Informatik II: Algorithmen und Datenstrukturen SS 2013

Vorlesung 2a, Dienstag, 23. April 2013
(Laufzeitanalyse MinSort und QuickSort)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü1 (Drumherum + Sortieren)
  - Alle non-code Abgaben bitte ausschließlich als **PDF**

    Außer natürlich die erfahrungen.txt !
- Laufzeitanalyse
  - MinSort ... "quadratische" Laufzeit
  - QuickSort ... meistens besser, aber auch nicht ganz "linear"
  - Beweistechnik Vollständige Induktion
  - Rechnen mit dem Logarithmus

Erfahrungen mit dem Ü1 (Sortieren)

- Zusammenfassung / Auszüge   Stand 23. April 15:36
  - Viel Zeit für das Einrichten gebraucht: Programmier-umgebung, SVN, Linux, C++, ... normal beim ersten Mal !
  - Sinn von quickSortDivide manchen nicht klar geworden

    Manche meinten sogar, braucht man nicht ... siehe spätere Folie
  - Beispielhafte Erklärung war nicht gut ... zumindest suboptimal
  - Bei Pivot Methode 1, Laufzeit sehr groß ... stimmt genau
  - Checkstyle war das größte Ärgernis an der Aufgabe
  - Freue mich, einige Vim-Tricks lernen zu können
  - Vorgabe führt zu umständlichen Programmen

    Sie müssen sich nicht dran halten, auch wenn ich es sehr empfehle

Erfahrungen mit dem Ü1 (Sortieren)

- Zusammenfassung / Auszüge ... Fortsetzung
  - Details zur Bedienung in den Video-Aufzeichnungen nachgucken ist recht umständlich

    Guter Punkt, Verbesserungsvorschläge (machbar) willkommen
  - Tempo beim Tippen verringern, sonst muss man die Aufzeichnungen in niedrigerer Geschwindigkeit angucken
  - Meine Prokrastination bringt mich noch ins Grab

    Keine Sorge, ich lebe auch noch
  - Die "10 Stunden pro Übungsblatt" sind ein Witz, oder?

    Eigentlich nicht ... das ist bei ≈ 30 ECTS pro Semester etwa eine 45-Stunden-Woche; Vorlesungsnachbereitung inklusive
  - In ant compile ein mkdir –p ./bin einbauen ... gute Idee

Die Methode quickSortDivide

- Braucht man sehr wohl !
  - Auch wenn die Eingabe gerade falsch herum sortiert ist

7 6 5 4 3 2 1 PIVOT: 5

4 3 2 1 | 7 6 5

rekursiv weiter | rekursiv weiter

1 2 3 4 5 6 7 ✓

Laufzeitanalyse allgemein 1/4

- Wie lange läuft unsere bisherigen Programme?
  - Für MinSort und QuickSort hatten wir dazu bisher zwei Schaubilder und Folgendes beobachtet

    MinSort : Laufzeit wird "unproportional" langsamer, je mehr Zahlen sortiert werden
  - QuickSort: bei schlecht gewähltem Pivot-Element passiert dasselbe, wenn gut gewählt aber nicht
  - Wie können wir präziser fassen, was da passiert?

Laufzeitanalyse allgemein 2/4

- Wie analysieren wir die Laufzeit?
  - Idealerweise hätten wir gerne eine Formel, die uns für eine bestimme Eingabe sagt, wie lange das Programm dann läuft
  - **Problem:** Laufzeit hängt auch noch von vielen anderen Umständen ab, insbesondere
    - auf was für einem Rechner wir den Code ausführen
    - was sonst gerade noch auf dem Rechner läuft
    - welchen Compiler wir benutzt haben
    - Jahreszeit, Mondphase, Aszendent, …
  - **Abstraktion 1:** Deshalb analysieren wir nicht die Laufzeit, sondern die Anzahl der (Grund-)Operationen

Laufzeitanalyse allgemein 3/4

- Was sind unsere Grundoperationen
  - Eine arithmetische Operation, z.B. a + b
  - Variablenzuweisung, z.B. x = y
  - Funktionsaufruf, z.B. Sorter.minSort(array)

    Natürlich zählt nur der **Sprung** zu der Funktion als Grundoperation
  - **Intuitiv:** eine Zeile Code
  - **Genauer wäre:** eine Zeile Maschinencode
  - **Noch genauer wäre:** ein Prozessorzyklus
  - Wir sehen später noch, dass es nicht so wichtig ist, wie genau wir die Grundoperationen definieren

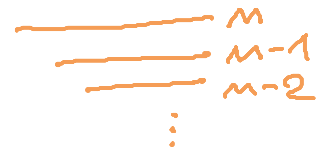
    Wichtig ist für uns hier erst mal, dass die Anzahl Operationen ungefähr **proportional** zur tatsächlichen Laufzeit ist

Laufzeitanalyse allgemein 4/4

- Abschätzung der Anzahl Grundoperationen
  - **Abstraktion 2:** Wir zählen die Operationen nicht genau, sondern berechnen obere (und selten auch untere) Schranken

    Grund: das erleichtert die Sache und wir haben ja eh abstrahiert von exakter Laufzeit zu Anzahl Operationen
  - Sei n die Größe der Eingabe (= des Eingabearrays)
  - **Beobachtung:** Die Anzahl Operationen hängt nur von n ab, nicht davon, welche n Zahlen das sind ... das ist häufig so !
  - Entsprechend untersuchen wir T(n) := die Anzahl der Operationen bei Eingabegröße n

Laufzeitanalyse MinSort 1/2



- Wir zeigen zuerst, dass gilt: $T(n) \le C_1 \cdot n^2$
  - … für irgendeine Konstante C_1

MinSort hat eine äußere Schleife und eine innere Schleife

Iteration 1: $\le A \cdot n + B$

Iteration 2: $\le A \cdot (n-1) + B$

Iteration 3: $\le A \cdot (n-2) + B$

für irgendwelche Konstanten A und B

und so weiter …

$$\Rightarrow T(n) \le A \cdot (\underbrace{n + n-1 + n-2 + \dots + 1}_{\sum_{i=1}^{n} i \,=\, \frac{1}{2} n \underbrace{(n+1)}_{\le 2n} \,\le\, n^2}) + n \cdot B$$

$$\le A \cdot n^2 + \underbrace{B \cdot n}_{\le n^2} \le \underbrace{(A+B)}_{=: C_1} \cdot n^2 \quad \blacksquare \quad \text{Q.E.D.}$$

Laufzeitanalyse MinSort 2/2

- Wir zeigen, dass auch gilt: $T(n) \geq C_2 \cdot n^2$
  - … für irgendeine (andere) Konstante $C_2 < C_1$

der äußeren Schleife von MinSort

Iteration 1: $\geq A' \cdot n + B'$, $A' < A$, $B' < B$ irgendwelche Konstante

Iteration 2: $\geq A' \cdot (n-1) + B'$

Iteration 3: $\geq A' \cdot (n-2) + B'$

usw.

$$\Rightarrow T(n) \geq A' \cdot \underbrace{(n + n-1 + n-2 + \dots + 1)}_{= \frac{1}{2} n(n+1) \geq \frac{1}{2} \cdot n^2} + B' \cdot n$$

$$\geq A'/2 \cdot n^2 + \underbrace{B' \cdot n}_{\geq 0} \geq \underbrace{A'/2}_{=: C_2} \cdot n^2$$

Quadratische Laufzeit

- Definition
  - Die Laufzeit T hängt von der Eingabegröße n ab
  - Es gibt Konstanten C_1 und C_2 mit $C_1 \cdot n^2 \leq T(n) \leq C_2 \cdot n^2$

Eingabegröße n: $T(n) \sim n^2$
——— " ——— $2n$: $T(2n) \sim (2n)^2 = 4n^2$

- Betrachtungen dazu
  - Doppelt so große Eingabe → viermal so große Laufzeit
  - Unabhängig von den Konstanten wird das schnell **sehr teuer**
    - $C = 1$ ns (1 einfache Anweisung ≈ 1 Nanosekunde)
    - $n = 10^6$ (1 Millionen Zahlen = 4 MB) [bei 4 Bytes/Zahl]
      - $C \cdot n^2 = 10^{-9} \cdot 10^{12} = 10^3$ s = 16.7 Minuten
    - $n = 10^9$ (1 Milliarde Zahlen = 4 GB)
      - $C \cdot n^2 = 10^{-9} \cdot 10^{18} = 10^9$ s = 31.7 Jahre

Quadr. Laufzeit = "große" Probleme unlösbar

Beweis über vollständige Induktion 1/2

- Prinzip
  - Man möchte beweisen, dass eine Aussage für alle natürlichen Zahlen gilt, also: $A(n)$ gilt für alle $n \in \mathbb{N}$
  - Dann hat ein Induktionsbeweis zwei Schritte
  - **Induktionsanfang**: Wir zeigen, dass $A(1), \ldots, A(k)$ gelten

    Meistens reicht $k = 1$, aber für's Ü2 braucht man $k = 2$
  - **Induktionsschritt:** Wir nehmen für ein beliebiges $n > k$ an, dass $A(1), \ldots, A(n-1)$ gelten, und zeigen: dann gilt auch $A(n)$

    Meistens braucht man dazu nur $A(n-1)$, für's Ü2 auch $A(n-2)$
  - Wenn wir die beiden Sachen gezeigt haben, haben wir nach dem Prinzip der **vollständigen Induktion** gezeigt, dass $A(n)$ für alle natürlichen Zahlen n gilt

Beweis über vollständige Induktion 2/2

- Beispiel
  - Wir haben gerade benutzt: $\Sigma_{i=1..n}\, i = ½ \cdot n \cdot (n + 1)$

Induktionsanfang:

$n = 1: \quad \sum_{i=1}^{1} i = 1 \quad ; \quad \frac{1}{2} \cdot n(n+1) = 1 \qquad \blacksquare$

Induktionsschritt:

$n \to n+1: \quad \sum_{i=1}^{n+1} i$ soll sein $\frac{1}{2}(n+1)(n+2)$

darf dabei benutzen $\sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$

(Induktionsvoraussetzung)

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + n + 1 \underset{IV}{=} \frac{1}{2} n(n+1) + n + 1$$

$$= \left(\frac{1}{2} n + 1\right)(n+1) = \frac{1}{2}(n+2)(n+1) \qquad \blacksquare$$

Laufzeitanalyse QuickSort 1/6

- Schlechtester Fall (engl. "worst case")
  - Im schlechtesten Fall teilt das Pivot-Element jedes Feld der Größe m in zwei Teilfelder der Größen 1 und $m - 1$
  - Dann gibt es eine Konstante $A > 0$ so dass gilt:

    Für $n \geq 2$: $T(n) \geq T(n\text{-}1) + A \cdot n$

    Für $n = 1$: $T(1) \geq A$
  - Daraus folgt: $T(n) \geq A \cdot (1 + 2 + \ldots + n) \geq A/2 \cdot n^2$

    d.h. im schlechtesten Fall ist auch QuickSort quadratisch!

$$T(n) \geq \underbrace{T(n-1)} + A \cdot n$$
$$\geq \underbrace{T(n-2)} + A \cdot (n-1)$$
$$\geq T(n-3) + A \cdot (n-2)$$
usw.

Laufzeitanalyse QuickSort 2/6

- Bester Fall (engl. "best case") $n = 2^k, k \in \mathbb{N}$
  - Im besten Fall ist n eine Zweierpotenz, und jedes Pivot-Element teilt das entsprechende Feld **genau** in der Mitte:

    Rekursionstiefe 1: 2 Teilfelder der Größe $n/2 = 2^{k-1}$

    Rekursionstiefe 2: 4 Teilfelder der Größe $n/4 = 2^{k-2}$

    Rekursionstiefe 3: 8 Teilfelder der Größe $n/8 = 2^{k-3}$

    Und so weiter ...
  - Dann gibt es eine Konstante $A > 0$ so dass gilt:

    Für $n \geq 2$: $T(n) \leq T(n/2) + T(n/2) + A \cdot n$ $(*)$

    Für $n = 1$: $T(1) \leq A$
  - Daraus folgt: $T(n) \leq A \cdot n \cdot (1 + \log_2 n)$

    Beweis auf der nächsten Folie

Laufzeitanalyse QuickSort 3/6

- Beweis von $T(n) \le A \cdot n \cdot (1 + \log_2 n)$ $\underline{n \ge 1}$

nach Annahme sind $n/2, n/4, n/8, \ldots$ alles ganze Zahlen.

$$T(n) \overset{(*)}{\le} T(n/2) + T(n/2) + A \cdot n$$

$$= 2 \cdot \underbrace{T(n/2)}_{\overset{(*)}{\le}\, 2 \cdot T(n/4) + A \cdot n/2} + A \cdot n$$

$$\le 4 \cdot \underbrace{T(n/4)}_{\overset{(*)}{\le}\, 2 \cdot T(n/8) + A \cdot n/4} + \underbrace{A \cdot n + A \cdot n}_{= 2 \cdot A \cdot n}$$

$$\le 8 \cdot T(n/8) + 3 \cdot A \cdot n$$

usw. $\le 2^k \cdot T(n/2^k) + k \cdot A \cdot n$

$2^k = n \iff k = \log_2 n$ (siehe nächste Folie)

$k = \log_2 n \curvearrowright = n \cdot \underbrace{T(1)}_{\le A} + \log_2 n \cdot A \cdot n$

$$\le A \cdot n \cdot (\underbrace{1 + \log_2 n}_{\text{für } n \ge 2:\ \le 2 \cdot \log_2 n}) \quad \blacksquare$$

Laufzeit proportional zu $n \cdot \log n$

- Schauen wir uns wieder Zahlenbeispiele an
  - Nehmen wir also an, es gibt Konstanten C_1 und C_2 mit

    $C_1 \cdot n \cdot \log_2 n \leq T(n) \leq C_2 \cdot n \cdot \log_2 n$ für $n \geq 2$

    für $n = 1$ $\log_2 n = 0$ $\Rightarrow$ DOOF
  - Dann dauert es bei doppelt so großer Eingabe nur geringfügig mehr als doppelt so lange
  - $C = 1$ ns (1 einfache Anweisung ≈ 1 Nanosekunde)
  - $n = 2^{20}$ (≈ 1 Millionen Zahlen = 4 MB) [bei 4 Bytes/Zahl]
    - $C \cdot n \cdot \log_2 n = 10^{-9} \cdot \underbrace{2^{20}}_{\approx 10^6} \cdot 20$ s = 21 Millisekunden
  - $n = 2^{30}$ (≈ 1 Milliarde Zahlen = 4 GB)
    - $C \cdot n \cdot \log_2 n = 10^{-9} \cdot \underbrace{2^{30}}_{\approx 10^9} \cdot 30$ s = 32 Sekunden

Laufzeit $n \cdot \log n$ ist also fast so gut wie linear!

Der Logarithmus   ($\neq$ Algorithmus)

UNI FREIBURG

- Definition Logarithmus
  - Der "Logarithmus zur Basis b" ist gerade die inverse Funktion zu "b hoch"

    Formal: $\log_b n = x \Leftrightarrow b^x = n$

    Beispiel: $\log_2 1024 = 10 \Leftrightarrow 2^{10} = 1024$
- Rechenenregeln ergeben sich dann
  - ... aus den bekannten Rechenregeln für das Potenzieren
  - Beispiel: $\log_b (x \cdot y) = (\log_b x) + (\log_b y)$ ... siehe unten
  - Beispiel : $\log_b x^y = y \cdot \log_b x$ ... Ü2, Aufgabe 2

$z := \log_b (x \cdot y) \Rightarrow b^z = x \cdot y$

$z_1 := \log_b x \Rightarrow b^{z_1} = x$

$z_2 := \log_b y \Rightarrow b^{z_2} = y$

$b^z = x \cdot y = b^{z_1} \cdot b^{z_2} = b^{z_1 + z_2}$

$\Longrightarrow z = z_1 + z_2$

Literatur / Links

- Weiterführende Literatur (bei Interesse)
  - Analyse für zufällig gewähltes Pivot-Element:

    Cormen / Leiserson / Rivest: II.8.4 Analysis of Quicksort

    http://en.wikipedia.org/wiki/Quicksort#Formal_analysis
  - Prokrastination

    https://en.wikipedia.org/wiki/Procrastination