WS 2006/07

Effiziente Algorithmen und Datenstrukturen

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2006WS/ea/

Wintersemester 2006/07

1. Ziel der Vorlesung

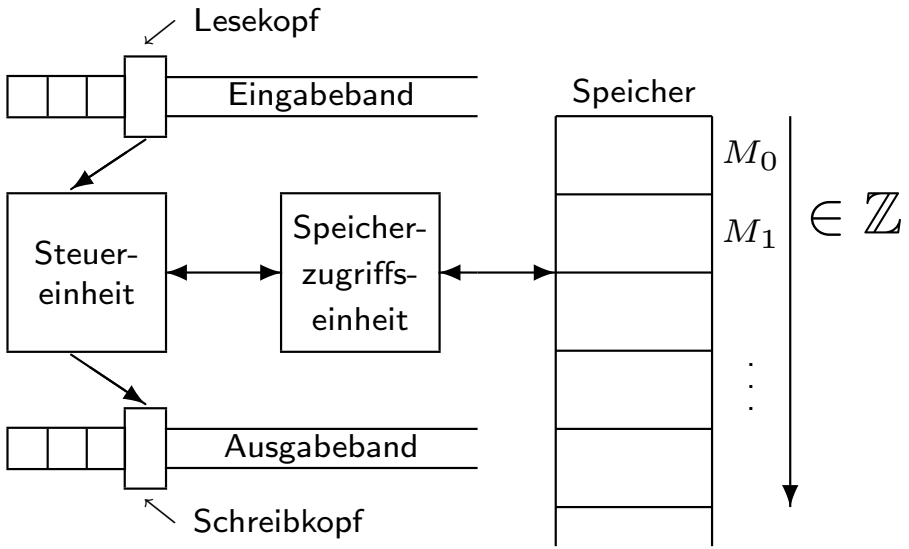
Der Zweck der Vorlesung ist das Studium fundamentaler Konzepte in der Algorithmentheorie. Es werden relevante Maschinenmodelle, grundlegende und höhere Datenstrukturen sowie der Entwurf und die Analyse sequentieller Algorithmen besprochen. Dabei wird eine Reihe verschiedener Analysemethoden (für entsprechend unterschiedliche Anforderungen) eingeführt.

Die betrachteten Problemklassen umfassen eine umfangreiche Auswahl der in der Praxis relevanten kombinatorischen Probleme, wobei die algorithmischen Ansätze sich in dieser Vorlesung jedoch praktisch auf deterministische, sequentielle, exakte Algorithmen beschränken.

Für weiterführende Konzepte und Ansätze (z.B. probabilistische, parallele, approximative Algorithmen) wird auf entsprechende Vorlesungen verwiesen.

2. Maschinenmodelle

- Turingmaschine (TM)
- Registermaschine (RAM)
- Boolesche Schaltkreise
- (Quantencomputer)
- (DNA-Computer)
- ...



Registermaschine

3. Komplexitätsmaße

Ein Problem ist formal eine Sprache $L \subseteq \Sigma^*$. $x \in \Sigma^*$ heißt eine Probleminstanz für L, wenn wir untersuchen wollen, ob $x \in L$.

Sei M eine (Turing- oder) Registermaschine.

- M entscheidet L, falls für alle $x \in \Sigma^*$ M nach endlicher Zeit hält mit

$$\begin{cases} \text{Antwort „ja“, falls } x \in L \\ \text{Antwort „nein“, falls } x \notin L \end{cases}$$

- M akzeptiert L, falls für alle $x \in \Sigma^*$ gilt

$$\begin{cases} \text{falls } x \in L : M \text{ hält mit Antwort “ja“} \\ \text{falls } x \notin L : M \text{ hält mit Antwort “nein“ oder } M \text{ hält nicht.} \end{cases}$$

- Berechnung von Funktionen:
  Seien Σ, Γ Alphabete. Eine TM (bzw. RAM) M berechnet eine (partielle) Funktion $f : \Sigma^* \to \Gamma^*$ gdw. für alle x im Definitionsbereich von f gilt:
  bei Eingabe x hält M nach endlich vielen Schritten, und zwar mit Ausgabe $f(x)$.

Wir berechnen die Komplexität eines Problems in Abhängigkeit von der Länge der Eingabe:
Eingaben $x \in \Sigma^n$ haben Länge n.

Insbesondere bei Funktionen oder Problemen, deren Eingabe als „als aus n Argumenten bestehend“ interpretiert werden kann, betrachten wir oft auch die uniforme Eingabelänge n.

Beispiel 1

Sollen n Schlüssel $\in \Sigma^*$ (vergleichsbasiert) sortiert werden, so nehmen wir als Eingabelänge gewöhnlich n, die Anzahl der Schlüssel, und nicht ihre Gesamtlänge.

Komplexitätsressourcen:

Man betrachtet u.a.

- Rechenzeit
- Speicherplatz
- Anzahl von Vergleichen
- Anzahl von Multiplikationen
- Schaltkreisgröße
- Programmgröße
- Schachtelungstiefe von Laufschleifen

Komplexität der Ressourceneinheiten:

Wir unterscheiden

- uniformes Kostenmodell: Die Kosten jeder Ressourceneinheit sind 1.
- logarithmisches Kostenmodell: Die Kosten eines Rechenschritts sind durch die Länge der Operanden bestimmt:

  1. Der Zeitbedarf eines Rechenschritts ist gleich der größten Länge eines Operanden des Rechenschritts.
  2. Der Platzbedarf einer Speicherzelle ist gleich der größten Länge eines darin gespeicherten Wertes.

Wir unterscheiden verschiedene Arten der Komplexität:

- worst-case Komplexität:

$$C_{\text{wc}}(n) := \max\{C(x);\ |x| = n\}$$

- durchschnittliche Komplexität (average complexity):

$$C_{\text{avg}}(n) := \frac{1}{|\Sigma^n|} \sum_{|x|=n} C(x)$$

allgemeiner: Wahrscheinlichkeitsmaß μ

$$C_{\text{avg}}(n) := \sum_{x \in \Sigma^n} \mu(x) \cdot C(x)$$

Wir unterscheiden verschiedene Arten der Komplexität:

- amortisierte Komplexität:
  durchschnittliche Kosten der Operationen in Folgen der Länge n
  worst-case über alle Folgen der Länge n von Operationen
- probabilistische oder randomisierte Komplexität:
  Algorithmus hat Zufallsbits zur Verfügung. Erwartete Laufzeit (über alle Zufallsfolgen) für feste Eingabe x, dann worst-case für alle $|x| = n$.

Beispiel 2

$r := 2$
for $i := 1$ **to** n **do** $r := r^2$ **od**
co `das Ergebnis ist` 2^{2^n} **oc**

- Zeitbedarf:
  - uniform: n Schritte
  - logarithmisch: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- Platzbedarf:
  - uniform: $\mathcal{O}(1)$
  - logarithmisch: 2^n

4. Wachstumsverhalten von Funktionen

f, g seien Funktionen von $\mathbb{N}_0$ nach $\mathbb{R}_+$.

- $g = \mathcal{O}(f)$ [auch: $g(n) = \mathcal{O}(f(n)$ oder $g \in \mathcal{O}(f)$] gdw.

$$(\exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0)[g(n) \leq c \cdot f(n)]$$

- $g = \Omega(f)$ [auch: $g(n) = \Omega(f(n)$ oder $g \in \Omega(f)$] gdw.

$$(\exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0)[g(n) \geq c \cdot f(n)]$$

- $g = \Theta(f)$ gdw. $g = \mathcal{O}(f)$ und $g = \Omega(f)$

4. Wachstumsverhalten von Funktionen

f, g seien Funktionen von $\mathbb{N}_0$ nach $\mathbb{R}_+$.

- $g = o(f)$ gdw.

$$(\forall c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0)[g(n) \leq c \cdot f(n)]$$

- $g = \omega(f)$ gdw.

$$(\forall c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0)[g(n) \geq c \cdot f(n)]$$

- $g = \Omega_\infty(f)$ gdw.

$$(\exists c > 0)[g(n) \geq c \cdot f(n) \text{ für unendlich viele } n]$$

- $g = \omega_\infty(f)$ gdw.

$$(\forall c > 0)[g(n) \geq c \cdot f(n) \text{ für unendlich viele } n]$$

Beispiel 3

$$n^3 \text{ ist } \underline{\text{nicht}} \ \mathcal{O}\left(\frac{n^3}{\log n}\right).$$

	Size n					
Growth Rates	10	20	30	40	50	60
n	.00001 seconds	.00002 seconds	.00003 seconds	.00004 seconds	.00005 seconds	.00006 seconds
n^2	.0001 seconds	.0004 seconds	.0009 seconds	.0016 seconds	.0025 seconds	.0036 seconds
n^3	.001 seconds	.008 seconds	.027 seconds	.064 seconds	.125 seconds	.216 seconds
n^5	.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 seconds	1.0 seconds	17.9 minutes	12.7 days	35.7 years	366 centuries
5^n	.059 seconds	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Beispiel 4 (Mergesort)

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + cn \\
&= cn + 2T\left(\frac{n}{2}\right) \\
&= cn + 2\left(c\frac{n}{2} + 2T\left(\frac{n}{4}\right)\right) \\
&= cn + cn + 4T\left(\frac{n}{4}\right) \\
&\approx cn \log_2 n \text{ (nur genau für Zweierpotenzen)}
\end{aligned}
$$

Methoden zur Lösung von Rekursionsgleichungen

1. Multiplikatorenmethode
2. Lineare homogene Rekursionsgleichungen können mit Hilfe des charakteristischen Polynoms gelöst werden
3. Umwandlung inhomogener Rekursionsgleichungen in homogene
4. Erzeugendenfunktionen
5. Transformation des Definitions- bzw. Wertebereichs
6. ...

Es gibt keinen vollständigen Satz von Methoden.