

Effiziente Algorithmen

gelesen von

Dr. Michael Brinkmeier

auf Basis der Vorlesung von
Prof. Dr. Manfred Kunde

Technische Universität Ilmenau
Fakultät Informatik und Automatisierung
Fachgebiet Automaten und Formale Sprachen

8.7.2005

Inhalte der Vorlesung

- Grundlagen der Algorithmenanalyse
 - ▶ Laufzeit und Speicher
- Sortieralgorithmen
 - ▶ Algorithmen und Analyse
 - ▶ Untere Schranken
 - ▶ Strukturierte Schlüssel
- Mengenoperationen
 - ▶ Union-Find-Strukturen
 - ▶ Prioritätswarteschlangen
- Graphalgorithmen
 - ▶ Datenstrukturen
 - ▶ Grundlegende Algorithmen (Suche, Spannbäume, Transitive Hülle)
 - ▶ Kürzeste Wege
- Matrizenmultiplikation

Scheinerwerb

1 3V

- ▶ mind. 13 (von 23) Anwesenheiten in den Vorlesungen
- ▶ Erfolgreiche Teilnahme an der abschließenden Klausur

2 1Ü

- ▶ mind. 3 (von 6) Anwesenheiten in den Übungen
- ▶ Erfolgreiche Teilnahme am Übungsprogramm
 - ★ Insgesamt 6 Blätter in 3 Blöcken zu je zwei Blättern
 - ★ 4 erfolgreich bearbeitete Aufgaben in 3 verschiedenen Blöcken

Dringende Empfehlung: Nehmen Sie an den Übungen aktiv teil!

Teil 1: Einführung

Grundlagen der Algorithmenanalyse

Herkunft des Begriffes *Algorithmus*

Auszug aus [wikipedia.de](#)

[...]

Das Wort **Algorithmus** ist eine Abwandlung oder Verballhornung des Namens **Muhammad ibn Musa al-Chwarizmi** (* ca. 783, ca. 850), der Autor des Buchs **Hisab al-dschabr wa-l-muqabala (825, Regeln zur Wiederherstellung und Reduktion)**, durch das die Algebra im Westen verbreitet wurde. Die lateinische Fassung beginnt mit **Dixit Algoritmi... (Algorithmus sprach...)**, womit der Autor gemeint war. Das Wort Algebra stammt ebenfalls (al-Jabr Einrenkung) aus dem Titel des Buches. Ursprünglich stand das Wort **Algorism** nur für die Regeln zur Arithmetik mit arabischen Ziffern. [...]

Der moderne Algorithmusbegriff

Algorithmus

Ein **Algorithmus** ist eine Handlungs-/Rechenvorschrift zur Lösung eines Problems.

Durch diese Definition wird der Begriff Algorithmus mittels zweier intuitiver Begriffe definiert:

- 1 Handlungs-/Rechenvorschrift
- 2 Problem

Beide Begriffe hängen dabei wesentlich vom betrachteten Kontext ab. Dazu gehören:

- 1 die Möglichkeiten der handelnden Instanz
- 2 die Spezifikation eines Problems

Spezifikation von Problemen

Ein **Problem** umfasst üblicherweise

- eine (möglicherweise unendliche) Menge von Fällen, die üblicherweise durch bestimmte **Parameter** beschrieben werden.
- eine Fragestellung, die von den Parametern abhängt.

Im Kontext von Computer-Programmen verwenden wir häufig die folgenden Bezeichnungen:

- **Eingabe**: Die Belegung der Parameter, d.h. der konkrete Fall.
- **Ausgabe**: Die Antwort auf die Fragestellung.

Spezifikation von Problemen

Wir stellen Probleme häufig in der folgenden Form dar.

PROBLEM

Eingabe: Eine Liste von Parametern.
Eine Fragestellung.

Beispiele sind:

MULT

Eingabe: $a, b \in \mathbb{Z}$
Welchen Wert hat das Produkt $a \cdot b$?

PRIM

Eingabe: $a \in \mathbb{N}$
Ist a eine Primzahl?

Rechenmodelle

Um die Lösungsvorschrift für ein Problem beschreiben zu können, benötigen wir einen Kontext, der vorgibt, wie wir mit den Parametern der Eingabe und eventuell benötigten anderen **Variablen** arbeiten.

Diesen Kontext liefern uns **Rechenmodelle**, z.B.:

- klassische Arithmetik, Algebra und Logik
- (Nicht-)Deterministische Endliche Automaten
- Turing-Maschinen
- Random Access Maschinen
- Quantencomputer
- ...

Diese Rechenmodelle ermöglichen die Manipulation von Werten mittels bestimmter Operationen und erlauben somit das Lösen bestimmter Probleme.

Welche Probleme lösbar sind, hängt dabei wesentlich vom verwendeten Rechenmodell ab.

Effizienz

Frage

Wie misst man die **Effizienz** eines Algorithmus?

Üblicherweise misst man die von dem Algorithmus für die Lösung des Problems benötigten Ressourcen. Dies sind insbesondere

- die verwendete Zeit und
- der verwendete Platz.

Frage

Wie misst man die benötigte Zeit und den benötigten Platz?

Die Antwort auf diese Frage hängt wieder vom verwendeten Rechenmodell und einem dazugehörigen **Kostenmodell** ab.

Verhalten im Schlechtesten Fall

Um einen Algorithmus A möglichst detailliert bewerten zu können, könnte man einfach für jede mögliche Eingabe x die Zeit $T_A(x)$, die A benötigt um das Problem zu lösen, messen.

Dies ist im Allgemeinen jedoch nicht praktikabel. Daher beschränkt man sich üblicherweise auf **das Verhalten im schlechtesten Fall**.

Dazu weist man jeder Eingabe x eine **Größe** $|x| \in \mathbb{N}$ zu, und betrachtet die maximale Zeit, die der Algorithmus zum Lösen des Problems auf allen Eingaben derselben Größe benötigt.

Definition (Worst-Case-Laufzeit)

$$T_A(n) := \sup \{ T_A(x) \mid x \in \text{PROBLEM}, |x| = n \}$$

Verhalten im Mittel

In vielen Fällen stellt das Verhalten im schlechtesten Fall nur ein ungenaues Maß dar. So kann z.B. die Lösung eines bestimmten Falles sehr lange brauchen, während alle anderen Fälle in sehr viel kürzerer Zeit gelöst werden können.

In solchen Fällen kann man das **Verhalten im Mittel** betrachten.

Dazu nimmt man an, dass die Eingabe x mit einer gewissen **Wahrscheinlichkeit** $0 < p(x) < 1$ auftritt.

Definition (Expected-Time)

$$T_A^{\text{avg}}(n) := E(\{T_A(x) \mid x \in \text{PROBLEM}, |x| = n\}) = \sum_{\substack{x \in \text{PROBLEM} \\ |x| = n}} p(x) T_A(x).$$

Asymptotische Notationen (Wiederholung)

Definition

Seien $f, g : \mathbb{N} \rightarrow \mathbb{N}$ Abbildungen.

- $O(f) := \{g \mid \exists c > 0 \exists n_0 \forall n \geq n_0 : g(n) \leq cf(n)\}$
- $\Omega(f) := \{g \mid \exists c > 0 \exists n_0 \forall n \geq n_0 : g(n) \geq cf(n)\}$
- $\Theta(f) := O(f) \cap \Omega(f)$

Die Sprech- und Schreibweise im Zusammenhang mit den asymptotischen Notationen ist häufig etwas lax, z.B.

$$n^2 + 5n = n^2 + O(n) = O(n^2)$$

statt

$$n^2 + 5n \in n^2 + O(n) \subseteq O(n^2).$$

Asymptotische Notationen (Wiederholung)

Frage

Seien A_1 und A_2 zwei Algorithmen für das gleiche Problem. Wann ist A_1 schneller als A_2 ?

A_1 ist **immer schneller** als A_2 , wenn $T_{A_1}(x) \leq T_{A_2}(x)$ für jede Eingabe x .

A_1 ist im **schlechtesten Fall schneller** als A_2 , wenn $T_{A_1}(n) \leq T_{A_2}(n)$ für alle $n \in \mathbb{N}$.

A_1 ist **asymptotisch schneller** als A_2 , wenn

$$\lim_{n \rightarrow \infty} \frac{T_{A_1}(n)}{T_{A_2}(n)} = 0.$$

Definition

$$o(f) := \left\{ g \mid \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \right\}.$$

Asymptotische Notationen (Wiederholung)

Ein asymptotisch schnellerer Algorithmus benötigt für kleinere Problemgrößen unter Umständen länger, als der „langsamere“ Algorithmus.

Komplexität von Problemen - Obere Schranken

Komplexität von Problemen

Ein Problem **PROB** hat die Zeitkomplexität $O(f)$ für $f: \mathbb{N} \rightarrow \mathbb{N}$, wenn es einen Algorithmus A für **PROB** gibt mit

$$T_A(n) \in O(f).$$

Eine solche **obere Schranke** für die Zeitkomplexität eines Problems kann somit durch die Analyse eines Algorithmus erreicht werden.

Häufig ergeben sich recht einfache obere Schranken, die aber sehr ungenau sind.

Komplexität von Problemen - Untere Schranken

Um die Komplexität genauer abschätzen zu können, ist es wünschenswert auch eine untere Schranke angeben zu können.

Komplexität von Problemen

Ein Problem PROB hat die Zeitkomplexität $\Omega(f)$ für $f: \mathbb{N} \rightarrow \mathbb{N}$, wenn für **jeden** Algorithmus A für PROB gilt:

$$T_A(n) \in \Omega(f).$$

Offensichtlich ist es schwieriger eine untere Schranke zu finden, da man eine Aussage über alle möglichen Algorithmen für das Problem machen muss.