Informatik II: Algorithmen und Datenstrukturen SS 2015

Vorlesung 2a, Dienstag, 28. April 2015

(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

UNI FREIBURG

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü1 (Drumherum + Sortieren)
  - Wahl der Programmiersprache
- Laufzeitanalyse
  - Allgemein … wie fasst man Laufzeit mathematisch
  - MinSort … "quadratische" Laufzeit
  - QuickSort … meistens besser, aber auch nicht ganz "linear"
  - Auffrischung: vollständige Induktion, Logarithmus

Erfahrungen mit dem Ü1 1/3

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 28. April 12:00
  - Probleme mit dem Programmieren ... siehe nächste Folie

    Viele sind aus der Übung oder waren noch nie richtig drin
  - Einrichten etc. hat einige Zeit gekostet

    Aber viel weniger als sonst, wohl weil aus Info 1 bekannt
  - Probleme mit beschränkter Rekursionstiefe
    - Xss1G für Java, sys.setrecursionlimit für Python
  - Probleme, die .TIP Datei in Python zu übersetzen

    Insbesondere: was ist call by value, was ist call by reference
  - Aufwand des Übungsblattes unterschätzt

Erfahrungen mit dem Ü1 2/3

UNI FREIBURG

- Empfehlung
  - Wenn Sie beim Ü1 Probleme mit dem Programmieren bzw. der Umgebung hatten:

    Wählen Sie für die nächsten Übungsblätter **Python !**

    (Die Algorithmen sind schon schwer genug)
  - Wenn Sie sich trotzdem für C++ oder Java entscheiden, sollten Sie akzeptieren, dass Sie dann mehr Zeit brauchen als für die Vorlesung eigentlich veranschlagt

    Wir geben Ihnen auch für Java und C++ Hilfestellung, aber wir können hier im Rahmen der Vorlesung keinen vollständigen Java / C++ Kurs nachholen

Erfahrungen mit dem Ü1 3/3

UNI FREIBURG

- Zusammenfassung
  - Eingabe: war gerade verkehrt herum sortiert
  - Schaubild je nach Wahl des Pivots:

    Pivot fix: Laufzeitverhalten wie bei MinSort

    Pivot zufällig: sieht ziemlich "linear" aus

Laufzeitanalyse allgemein 1/5

UNI FREIBURG

- Wie lange laufen 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, bei guten Pivot aber nicht

Laufzeitanalyse allgemein 2/5

UNI FREIBURG

- Wie können wir das präziser fassen
  - Idealerweise: eine Formel, die uns für eine bestimmte Eingabe sagt, wie lange das Programm darauf läuft
  - Problem: Laufzeit hängt auch noch von vielen anderen Umständen ab, insbesondere
    - auf was für einem Rechner wird den Code ausführen
    - was sonst gerade noch auf dem Rechner läuft
    - welchen Compiler wir benutzt haben
    - Jahreszeit, Mondphase, Aszendent, ...

Laufzeitanalyse allgemein 3/5

UNI FREIBURG

- Abstraktion 1: Anzahl Operationen statt Laufzeit
  - Intuitiv: eine Operation = eine Zeile Code, zum Beispiel:

Arithmetische Operationen	a + b
Variablenzuweisungen	x = y
Verzweigungen	if … else …
Sprung zu einer Funktion	min_sort(…)

  - Genauer wäre: eine Zeile Maschinencode … oder noch genauer: ein Prozessorzyklus

    Wir sehen später noch, dass es nicht so wichtig ist, wie genau wir diese Operationen definieren

    Wichtig ist, dass die Anzahl Operationen ungefähr **proportional** zur tatsächlichen Laufzeit ist

Laufzeitanalyse allgemein 4/5

UNI FREIBURG

- Abstraktion 2: Abschätzung statt genau zählen
  - Meistens Abschätzung nach oben ("obere Schranke")

    Dann wissen wir, wie lange ein Programm höchstens braucht
  - Seltener Abschätzung nach unten ("untere Schranke")

    Dann wissen wir, wie lange ein Programm mindestens braucht

    Schätzen satt genau zählen erleichtert die Aufgabe, und wir haben ja eh schon abstrahiert von der exakten Laufzeit zu der Anzahl Operationen

Laufzeitanalyse allgemein 5/5

UNI FREIBURG

- Abstraktion 3: Schranken pro Eingabegröße n
  - Oft hängt die Laufzeit vor allem von der Größe der Eingabe ab, und nur wenig davon, wie die Eingabe genau aussieht

    Zum Beispiel hängt die Laufzeit von MinSort für eine Eingabegröße n nur minimal von der genauen Eingabe ab
  - Wir schreiben deswegen für die Laufzeit oft **T(n)**

    Wenn wir obere Schranken berechnen wollen, ist damit die **maximale** Laufzeit für eine Eingabe der Größe n gemeint

    Wenn wir untere Schranken berechnen wollen, ist damit die **minimale** Laufzeit für eine Eingabe der Größe n gemeint

    Diese Notation ist mathematisch etwas unpräzise, aber es ist in aller Regel klar, was gemeint ist

Laufzeitanalyse MinSort 1/4

- Es gilt: $T(n) \leq C_1 \cdot n^2$ ... für irgendeine Konstante C_1
  - MinSort hat eine äußere und eine innere Schleife
  - Für jede Iteration der äußeren Schleife, schätzen wir die Anzahl Operationen der inneren Schleife ab

sogar: $\leq A \cdot (n-1) + B \leq A \cdot n + B$

Iteration 1: $\leq A \cdot n + B$, $A, B > 0$

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

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

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

$$\leq A \cdot \frac{1}{2} n (n+1) + n \cdot B$$

$$\leq 2n \qquad \leq n^2$$

$$\leq A \cdot n^2 + B \cdot n^2 \leq \underbrace{(A+B)}_{=: C_1} \cdot n^2$$

Laufzeitanalyse MinSort 2/4

- Es gilt auch: $T(n) \geq C_2 \cdot n^2$ ... für eine Konst. $C_2 < C_1$

für $n \geq 2$

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

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

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

, $A', B' > 0$

sogar: $\geq B' \cdot n$

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

$$\geq A' \cdot \frac{1}{2}\underbrace{(n-1)}_{\geq n/2} \cdot n + \underbrace{B'(n-1)}_{\geq 0}$$

$$\geq \underbrace{A' \cdot \frac{1}{4}}_{=: C_2} n^2 = C_2 \cdot n^2$$ ∎

Laufzeitanalyse MinSort 3/4

UNI FREIBURG

- Quadratische Laufzeit
  - Es gibt Konstanten C_1 und C_2 , so dass gilt

    $C_1 \cdot n^2 \leq T(n) \leq C_2 \cdot n^2$
  - Dann gilt insbesondere

    Doppelt so große Eingabe → viermal so große Laufzeit

$$\underbrace{C \cdot (2n)^2}_{T(2n)} = 4 \cdot \underbrace{C \cdot n^2}_{T(n)}$$

Laufzeitanalyse MinSort 4/4

UNI FREIBURG

- Quadratische Laufzeit, Rechenbeispiel
  - Unabhängig von den Konstanten wird das schnell sehr teuer
  - Annahme: $C = 1$ ns (1 einfache Anweisung ≈ 1 Nanosekunde)
  - Beispiel 1: $n = 10^6$ (1 Millionen Zahlen = 4 MB, 4 Bytes/Zahl)

    … dann $C \cdot n^2 = 10^{-9} \cdot 10^{12}$ s $= 10^3$ s = 16.7 Minuten
  - Beispiel 2: $n = 10^9$ (1 Milliarde Zahlen = 4 GB)

    … dann $C \cdot n^2 = 10^{-9} \cdot 10^{18}$ s $= 10^9$ s = 31.7 Jahre

Quadratische Laufzeit = "große" Probleme unlösbar

Laufzeitanalyse QuickSort 1/6

UNI FREIBURG

- Vorbetrachtung
  - Bei QuickSort hängt die Laufzeit stark davon ab, wie gut die Pivot-Elemente gewählt wurden

    Siehe Schaubilder vom Ü1
  - In solchen Fällen macht man die Analyse oft für

    den schlechtesten Fall (englisch: "worst case")

    den besten Fall (englisch: "best case")

    den typischen Fall (englisch: "average case")

    Heute machen wir "worst case" und "best case" … morgen dann auch den (komplizierteren) "average case"

Laufzeitanalyse QuickSort 2/6

UNI FREIBURG

- Analyse des "worst case"
  - Im schlechtesten Fall teilt jeder Pivot sein Feld der Größe m in zwei Teilfelder der Größen 1 und $m - 1$
  - Dann gibt es eine Konstante $A > 0$ so dass gilt:

    Für $n \geq 2$: $T(n) \geq T(n-1) + A \cdot n$ (*)

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

eigentlich Θ, weil wir den Rest nicht mit sortieren müssen

$\frac{1}{2} n(n+1) \geq \frac{n^2}{2}$

$T(n) \overset{(*)}{\geq} T(n-1) + A \cdot n$

$\overset{(*)}{\geq} T(n-2) + A \cdot (n-1)$

$\overset{(*)}{\geq} T(n-3) + A \cdot (n-2)$

usw.

Im "worst case" ist also auch QuickSort "quadratisch"

Laufzeitanalyse QuickSort 3/6

UNI FREIBURG

- Analyse des "best case"

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

  – Annahme: n ist eine Zweierpotenz, und jeder Pivot teilt das entsprechende Feld **genau** in der Mitte:

  genauer: $\leq n/2$

  Rekursionstiefe 1: 2 Teilfelder der Größe n/2

  Rekursionstiefe 2: 4 Teilfelder der Größe n/4

  Rekursionstiefe 3: 8 Teilfelder der Größe n/8 usw.

  – Dann gibt es eine Konstante $A > 0$ so dass gilt:

  für das QS Divide

  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: $T(n) \leq A \cdot n \cdot (1 + \log_2 n)$

Laufzeitanalyse QuickSort 4/6

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

- Analyse des "best case" ... Beweis $T(n) \leq A \cdot n \cdot (1 + \log_2 n)$

$$
\begin{aligned}
T(n) &\overset{(*)}{\leq} 2 \cdot T(n/2) + A \cdot n \\
&\overset{(*)}{\leq} 2 \cdot [\, 2 \cdot T(n/4) + A \cdot n/2 \,] + A \cdot n \\
&= 4 \cdot T(n/4) + 2 \cdot A \cdot n \\
&\overset{(*)}{\leq} 4 \cdot [\, 2 \cdot T(n/8) + A \cdot n/4 \,] + 2 \cdot A \cdot n \\
&= 8 \cdot T(n/8) + 3 \cdot A \cdot n \\
&\vdots \\
&\leq 2^k \cdot T(n/2^k) + k \cdot A \cdot n \\
&\leq n \cdot \underbrace{T(1)}_{\leq A} + \log_2 n \cdot A \cdot n \\
&\leq A \cdot n \cdot (1 + \log_2 n) \qquad \blacksquare
\end{aligned}
$$

$n = 2^k$

$k = \log_2 n$

z.B. $10 = \log_2 1024$

weil $2^{10} = 1024$

Laufzeitanalyse QuickSort 5/6

UNI FREIBURG

- Laufzeit $n \cdot \log n$
  - Es gibt Konstanten C_1 und C_2 , so dass gilt

    $C_1 \cdot n \cdot \log_2 n \leq T(n) \leq C_2 \cdot n \cdot \log_2 n$ für $n \geq 2$
  - Dann gilt insbesondere

    Doppelt so große Eingabe → etwas mehr als doppelt so lange

$\underbrace{C \cdot (2n) \cdot \log_2 (2n)}_{T(2n)} = 2 \cdot C \cdot n \cdot \underbrace{(1 + \log_2 n)}_{\text{minstens größer als } \log_2 n}$

$2 \cdot \underbrace{C \cdot n \cdot \log_2 n}_{T(n)}$

Laufzeitanalyse QuickSort 6/6

UNI FREIBURG

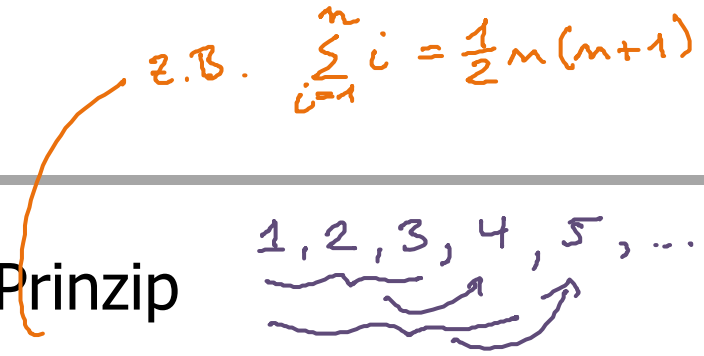
- Laufzeit $n \cdot \log n$, Rechenbeispiel
  - Annahme: $C = 1$ ns (1 einfache Anweisung ≈ 1 Nanosekunde)
  - Beispiel 1: $n = 2^{20}$ (≈ 1 Millionen Zahlen = 4 MB)

    … dann $C \cdot n \cdot \log_2 n = 10^{-9} \cdot 2^{20} \cdot 20$ s = 21 Millisekunden
  - Beispiel 2: $n = 2^{30}$ (≈ 1 Milliarde Zahlen = 4 GB)

    … dann $C \cdot n \cdot \log_2 n = 10^{-9} \cdot 2^{30} \cdot 30$ s = 32 Sekunden

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

Auffrischung 1/3



- Vollständige Induktion, Prinzip
  - Man möchte beweisen, dass eine Aussage für alle natürlichen Zahlen gilt, also: A(n) gilt für alle n ∈ **N**
  - **Induktionsanfang**: Wir zeigen, dass A(1), ..., A(k) gelten

    Meistens reicht k = 1, aber manchmal auch mehr
  - **Induktionsschritt:** Wir nehmen für ein beliebiges n > k an, dass A(1), ..., A(n-1) gelten, und zeigen: dann gilt auch A(n)

    Meistens reicht A(n-1), aber manchmal 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

Auffrischung 2/3

- Vollständige Induktion, Beispiel
  - Wir haben vorhin benutzt: $\Sigma_{i=1..n}\, i = \frac{1}{2} \cdot n \cdot (n+1)$

Induktionsanfang

IA: $n = 1 \quad 1 = \sum_{i=1}^{1} i = \frac{1}{2} \cdot 1 \underbrace{(1+1)}_{2} = 1$ ∎

Induktionsschritt

IS: $1, \dots, n \to n+1$

Jetzt dürfen wir voraussetzen (IV), dass die Aussage gilt für $1, \dots, n$

$$\sum_{i=1}^{n+1} i = \underbrace{\sum_{i=1}^{n} i}_{= \frac{1}{2} n(n+1) \text{ nach IV}} + n+1 = \frac{1}{2} n(n+1) + n+1$$

$$= \underbrace{\left(\frac{1}{2} n + 1\right)}_{= \frac{1}{2}(n+2)} (n+1) = \frac{1}{2}(n+1)(n+2)$$ ∎

Auffrischung 3/3

UNI FREIBURG

- Der Logarithmus (≠ Algorithmus)
  - Der "Logarithmus zur Basis b" ist gerade die inverse Funktion zu "b hoch"

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

    Beispiel: $\log_2 1024 = 10 \Leftrightarrow 2^{10} = 1024$
  - Die Rechenregeln ergeben sich dann einfach aus den Rechenregeln für das Potenzieren
  - Beispiel: $\underbrace{\log_b (x \cdot y)}_{z} = \underbrace{(\log_b x)}_{z_1} + \underbrace{(\log_b y)}_{z_2}$

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

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

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

$\Rightarrow$

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

$\Rightarrow z = z_1 + z_2$ ∎