frac{1}{|\{I | \text{size}(I) = n\}|} \sum_{\{I | \text{size}(I) = n\}} \text{time}(I)$$

$\rightarrow$ definiere zugrundeliegenden Wahrscheinlichkeitsraum und berechne erwartete Anzahl

2.7 Analyse im mittleren Fall

2.7.1 Inkrementieren eines Zählers

jede n-Bitfolge hat gleiche Wahrscheinlichkeit $1/2^n$

die while-Schleife wird k-mal ausgeführt gdw.

$$[k < n \ \wedge \ a[0] = \ldots = a[k-1] = 1 \ \wedge \ a[k] = 0]$$
$$\vee \ [k = n \ \wedge \ a[0] = \ldots = a[k] = 1]$$

$$\begin{aligned}
\mathbb{E}(\text{Anzahl Durchläufe}) &= \textstyle\sum_{k=1}^{n-1} k \cdot 2^{-(k+1)} + n2^{-n} \\
&\leq \textstyle\sum_{k=1}^{n} k \cdot 2^{-k} \\
&\leq \textstyle\sum_{k \geq 1} k \cdot 2^{-k} \\
&= 2 = \Theta(1)
\end{aligned}$$

2.7.2 Zwischenmaxima

RUB

Wir suchen den größten Eintrag in einem Array $a[1 \ldots n]$.

$m := a[1]; \qquad \textbf{for } i := 2 \textbf{ to } n \textbf{ do if } a[i] > m \textbf{ then } m := a[i]$

Wie häufig wird die Zuweisung $m := a[i]$ ausgeführt?

- Im besten Fall 0-mal, im schlechtesten Fall $n-1$-mal.
- Und im mittleren Fall?

Wahrscheinlichkeitsraum:

Annahme: n unterschiedliche Zahlen, jede Anordnung gleich wahrscheinlich

$\rightarrow n!$ Permutationen, jede mit Wahrscheinlichkeit $1/n!$

Zwischenmaxima: Eintrag $a[i]$ mit $a[i] > a[j]$ für alle $j < i$

$\#$ Zuweisungen $m := a[i] = \#$ Zwischenmaxima -1

Was ist die erwartete Anzahl M_n von Zwischenmaxima bei n Zahlen?

Indikatorvariablen I_j: $a[j]$ ist Zwischenmaxima.

$$\mathbb{E}(M_n) = \mathbb{E}(I_1 + \ldots + I_n) = \mathbb{E}(I_1) + \ldots + \mathbb{E}(I_n)$$
$$= \mathbb{P}[I_1 = 1] + \ldots + \mathbb{P}[I_n = 1] = \sum_{k=1}^{n} 1/k = H_n = O(\log n)$$

3.3 Amortisierte Analyse

RUB

Was sind die Kosten von m Operationen?

Gegeben eine Folge $\langle \mathsf{Op}_1, \ldots, \mathsf{Op}_m \rangle$ von Operationen auf einer Datenstruktur mit Startzustand s_0. Betrachte Folge $s_0 \overset{\mathsf{Op}_1}{\to} \ldots \overset{\mathsf{Op}_m}{\to} s_m$. Die Kosten hierfür betragen $\sum_{i=1}^{m} T_{\mathsf{Op}_i}(s_{i-1})$.

Einfaches Beispiel: Inkrementieren eines Zählers.

Was kosten m sukzessive Inkrementierungen von $a = \langle 0 \ldots 0 \rangle$?

Wir wissen bereits: eine Inkrementierung kostet erwartet $O(1)$?
Jetzt: m Inkrementierungen kosten $O(m)$.

bin(m) hat $L := \lceil \log m \rceil$ bits. Von den Zahlen von 0 bis $m-1$ enden nur $\leq 2^{L-(k+1)}$ binär auf $0\underbrace{1 \ldots 1}_{k}$.

Kosten $\leq \sum_{k=0}^{L-1} 2^{L-k-1} \leq 2^L \sum_{k \geq 1} k/2^k = 2 \cdot 2^L \leq 4m$

3.3 Amortisierte Analyse

RUB

Was sind die Kosten von m Operationen?

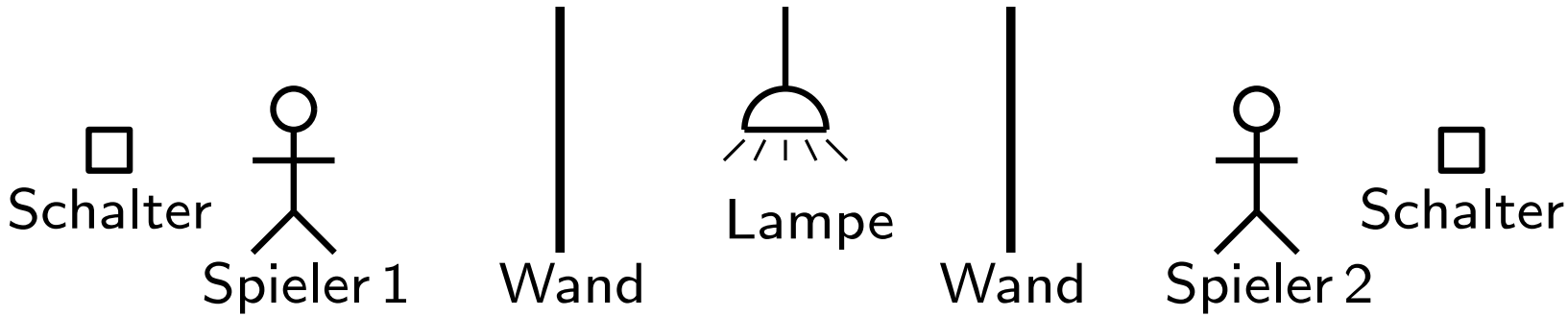
Gegeben eine Folge $\langle \mathsf{Op}_1, \ldots, \mathsf{Op}_m \rangle$ von Operationen auf einer Datenstruktur mit Startzustand s_0. Betrachte Folge $s_0 \overset{\mathsf{Op}_1}{\to} \ldots \overset{\mathsf{Op}_m}{\to} s_m$. Die Kosten hierfür betragen $\sum_{i=1}^{m} T_{\mathsf{Op}_i}(s_{i-1})$.

Verschiedene Methoden:

- Aggregatmethode
- Bankkontomethode
- Potentialmethode

2.8 Randomisierte Algorithmen

Einfaches Beispiel: Betrachte folgendes Spiel.



Beide Spieler können unabhängig voneinander einen Schalter umlegen. Die Lampe geht an, und die Spieler gewinnen, wenn einer von ihnen dies tut. Was ist eine gute Strategie?

Deterministische Strategie: Schalte oder schalte nicht führt zu Verlust.
Randomisierte Strategie: Beide schalten mit Wahrscheinlichkeit $1/2$ führt mit Wahrscheinlichkeit $1/2$ zum Gewinn.

Weiteres Beispiel: Felizitas nimmt an einer Spielshow teil.

100 geschlossene Kästchen mit den Zahlen $1 \ldots 100$. Felizitas darf die Kästchen in beliebiger Reihenfolge öffnen und sie hat 10 Spielmarken. Jedesmal wenn sie ein Kästchen mit einer höheren Zahl als alle bisherigen öffnet, muss sie eine Marke abgeben. Muss sie alle Marken abgeben, so hat sie verloren. Der Spielmaster schlägt ihr Kästchen vor, die sie öffnen soll.

Was ist eine gute Strategie?

Bei *zufälliger Reihenfolge* treten $H_{100} < 6$ Zwischenmaxima auf!

2.8 Randomisierte Algorithmen

RUB

Es gibt zwei Arten:

- **Las-Vegas-Algorithmen:** die Laufzeit (nicht das Ergebnis) hängt vom Zufall ab
- **Monte-Carlo-Algorithmen:** das Ergebnis (nicht die Laufzeit) hängt vom Zufall ab

Wir betrachten Erstere.

2.10 P und NP

RUB

Ein Algorithmus mit polynomieller Laufzeit heisst effizient.

Es gibt wichtige Probleme, für die keine effizienten Algorithmen bekannt sind (z.B. SAT, Hamiltonkreis, Clique, TSP, Rucksack).

Aus der Theoretischen Informatik kennen Sie möglicherweise die Komplexitätsklassen:

P = Klasse aller Probleme, für die es einen deterministischen Algorithmus mit polynomieller Laufzeit gibt.

NP = Klasse aller Probleme, für die es einen nicht-deterministischen Algorithmus mit polynomieller Laufzeit gibt.

Offene Frage: $P \neq NP$?

Für obige Probleme hat man gezeigt, dass sie NP-schwer sind.

Ankündigungen

- Vorlesung am Di 24.4. zu Graphen (Kap 2.9)
- Vorlesung am Do 26.4. zu Arrays und Listen (Kap 3)
  beginnt vermutlich erst um 14:45 Uhr