Computerorientierte Mathematik I

10. Vorlesung

Carsten Gräser

Freie Universität Berlin

20.12.2019

Rückblick: Vorlesung vom 13.12.2019

Freie Universität Berlin

Stabilitätsabschätzungen und Effizienz

Auswertungsbäume zur systematischen Stabilitätsabschätzung

- Gesamtfehlerabschätzung

Summationsalgorithmen

- Rekursive Summation, Auswertungsbäume, Stabilitätsanalyse
- Hierarchische Summation

Komplexität und Aufwand

- Aufwand: Anzahl dominanter Operationen
- Landau-Symbol: $O(n)$, Beispiel
- Definition: Aufwand eines Algorithmus, Komplexität eines Problems
- Beispiel: Summationsalgorithmen (Aufwand, Komplexität)

Aufwand und Komplexität

Gegeben:

- Problem: (P)
- Menge aller Algorithmen zur Lösung von (P): $\mathcal{A}(P)$
- Eingabedaten der Länge n
- Referenzoperationen (problemspezifisch)

Definition (2.4)

Der Aufwand $T_A(n)$ eines Algorithmus $A \in \mathcal{A}(P)$ ist das

Maximum der benötigten Anzahl von Referenzoperationen
über alle zulässigen Eingabedaten der Länge n.

Definition (2.5)

Die Komplexität $\mathcal{K}_P(n)$ von (P) ist

$$\mathcal{K}_P(n) = \inf_{a \in \mathcal{A}(P)} T_A(n).$$

Gegeben:

- Problem: (P)
- Algorithmus $A \in \mathcal{A}(P)$ zur Lösung von (P)

Ein Algorithmus A ist effizient, wenn

- Theoretische Informatik: $T_A(n) = O(n^p)$
- Numerische Mathematik: $T_A(n) = O(\mathcal{K}_P(n))$

Definition

Landau-Symbol $O(\cdot)$ für $f(n), g(n) \to \infty$ für $n \to \infty$

$$f(n) \in O(g(n)) \quad \text{für } n \to \infty \qquad \Leftrightarrow \qquad \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty.$$

Beispiel: $18n^3 + 3n^2 + \sin(e^n) \in O(n^3)$

Gegeben:

Liste von Werten $z_1, \ldots, z_n \in \mathbb{R}$ Gesucht:

Eine Permutation (Umordnung) π mit: $z_{\pi(1)} \leq z_{\pi(2)} \leq \cdots \leq z_{\pi(n)}$

Algorithmus: TumbSort

- Alle Umordnungen durchprobieren
- Aufwandsmaß: Vergleich zweier Werte
- Aufwand von TumbSort: $O(n^n)$

Algorithmus: BubbleSort

- Sukzessive Maximierung
- Aufwandsmaß: Vergleich zweier Werte

Algorithmus: BubbleSort

- Sukzessive Maximierung
- Aufwandsmaß: Vergleich zweier Werte
- Pro Durchgang
  - Gehe vom Anfang der Liste bis zum Ende
  - Wenn $z_{i-1} > z_i$, tausche z_{i-1} und z_i
  - Nach dem Durchgang ist das größte Element am Ende

Algorithmus: BubbleSort

- Sukzessive Maximierung
- Aufwandsmaß: Vergleich zweier Werte
- Pro Durchgang
  - Gehe vom Anfang der Liste bis zum Ende
  - Wenn $z_{i-1} > z_i$, tausche z_{i-1} und z_i
  - Nach dem Durchgang ist das größte Element am Ende
- Nach n Durchgängen ist die Liste sortiert

Algorithmus: BubbleSort

- Sukzessive Maximierung
- Aufwandsmaß: Vergleich zweier Werte
- Pro Durchgang
  - Gehe vom Anfang der Liste bis zum Ende
  - Wenn $z_{i-1} > z_i$, tausche z_{i-1} und z_i
  - Nach dem Durchgang ist das größte Element am Ende
- Nach n Durchgängen ist die Liste sortiert
- Vereinfachung: Höre in k-ten Durchgang beim $i = n - k$ auf

BubbleSort

Algorithmus: BubbleSort

- Sukzessive Maximierung
- Aufwandsmaß: Vergleich zweier Werte
- Pro Durchgang
  - Gehe vom Anfang der Liste bis zum Ende
  - Wenn $z_{i-1} > z_i$, tausche z_{i-1} und z_i
  - Nach dem Durchgang ist das größte Element am Ende
- Nach n Durchgängen ist die Liste sortiert
- Vereinfachung: Höre in k-ten Durchgang beim $i = n - k$ auf

Beispiel: (Tafel)

Algorithmus: BubbleSort

- Sukzessive Maximierung
- Aufwandsmaß: Vergleich zweier Werte
- Pro Durchgang
  - Gehe vom Anfang der Liste bis zum Ende
  - Wenn $z_{i-1} > z_i$, tausche z_{i-1} und z_i
  - Nach dem Durchgang ist das größte Element am Ende
- Nach n Durchgängen ist die Liste sortiert
- Vereinfachung: Höre in k-ten Durchgang beim $i = n - k$ auf

Beispiel: (Tafel)

Aufwand von BubbleSort

$$(n-1) + (n-2) + \cdots + 1 = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = O(n^2) \ll O(n^n)$$

Lemma (2.7)

Gegeben seien zwei sortierte Mengen

$$S_x = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad S_y = \{y_1 \leq y_2 \leq \cdots \leq y_n\}.$$

Dann läßt sich die Menge $S = S_x \cup S_y$ mit $n + m$ Vergleichen sortieren.

Beweis:

- Reißverschluss-System
- Wähle sukzessive das kleinste Element aus $S_x \cup S_y$
- Füge kleinstes Element in neue Liste ein, entferne es aus S_x bzw. S_y.
- Beispiel
- Worst-case: $n + m$ Vergleiche

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Satz

Der Aufwand von MergeSort ist $O(n \cdot \log(n))$.

Beweis:

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Satz

Der Aufwand von MergeSort ist $O(n \cdot \log(n))$.

Beweis:

- Annahme: $n = 2^J$ (ggf. vergrößere n)

MergeSort

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Satz

Der Aufwand von MergeSort ist $O(n \cdot \log(n))$.

Beweis:

- Annahme: $n = 2^J$ (ggf. vergrößere n)
- Wir zeigen: $T(n) \leq n \log_2(n)$

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Satz

Der Aufwand von MergeSort ist $O(n \cdot \log(n))$.

Beweis:

- Annahme: $n = 2^J$ (ggf. vergrößere n)
- Wir zeigen: $T(n) \leq n \log_2(n)$
- Induktionsannahme: $T(\frac{n}{2}) \leq \frac{n}{2} \log_2(\frac{n}{2})$

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Satz

Der Aufwand von MergeSort ist $O(n \cdot \log(n))$.

Beweis:

- Annahme: $n = 2^J$ (ggf. vergrößere n)
- Wir zeigen: $T(n) \leq n \log_2(n)$
- Induktionsannahme: $T(\frac{n}{2}) \leq \frac{n}{2} \log_2(\frac{n}{2})$
- Aufwand (sortiere und vereinige Teillisten):

$$\begin{aligned} T(n) &= T(n/2) + T(n/2) + n \leq n \log_2(\tfrac{n}{2}) + n \\ &= n \log_2(n) - n \log_2(2) + n = n \log_2(n). \end{aligned}$$

Algorithmus: MergeSort

- Teile die Liste in zwei gleichgroße Teile
- Sortiere beide Teile mit MergeSort
- Füge die sortierten Teile zusammen
- Beispiel (Tafel)

Satz

Der Aufwand von MergeSort ist $O(n \cdot \log(n))$.

Beweis:

- Annahme: $n = 2^J$ (ggf. vergrößere n)
- Wir zeigen: $T(n) \leq n \log_2(n)$
- Induktionsannahme: $T(\frac{n}{2}) \leq \frac{n}{2} \log_2(\frac{n}{2})$
- Aufwand (sortiere und vereinige Teillisten):

$$\begin{aligned} T(n) &= T(n/2) + T(n/2) + n \leq n \log_2(\tfrac{n}{2}) + n \\ &= n \log_2(n) - n \log_2(2) + n = n \log_2(n). \end{aligned}$$

Tatsächlich ist der Aufwand $T(n) \in O(n \log(n))$ optimal.

Definition (4.1, Größter gemeinsamer Teiler)

Eine Zahl $d \in \mathbb{N}$, die zwei Zahlen $a, b \in \mathbb{N}$ teilt, ($d|a$ und $d|b$) heißt gemeinsamer Teiler von a und b.
Die größte positive Zahl $d \in \mathbb{N}$, die gemeinsamer Teiler von a und b ist heißt größter gemeinsamer Teiler von a und b oder kurz

$$ggT(a,b) = \max\{d \in \mathbb{N} \,\big|\, d|a \text{ und } d|b\}.$$

Problem:

Berechne $ggT(a,b)$ für Eingabedaten $a \geq b \in \mathbb{N} \setminus \{0\}$.

TumbGGT (Algorithmus 4.3)

Freie Universität Berlin

Eingabe: $a, b \in \mathbb{N}$ mit $a \geq b > 0$

Ausgabe: $ggT(a, b)$.

Algorithmus: TumbGGT

- Setze `ggT := 1`
- Für `i = 2,...,b`
  - Wenn `i|a` und `i|b`, dann
    - Setze `ggT := i`
- `return ggT`

Aufwandsmaß: Anzahl der Divisionen mit Rest

Aufwand: $2(b-1)$

Geschickteres Ausprobieren: TumbGGT++

Eingabe: $a, b \in \mathbb{N}$ mit $a \geq b > 0$

Ausgabe: $ggT(a, b)$.

Algorithmus: TumbGGT++

- Für `i = b,...,2`
  - Wenn `i|a` und `i|b`, dann
    - `return i`
- `return 1`

Aufwandsmaß: Anzahl der Divisionen mit Rest

Aufwand: $2(b-1)$

Definition (4.6, Kongruenz)

Der bei Division von a durch b bleibende Rest heißt

$$a \bmod b = a - \lfloor a/b \rfloor b.$$

$m, n \in \mathbb{Z}$ heißen kongruent modulo b ($m \equiv n$) falls $m \bmod b = n \bmod b$.

Module-Funktion: $\texttt{mod}(a, b) = a \bmod b$

Lemma (Rekursionssatz für den ggT)

Es gilt

$$ggT(a, b) = ggT(b, a \bmod b) \qquad \forall a, b \in \mathbb{N}, a \geq b$$

Umsetzung: Der Euklidische Algorithmus

Freie Universität Berlin

Eingabe: $a, b \in \mathbb{N}$ mit $a \geq b > 0$

Ausgabe: $ggT(a, b)$.

Euklidischer Algorithmus:

- Setze `m = a`
- Setze `n = b`
- Solange `n > 0`
  - Rerechne `r = m modulo n`
  - Setze `m = n`
  - Setze `n = r`
- `return m`

Satz

Der Euklidische Algorithmus terminiert nach höchstens b Schritten.

Definition (Fibonacci-Zahlen)

Die durch die Drei-Term-Rekursion

$$F_{n+1} = F_n + F_{n-1}, \qquad F_0 = 0, F_1 = 1$$

definierten Zahlen $F_k, k = 0, 1, \ldots$ heißen Fibonacci-Zahlen.

Lemma (4.14)

Es sei $a > b$. Terminiert der Euklidische Algorithmus nach genau $k \geq 1$ Schritten, so gilt $a \geq F_{k+1}$ und $b \geq F_k$.

Satz (4.15, Lamé, 1848)

Es gelte $a > b$ und $b < F_{k+1}$ mit $k \in \mathbb{N}$. Dann terminiert der Euklidische Algorithmus nach höchstens k Schritten.

Bemerkung: Für $b = F_k, a = F_{k+1}$ braucht man k Schritte.

Aufwandsschranken für den Euklidischen Algorithmus

Moivre-Binet Formel: $F_k = (\phi^k - (1-\phi)^k)/\sqrt{5}, \qquad \phi = \frac{1+\sqrt{5}}{2}$

Satz (4.16)

Für den Aufwand $T_E(n)$ des Euklidischen Algorithmus bei Anwendung auf eine n-stellige Zahl b und $a > b$ gilt

$$T_E(n) \leq 1 + \log(\phi)^{-1} n.$$

Bemerkung

Das ist eine exponentielle Verbesserung gegebüber TumbGGT!

Beispiel

$n = \log(10000) \ll 10000$