Komplexitätstheorie

(3 V + 2 Ü)

Nicole Schweikardt
Goethe-Universität Frankfurt am Main

Webseite:

www.tks.informatik.uni-frankfurt.de/teaching/komp

Literatur:

[AB] Sanjeev Arora, Boaz Barak:
Computational Complexity: A Modern Approach.
Cambridge University Press, 2009

Kapitel 0: Einleitung und grundlegende Notationen

0.0 Einleitung

Ziel: Charakterisierung des Schwierigkeitsgrads bestimmter Probleme.

Insbes: Berechnungskomplexität — d.h.: Wie schwer ist es, die Lösung eines vorgegebenen Problems zu berechnen?

Beispiele für Berechnungsprobleme:

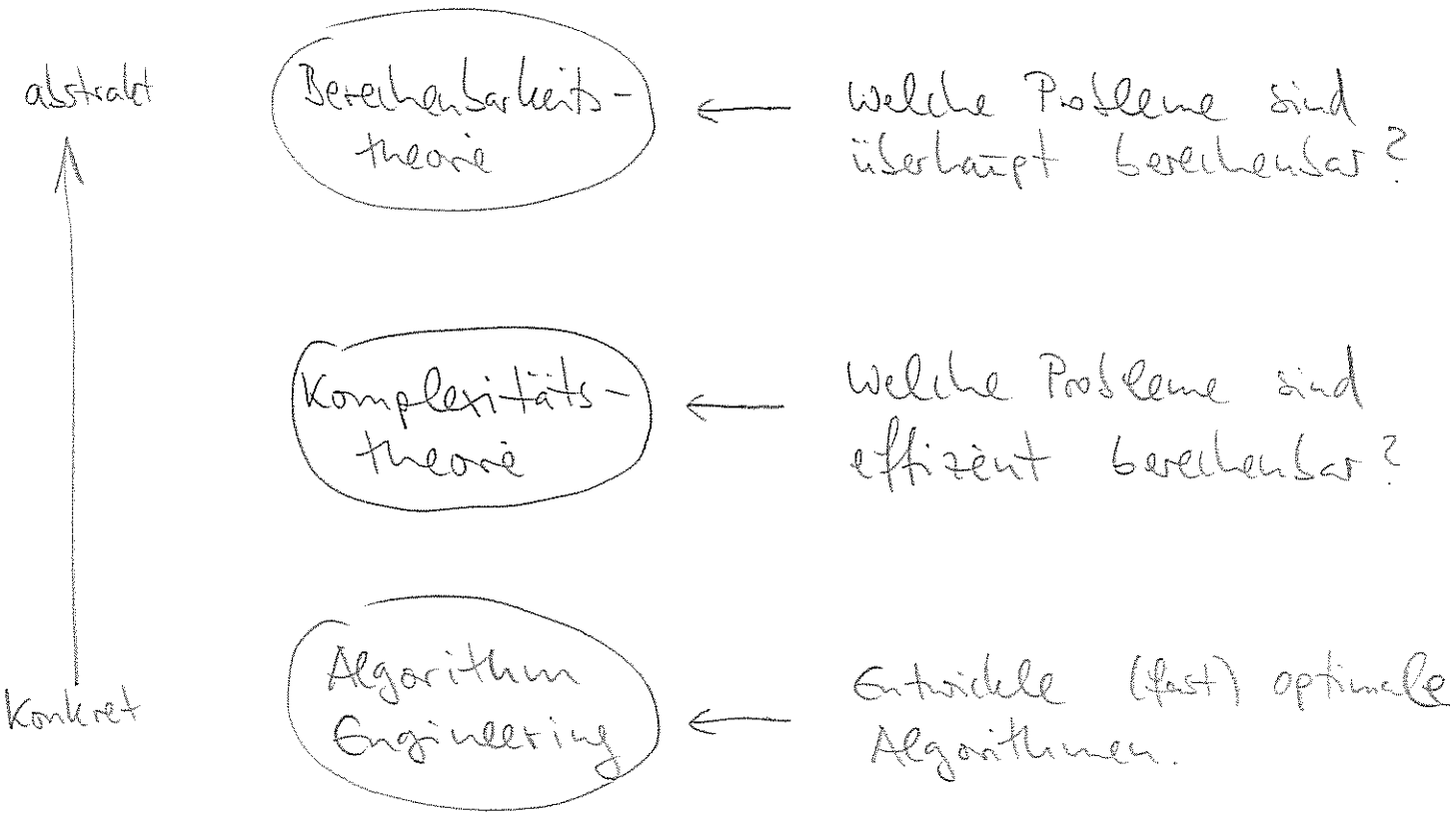
(i) Gegeben zwei natürliche Zahlen, berechne deren Produkt.

(ii) Gegeben ein lineares Gleichungssystem bestehend aus n linearen Gleichungen über n Variablen, finde eine Lösung des linearen Gleichungssystems (falls es eine gibt).

(iii) Gegeben eine Liste von Personen und eine Liste von Paaren $\{x,y\}$ von Personen, die nicht miteinander auskommen, finde eine möglichst große Menge der Personen, die Du zu einer Feier einladen kannst, so dass alle geladenen Gäste miteinander auskommen.

Forschungsgebiete:

Fragestellung / Ziel:



Beispiele für effiziente Berechnungen.

(i) Multiplikation zweier natürlicher Zahlen a und b:

Algorithmus 1: "Wiederholtes Addieren":

```
p := 0
for i := 1 to b do
    p := p + a
end
```

Algorithmus 2: "Grundschul-Multiplikation"

Bsp: $577 \cdot 423$

$$
\begin{array}{r}
1731 \\
1154 \\
2308 \\
\hline
244071
\end{array}
$$

Frage: Was ist effizienter?

Idee:
- Algo 1 benötigt 423 Additionsschritte für Berechnung von $577 \cdot 423$,
- Algo 2 benötigt 3 Multiplikationen der Zahl 577 mit einer Ziffer und zusätzlich die Addition von 3 Zahlen

Allgemein: Für Multiplikation zweier n-stelliger Zahlen (d.h. Zahlen zwischen 10^{n-1} und 10^n) benötigt
- Algo 1 mindestens 10^{n-1} Additionsschritte
- Algo 2 höchstens n Multiplikationsschritte und n Additionsschritte.

Jeder Additions- und Multiplikationsschritt benötigt $O(n)$ Elementarschritte

$\Rightarrow$ Bereits für 20-stellige Zahlen liefert Algo 2, ausgeführt von einem Viertklässler, schneller ein Ergebnis als Algo 1, ausgeführt auf einem Supercomputer.

Bemerkung: Es ist ein noch schnellerer Algorithmus zur Multiplikation bekannt, der auf der schnellen Fouriertransformation beruht und in $O(n \cdot \log n \cdot \log\log n)$ Elementarschritten zum Ziel führt; siehe Kapitel 16 von [AB].

(ii): Lösung eines linearen Gleichungssystems aus n Gleichungen über n Variablen:

Algo 1: Gauß-Elimination

$\rightsquigarrow$ $O(n^3)$ arithmetische Operationen

Algo 2: Strassens Algorithmus (1960er Jahre)

$\rightsquigarrow$ $O(n^{2,81})$ Operationen

... Noch effizientere Algorithmen bekannt.

(iii): "konfliktfreie Gästeliste":

Naiver Algorithmus:

~~For~~ $i := n$ to 1:

Betrachte jede i-elementige Teilmenge der gegebenen n-elementigen Liste von Personen und teste, ob diese Teilmenge ein Paar $\{x, y\}$ enthält, das nicht miteinander auskommt.

Falls nein: STOP; die gerade betrachtete Teilmenge ist eine größtmögliche konfliktfreie Gästeliste.

Anzahl Schritte im Worst-Case: mehr als 2^n.

Bemerkung: Bis heute ist kein wesentlich effizienteres Algorithmus für Lösung dieses Problems bekannt. Mehr dazu in Kapitel 2 bzgl des "Independent Set" - Problems, das NP-vollständig ist

Eins der Ziele der Komplexitätstheorie:

Nachweis unterer Schranken für die Ressourcen, die für Lösung eines Problems prinzipiell benötigt werden.

Schön wäre z.B., wenn wir beweisen könnten, dass es keinen Algorithmus geben kann, der das "Konfliktfreie Gästeliste" - Problem löst und dabei weniger als $2^{n/10}$ Schritte durchführt.

0.1 Grundlegende Notationen und Konventionen

- $\mathbb{N} := \{0, 1, 2, \dots\}$ Menge der natürlichen Zahlen
- Zahlen, die durch Buchstaben i, j, k, ℓ, m, n bezeichnet werden, sind stets ganze Zahlen
- Für $n \geq 1$ ist $[n] := \{1, \dots, n\}$
- Für $x \in \mathbb{R}$ ist
  - $\lceil x \rceil$ das kleinste $z \in \mathbb{Z}$ mit $x \leq z$
  - $\lfloor x \rfloor$ das größte $z \in \mathbb{Z}$ mit $z \leq x$
- Wann immer wir eine reelle Zahl x in einem Kontext benutzen, in dem eigentlich eine ganze Zahl erwartet wird, ist $\lceil x \rceil$ gemeint.
- $\log x := \log_2 x$
- Wir sagen
  "eine Bedingung $P(n)$ gilt für hinreichend große n",
  falls es eine Zahl N gibt, so dass $P(n)$ für alle $n > N$ gilt
  (Bsp: $2^n > 100n^2$ gilt für hinreichend große n)
- Falls u ein Wort (oder Vektor) ist, so bezeichnet u_i das i-te Symbol von u (oder die i-te Koordinate von u)
  (Bsp: Für $u = aabab$ gilt $u_3 = b$, $u_4 = a$)

- Ist S ein Alphabet (d.h. eine Menge), so bezeichnet S^n die Menge aller Worte der Länge n über S (für $n \in \mathbb{N}$).
- $S^* = \bigcup_{n \in \mathbb{N}} S^n$
- $x \in S^*$, $k \in \mathbb{N}$ $\Rightarrow$ $x^k := \underbrace{x \circ \dots \circ x}_{k\text{-mal } x \text{ hintereinander-geschrieben}}$
- $x \in S^*$ $\Rightarrow$ $|x|$ bezeichnet die Länge von x

  Für $a \in S$ bezeichnet $|x|_a$ die Anzahl der Vorkommen des Buchstabens a in x

  (Bsp: Für $x = aabab$ ist $|x|_a = 3$).

Repräsentation von Objekten durch Worte

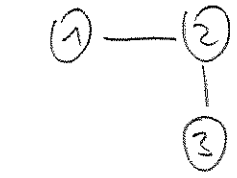
Die Berechnungsaufgaben, die wir in der Veranstaltung "Komplexitätstheorie" betrachten, betreffen zumeist Funktionen, deren Ein- und Ausgabe Bitstrings sind, d.h. Funktionen $f: \{0,1\}^* \to \{0,1\}^*$.

Dies ist keine wirkliche Einschränkung, da wir allgemeine Objekte leicht durch Bitstrings repräsentieren können:

z.B. repräsentieren wir natürliche Zahlen durch deren Binärdarstellung (z.B. 18 durch 10010).

Einen Graph G auf der Knotenmenge $[n] = \{1,\dots,n\}$ repräsentieren wir durch seine Adjazenzmatrix A mit $A_{i,j} = 1 \iff \{i,j\} \in E(G)$ (f.a. $i,j \in [n]$), deren Zeilen wir hintereinander schreiben

Bsp: G:



A

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

Bitstring: 010 101 010

Meistens werden wir es vermeiden, uns um die "Low-Level"-Details der Repräsentation zu kümmern und schreiben kurz $\langle x \rangle$, um eine kanonische (und nicht näher spezifizierte) Binärrepräsentation eines Objekts x zu bezeichnen. Oft schreiben wir auch einfach x an Stelle des formal korrekten $\langle x \rangle$.

Beachte: Die Binärrepräsentation $\langle x \rangle$ hat nichts mit dem Symbol $\lfloor x \rfloor$ zum Abrunden einer reellen Zahl zu tun. — Wir verwenden hier die Notation aus [AB].

Repräsentation von Paaren und Tupeln

Wir schreiben $\langle x, y \rangle$, um das geordnete Paar bestehend aus x und y zu bezeichnen.

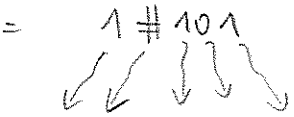
Eine kanonische Repräsentation von $\langle x, y \rangle$ durch einen Bitstring erhalten wir wie folgt:

Repräsentiere $\langle x, y \rangle$ durch das Wort $\lfloor x \rfloor \# \lfloor y \rfloor$ über dem Alphabet $\{0, 1, \#\}$

und ersetze dann jedes Symbol

0 durch 00
1 durch 11
durch 01

Bsp: $\langle 1, 5 \rangle \rightsquigarrow \lfloor 1 \rfloor \# \lfloor 5 \rfloor = 1\#101$



$11\,01\,11\,00\,11 = \lfloor \langle 1, 5 \rangle \rfloor$

Meistens schreiben wir einfach $\langle x, y \rangle$ an Stelle von $\lfloor \langle x, y \rangle \rfloor$

Auf die gleiche Weise können wir auch 3-Tupel $\langle x, y, z \rangle$ oder allgemein k-Tupel $\langle x_1, \ldots, x_k \rangle$ durch Bitstrings repräsentieren.

Berechnung von Funktionen, die nicht auf Bitstrings operieren

Funktionen f, deren Definitions- oder Wertebereich nicht $\{0,1\}^*$ ist, werden wir stillschweigend mit einer entsprechenden Funktion $g: \{0,1\}^* \to \{0,1\}^*$ identifizieren, so dass $f.a.$ x im Definitionsbereich von f gilt: $g(x) = f(x)$.

0.2 Entscheidungsprobleme und Sprachen

Wichtiger Spezialfall von Funktionen, die Bitstrings auf Bitstrings abbilden:

Boolesche Funktionen, d.h. $f: \{0,1\}^* \to \{0,1\}$.

Wir identifizieren eine Boolesche Funktion f mit der Menge

$$L_f := \{ x \in \{0,1\}^* : f(x) = 1 \}.$$

L_f wird auch Sprache (als Teilmenge von $\{0,1\}^*$) oder Entscheidungsproblem (gegeben $x \in \{0,1\}^*$, entscheide, ob $x \in L_f$ ist) genannt.

Wir identifizieren das Problem, $f: \{0,1\}^* \to \{0,1\}$ zu berechnen (d.h. bei Eingabe x den Wert $f(x)$ zu berechnen) mit dem Problem, die Sprache L_f zu entscheiden (d.h. bei Eingabe x zu entscheiden, ob $x \in L_f$ ist).

Beispiel 0.1:

Das "Konfliktfreie Gästeliste"-Problem können wir durch einen Graphen mit Knotenmenge $[n]$ repräsentieren, bei dem eine Kante zwischen zwei Personen i und j anzeigt, dass i und j nicht miteinander auskommen.
Ziel ist, eine möglichst große Menge unabhängiger Knoten zu finden, d.h. Knoten, zwischen denen keine Kante verläuft — ein sog. "independent set".

Das zugehörige Entscheidungsproblem ist

$$\text{INDSET} := \{ \langle G,k \rangle : \exists\, S \subseteq V(G) \text{ s.d. } |S| \geq k \text{ und } \forall u,v \in S \text{ ist } \{u,v\} \notin E(G) \}$$

Ein Algorithmus, der INDSET löst, sagt uns für gegebenes G und k, ob eine konfliktfreie Gästeliste der Größe $\geq k$ existiert. In Kapitel 2 werden wir sehen, dass man daraus auch tatsächlich eine entsprechende Gästeliste berechnen kann

0.3 Die $\mathcal{O}$-Notation

Definition 0.2:

Falls $f, g : \mathbb{N} \to \mathbb{N}$, so sagen wir

(1) $f = \mathcal{O}(g)$, falls es $c \in \mathbb{N}$ s.d.
$f(n) \leq c \cdot g(n)$ für hinreichend große n

(2) $f = \Omega(g)$, falls $g = \mathcal{O}(f)$

(3) $f = \Theta(g)$, falls $f = \mathcal{O}(g)$ und $f = \Omega(g)$

(4) $f = o(g)$, falls f.a. $\varepsilon > 0$ gilt.
$f(n) \leq \varepsilon \cdot g(n)$ für hinreichend große n
(d.h.: $\frac{f(n)}{g(n)} \xrightarrow{n \to \infty} 0$)

(5) $f = \omega(g)$, falls $g = o(f)$

Oft schreiben wir $f(n) = \mathcal{O}(g(n))$ an Stelle von $f = \mathcal{O}(g)$.

Beispiele 0.3:

(1) Für $f(n) = 100n \cdot \log n$ und $g(n) = n^2$ gilt:
$f = \mathcal{O}(g)$, $g = \Omega(f)$, $f = o(g)$, $g = \omega(f)$

(2) Für $f(n) = 100n^2 + 24n + 2\log_2 n$ gilt:

$f(n) = \Theta(n^2)$

(3) Für $f(n) = \min\{n, 10^6\}$ und $g(n) = 1$ $(\forall\, n \in \mathbb{N})$ gilt: $f = \Theta(g)$.

kurz: $f = \Theta(1)$.

(4) Für jedes $c \in \mathbb{N}$ gilt: $n^c = o(2^n)$.

Schreibweise: Wir schreiben $h(n) = n^{O(1)}$ (oder $h(n) = \text{poly}(n)$), um auszusagen, dass es ein $c \in \mathbb{N}$ gibt, so dass $h(n) \leq n^c$ für hinreichend große n gilt.

Übungsaufgaben:

A0.1

(a) Teile (a), (b), (e), (f) von [AB], Exercise 0.1

(b) Teile (a), (c), (d), (g) ------ 0.2

↑

u.a. $f: \mathbb{N} \to \mathbb{N}$ mit $f(0) = 10$ und, $\forall\, n \geq 1$:

(a) $f(n) = f(n-1) + 10$
$\rightsquigarrow f = \Theta(n)$

(b) $f(n) = f(n-1) + n$
$\rightsquigarrow f = \Theta(n^2)$

(c) $f(n) = f(\frac{n}{2}) + 10$
$\rightsquigarrow f = \Theta(\log n)$

A0.2

[AB], Exercise 0.3.