Übungen zu Algorithmentechnik WS 09/10

1. Kurzsitzung

Thomas Pajor

Algorithmics Group
Faculty of Informatics
Karlsruhe Institute of Technology
Universität Karlsruhe (TH)

22. Oktober 2009

Eure Übungsleiter

Tanja Hartmann
t.hartmann@kit.edu
Raum 306, Gebäude 50.34

Thomas Pajor
pajor@kit.edu
Raum 322, Gebäude 50.34

Sprechzeiten nach Vereinbarung oder kommt einfach vorbei!

Organisatorisches

Übungstermine

Grob alle zwei Wochen, donnerstags 15:45 Uhr.

Vorläufige Liste der Übungstermine:

Termin	*n*. Übung	Hörsaal	Person
Do, 22.10.2009	1. Übung	Tulla	Thomas
Do, 05.11.2009	2. Übung	Tulla	Tanja
Do, 19.11.2009	3. Übung	Tulla	Thomas
Do, 03.12.2009	4. Übung	Tulla	Tanja
Do, 17.12.2009	5. Übung	Tulla	Thomas
Di, 12.01.2010	6. Übung	Neue Chemie	Tanja
Do, 21.01.2010	7. Übung	Tulla	Thomas
Do, 04.02.2010	8. Übung	Tulla	Tanja

Organisatorisches

Algorithmentechnik-Homepage (mit Forum)

`http://i11www.ira.uka.de/teaching/winter2009/algotech/`

Elektronische Kurswaren

- Vorläufiges Skript
  Wird evtl. ergänzt und verbessert!
- Übungsblätter und Lösungsvorschläge
- Übungsfolien

Übungsablauf

i-te Übung

- Ausgabe des i-ten Übungsblattes
- Besprechung der Aufgaben des $(i-1)$-ten Übungsblattes
- Eventuell kurze Ergänzungen zum Stoff, bzw. Wiederholung (heute)

Übungsblätter (werden von uns korrigiert!)

- Freiwillige Abgabe, kein Schein, kein Klausurbonus
- Aber: Helfen euch beim Verständnis des Stoffs!

$\Rightarrow$ **Signifikant** weniger Lernaufwand für die Klausur und bessere Noten

Übungsblätter

Auf die Übungsblätter kommt. . .

- Name und E-Mail Adresse
- **Nicht:** Matrikelnummer

Organisatorisches

- **Ausgabe:** In der i-ten Übung
- **Abgabe:** 3. Stock, Geb. 50.34, ITI Wagner im grünen Kasten bei Raum 319
- **Rückgabe:** In der $(i + 1)$-ten Übung

Klausur

Zwei Klausurtermine (vorläufig!):

- **Hauptklausur:** Montag, 1. März 2010
- **Nachklausur:** Freitag, 16. April 2010

Verbindlich ist: `http://www.informatik.kit.edu/klausuren.php`

Thema der heutigen Übung

Aufwandsanalyse von Algorithmen

- Wiederholung: Landau-Notation (O-Kalkül)
- Wiederholung: Amortisierte Analyse
- Wiederholung: "Divide-and-Conquer"-Verfahren und Rekursion

Aufwandsabschätzungen

Landau Notation:
Charakterisiert *asymptotisches* Laufzeitverhalten von Algorithmen.

Obere Abschätzungen:

$f \in O(g)$ (f wächst höchstens so schnell wie g):

$$f \in O(g) \quad :\Leftrightarrow \quad \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n > n_0 : f(n) \leq c \cdot g(n)$$

$f \in o(g)$ (f wächst weniger schnell als g):

$$f \in o(g) \quad :\Leftrightarrow \quad \forall c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n > n_0 : f(n) < c \cdot g(n)$$

Aufwandsabschätzungen

Untere Abschätzungen:

$f \in \Omega(g)$ (f wächst mindestens so schnell wie g):

$$f \in \Omega(g) \quad :\Leftrightarrow \quad \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n > n_0 : f(n) \geq c \cdot g(n)$$

$f \in \omega(g)$ (f wächst schneller als g):

$$f \in \omega(g) \quad :\Leftrightarrow \quad \forall c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n > n_0 : f(n) > c \cdot g(n)$$

Aufwandsabschätzungen

Exakte Abschätzung:

$f \in \Theta(g)$ (f wächst genauso schnell wie g):

$$f \in \Theta(g) \quad :\Leftrightarrow \quad \exists c_0, c_1 \in R^+, \exists n_0 \in \mathbb{N} : \\ \forall n > n_0 : c_0 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)$$

Es gilt:

- $O(g) \cap \Omega(g) = \Theta(g)$
- $o(g) \cap \omega(g) = \emptyset$
- $f_1 + f_2 \in O(f_1)$ wenn $f_2 \in o(f_1)$
  Beispiel: $n^2 + n \in O(n^2)$
- Weitere Rechenregeln siehe Info II

Laufzeit-Aufwände von Algorithmen

Sei $t_{\mathcal{A}}(x)$ die Anzahl Operationen, die ein Algorithmus bei Eingabe x ausführt.

Worst-Case-Laufzeit:

$$T_{\mathcal{A}}^{\text{worst}}(n) := \max_{\substack{x \text{ Eingabe,} \\ |x|=n}} \{t_{\mathcal{A}}(x)\}$$

Average-Case-Laufzeit:

$$T_{\mathcal{A}}^{\text{avg}}(n) := \frac{1}{|\{x \mid x \text{ Eingabe}, |x| = n\}|} \sum_{x:|x|=n} t_{\mathcal{A}}(x)$$

Beispiel: Vergleichsbasiertes Sortieren

Gegeben:
Ein Array $A = A[1, \dots, n]$ von Objekten

Gesucht:
Algorithmus, der A sortiert und dabei die Schlüssel in A nur durch Vergleiche $\leq$ auswertet.

Algorithmen

- Bubblesort: $T^{\text{worst}}(n) \in O(n^2)$, $T^{\text{avg}}(n) \in O(n^2)$
- Quicksort: $T^{\text{worst}}(n) \in O(n^2)$, $T^{\text{avg}}(n) \in O(n \log n)$
- Mergesort: $T^{\text{worst}}(n) \in O(n \log n)$, $T^{\text{avg}}(n) \in O(n \log n)$
- Für alle Algorithmen: $T^{\text{worst}}(n) \in \Omega(n \log n)$

Amortisierte Analyse

Gegeben:
Folge von n Operationen.

Nachteil gewöhnlicher Analyse:
Jede Operation wird einzeln betrachtet, Gesamtkosten orientieren sich an Operation mit höchsten Kosten.

Ziel von amortisierter Analyse:
Zeige, dass die mittleren (amortisierten) Kosten einer Operation "klein" sind, auch wenn eine einzelne Operation mal hohe Kosten haben darf.

Prinzipielles Vorgehen:
Mittle die Kosten über die Gesamtkosten aller auszuführenden Operationen.

Vorteile Amortisierter Analyse

Manchmal bessere Laufzeitaussagen möglich!

Amortisierte Analyse $\neq$ Average-Case Analyse:

- Keine Annahmen über die Verteilung der Eingabe nötig.

Häufiger Anwendungsfall:
Analyse von Operationen einer Datenstruktur $\mathcal{D}$.

Beispiel: Stack mit MULTIPOP

Datenstruktur S mit “Last-in, First-out” und den Operationen

- PUSH(S, x) – lege Objekt x auf Stack S
- POP(S) – gib oberstes Objekt von S aus und lösche es vom Stack

Aufwand: jeweils $O(1)$, veranschlage Kosten 1.

Zusätzliche Operation

- MULTIPOP(S, k) – gib obersten k Objekte von S aus und lösche sie vom Stack. Falls $k > |S|$, gib alle Objekte von S aus, bis S leer.

Aufwand: linear in Anzahl POPs, also $O(\min(|S|, k))$.

Beispiel

				Ausgabe: 17				Ausgabe: 8,1,8
		17						
8		8		8				
1		1		1				
8		8		8				
9		9		9				9
7		7		7				7
4	$\xrightarrow{\textsc{Push}(S,17)}$	4	$\xrightarrow{\textsc{Pop}(S)}$	4		$\xrightarrow{\textsc{Multipop}(S,3)}$		4

Ausgabe: 9,7,4

$\xrightarrow{\textsc{Multipop}(S,3)}$

Klassische Analyse

Gegeben:
Folge von n Push-, Pop- und Multipop-Operationen.

Es ergibt sich

- Worst-case Laufzeit einer Multipop-Operation: $O(n)$
- Worst-case Laufzeit für beliebige Operation damit auch $O(n)$
- Für n Operationen daher: Worst-case $O(n^2)$!

Geht das nicht auch besser?

Verfahren zur Amortisierten Analyse

1. **Ganzheitsmethode (Aggregat-Analyse)**
   - Bestimme obere Schranke $T(n)$ für die *Gesamtkosten* von n Operationen.

$\Rightarrow$ Amortisierten Kosten pro Operation sind $T(n)/n$.

Verfahren zur Amortisierten Analyse

2. Buchungsmethode (engl. accounting)

- Bestimme amortisierte Kosten für jede Operation.
- Verschiedene Kosten für verschiedene Operationen möglich.
- Frühe Operationen bekommen zu hohe Kosten für festgelegte Objekte, die als “Kredit” veranschlagt werden.
- Spätere Operationen können Kredit früherer Opertaionen verbrauchen.

Verfahren zur Amortisierten Analyse

3. Potentialmethode (engl. accounting)

- Sei $\mathcal{D}_i$ die zugrundeliegende Datenstrukturr im i-ten Schritt.
- Seien c_i die *tatsächlichen* Kosten von Schritt i.
- Ordne jedem $\mathcal{D}_i$ ein Potential $\Phi(\mathcal{D}_i)$ zu.
- Ordne dem i-ten Schritt amortisierte Kosten zu:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Damit gilt:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(\mathcal{D}_i) - \Phi(\mathcal{D}_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(\mathcal{D}_n) - \Phi(\mathcal{D}_0)$$

$\Rightarrow$ $\sum_{i=1}^{n} \hat{c}_i$ ist obere Schranke für die tatsächlichen Kosten **falls** $\Phi(\mathcal{D}_n) \geq \Phi(\mathcal{D}_0)$.

Divide-and-Conquer: Das Prinzip der Rekursion

Prinzip

- Divide-and-Conquer Verfahren zerlegen Probleme in kleinere Teilprobleme (häufig desselben Typs). Aus den Lösungen dieser Hilfsprobleme wird eine Lösung für das Ausgangsproblem konstruiert.
- Häufiger Spezialfall: Rekursive Algorithmen (z. B. Mergesort aus Informatik II)

Frage: Wie kann man worst-case Laufzeiten von solchen Algorithmen ermitteln?

Methoden zur Rekursionsabschätzung

Die folgenden Methoden stehen zur Verfügung:

- **Substitutionsmethode:**
  Wir vermuten eine Lösung und beweisen deren Korrektheit induktiv.
- **Iterationsmethode:**
  Die Rekursionsabschätzung wird in eine Summe umgewandelt und dann mittels Techniken zur Abschätzung von Summen aufgelöst.
- **Aufteilungs- und Beschleunigungssatz:**
  Man beweist einen allgemeinen Satz zur Abschätzung von rekursiven Ausdrücken; Zum Beispiel der Form

$$T(n) = a \cdot T(n/b) + f(n), \quad \text{wobei} \quad a \geq 1 \quad \text{und} \quad b > 1.$$

Aufteilungs- und Beschleunigungssatz

Theorem (Aufteilungs- und Beschleunigungssatz)

Gegeben: *Rekursion der Form*

$$T(n) = a \cdot T(n/b) + f(n) \qquad n \in \mathbb{N}$$

mit $a \geq 1$, $b > 1$.

Es gilt dann:

(i) $T(n) \in \Theta(f(n))$ *falls* $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ *und* $a \cdot f(n/b) \leq c \cdot f(n)$ *für eine Konstante* $c < 1$ *und* $n \geq n_0$.

(ii) $T(n) \in \Theta(n^{\log_b a} \log n)$ *falls* $f(n) \in \Theta(n^{\log_b a})$.

(iii) $T(n) \in \Theta(n^{\log_b a}) $ *falls* $f(n) \in O(n^{\log_b a - \varepsilon})$.

Aufteilungs- und Beschleunigungssatz

Theorem (Aufteilungs- und Beschleunigungssatz (allgemein))

Gegeben: *Rekursion der Form*

$$T(n) = \sum_{i=1}^{m} T(\alpha_i n) + f(n)$$

mit $m \geq 1$, $0 < \alpha_i < 1$ *und* $f(n) \in \Theta(n^k)$, $k \geq 0$.

Es gilt dann:

(i) $T(n) \in \Theta(n^k)$, *falls* $\sum_{i=1}^{m} \alpha_i^k < 1$.

(ii) $T(n) \in \Theta(n^k \log n)$ *falls* $\sum_{i=1}^{m} \alpha_i^k = 1$.

(iii) $T(n) \in \Theta(n^c)$ *falls* $\sum_{i=1}^{m} \alpha_i^k > 1$, *wobei* c *bestimmt wird durch* $\sum_{i=1}^{m} \alpha_i^c = 1$.