

Vorlesung
Grundlagen der Theoretischen Informatik /
Einführung in die Theoretische Informatik I

Bernhard Beckert

Institut für Informatik



Sommersemester 2007

Diese Vorlesungsmaterialien basieren ganz wesentlich auf den Folien zu den Vorlesungen von

Katrin Erk (gehalten an der Universität Koblenz-Landau)

Jürgen Dix (gehalten an der TU Clausthal)

Ihnen beiden gilt mein herzlicher Dank.

– Bernhard Beckert, April 2007

Inhalt von Teil VI

- Definition der berühmten Klassen **P** und **NP**.
- Begriff der **Reduktion**: ein Problem (eine Sprache) wird auf eine zweite reduziert. Das erste Problem ist dann höchstens so schwer wie das zweite.
- Den Begriff eines **NP-schweren** Problems.
- Einige Probleme der Graphentheorie: sie sind **NP-vollständig**.
- Die wichtigsten **Komplexitätsklassen** und ihre Struktur.

Teil VI

Komplexitätstheorie

- 1 **Die Struktur von PSPACE**
- 2 Vollständige und harte Probleme
- 3 Beispiele

- 1 **Sortieralgorithmen:** Bubble Sort, Quicksort, ...
- 2 **Erfüllbarkeitsproblem:**
Gibt es eine erfüllende Belegung für die Variablen $\{x_1, x_2, \dots, x_n\}$?
- 3 **Graphenprobleme:**
Gibt es einen hamiltonschen Kreis in einem Graphen?
Sind zwei Knoten in einem Graphen voneinander erreichbar?

Welche Arten von Komplexität gibt es?

- Zeit
- Speicher

Definition 16.1 (NTIME($T(n)$), DTIME($T(n)$))

Basismodell: k -DTM \mathcal{M} (ein Band für die Eingabe).

Wenn \mathcal{M} mit jedem Eingabewort der Länge n höchstens $T(n)$ Schritte macht, dann wird sie **$T(n)$ -zeitbeschränkt** genannt.

Die von \mathcal{M} akzeptierte Sprache hat **Zeitkomplexität $T(n)$** (tatsächlich meinen wir $\max(n+1, \lceil T(n) \rceil)$).

- **DTIME($T(n)$)** ist die Klasse der Sprachen, die von $T(n)$ -zeitbeschränkten, DTMn akzeptiert werden.
- **NTIME($T(n)$)** ist die Klasse der Sprachen, die von $T(n)$ -zeitbeschränkten, NTMn akzeptiert werden.

Definition 16.2 ($\text{NSPACE}(S(n))$, $\text{DSPACE}(S(n))$)

Basismodell: k -DTM \mathcal{M} davon ein spezielles Eingabeband (**offline DTM**).

Wenn \mathcal{M} für jedes Eingabewort der Länge n maximal $S(n)$ Zellen auf den Ablagebändern benutzt, dann heißt \mathcal{M} **$S(n)$ -speicherbeschränkt**.

Die von \mathcal{M} akzeptierte Sprache hat **Speicherkomplexität $S(n)$** (tatsächlich meinen wir $\max(1, \lceil S(n) \rceil)$)

- **DSPACE** $(S(n))$ ist die Klasse der Sprachen, die von $S(n)$ -speicherbeschränkten, DTMn akzeptiert werden.
- **NSPACE** $(S(n))$ ist die Klasse der Sprachen, die von $S(n)$ -speicherbeschränkten, NTMn akzeptiert werden.

Frage:

Wieso eine *offline*-Turing-Maschine?

Bandbeschränkung von weniger als linearem Wachstum.

Beispiel

Zu welcher Zeit-/Speicherkomplexitätsklasse gehört

$$\mathcal{L}_{\text{mirror}} := \{wcw^R : w \in (0+1)^*\},$$

also die Menge aller Wörter die um den mittleren Buchstaben c gespiegelt werden können?

Beispiel (Forts.)

Zeit: $\text{DTIME}(n+1)$. Die Eingabe wird einfach rechts vom c in umgekehrter Reihenfolge kopiert. Wenn das c gefunden wird, wird einfach der übrige Teil (das w) mit der Kopie von w auf dem Band verglichen.

Speicher: Die gerade beschriebene Maschine liefert eine Schranke von $\text{DSPACE}(n)$.

Geht es noch besser?

Ja: $\text{DSPACE}(\lg n)$. Wir benutzen zwei Bänder als Binär-Zähler. Die Eingabe auf das Auftreten von genau einem c benötigt keinen Speicher (kann mit einigen Zuständen getan werden). Als zweites prüfen wir Zeichen für Zeichen auf der linken und auf der rechten Seite: dazu müssen die zu prüfenden Positionen gespeichert werden (sie werden auf den beiden Bändern kodiert). Man kommt auch mit einem einzigen Zähler aus (und einem Band).

Wichtige Fragen (1)

Zeit: Wird jede Sprache aus $\mathbf{DTIME}(f(n))$ von einer DTM entschieden?

Speicher: Wird jede Sprache aus $\mathbf{DSPACE}(f(n))$ von einer DTM entschieden?

Die Funktionen f sind immer sehr einfache Funktionen, insbesondere sind sie alle berechenbar. In dieser Vorlesung betrachten wir nur Potenzen $f(n) = n^i$.

Wichtige Fragen (2)

Zeit/Speicher: Wie sieht es mit **NTIME**($f(n)$), **NSPACE**($f(n)$) aus?

Zeit vs. Speicher: Welche Beziehungen gelten zwischen **DTIME**($f(n)$), **DSPACE**($f(n)$), **NTIME**($f(n)$), **NSPACE**($f(n)$) ?

Halten, hängen nach n Schritten.

Zeitbeschränkt: Was bedeutet es, daß **eine DTM höchstens n Schritte macht?**

Halten?: Streng genommen, müßte sie dann entweder **halten** oder **hängen**. Das bedeutet, im ersten Fall, daß die Eingabe **akzeptiert** wird.

Hängen?: DTM'n auf beidseitig unendlichem Band können aber nicht hängen.

Stoppen nach n Schritten.

Stoppen: Wir wollen unter **eine DTM macht höchstens n Schritte** folgendes verstehen:

- Sie **hält** (und **akzeptiert** die Eingabe) innerhalb von n Schritten.
- Sie **hängt** (und **akzeptiert** die Eingabe **nicht**) innerhalb von n Schritten.
- Sie **stoppt** innerhalb von n Schritten **ohne in den Haltezustand überzugehen**. Auch hier wird die Eingabe nicht akzeptiert.

Antworten (informell)

Zeit: Jede Sprache aus $\mathbf{DTIME}(f(n))$ ist entscheidbar: Man warte einfach solange wie $f(n)$ angibt. Falls bis dahin kein “Ja” gekommen ist, ist die Antwort eben “Nein”.

Speicher: Jede Sprache aus $\mathbf{DSPACE}(f(n))$ ist entscheidbar. Denn es gibt nur **endlich viele verschiedene Konfigurationen**. Falls also die DTM nicht terminiert (was passiert), macht man folgendes: man schreibt alle Konfigurationen auf (die komplette Rechnung). Falls die DTM nicht terminiert, muss sie in eine Schleife laufen (eine Konfiguration erreichen, die sie schon einmal hatte). Das lässt sich feststellen.

Antworten (informell)

NTM vs. DTM: Trivial sind $\mathbf{DTIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$ und $\mathbf{DSPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$. Versucht man aber eine NTM durch eine DTM zu simulieren, braucht man (vermutlich) exponentiell mehr Zeit.

Für die Speicherkomplexität kann man zeigen:

$$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DSPACE}(f^2(n)).$$

Antworten (informell):

Zeit vs. Speicher: Trivial sind $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n))$ und $\mathbf{NTIME}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$.

Aber $\mathbf{DSPACE}(f(n))$, $\mathbf{NSPACE}(f(n))$ sind viel größer.

Konstante Faktoren werden ignoriert

Nur **die funktionale Wachstumsrate einer Funktion in Komplexitätsklassen zählt**: Konstante Faktoren werden ignoriert.

Theorem 16.3 (Bandkompression)

Für jedes $c \in \mathbb{R}^+$ und jede Speicherfunktion $S(n)$ gilt:

$$\mathbf{DSPACE}(S(n)) = \mathbf{DSPACE}(cS(n))$$

$$\mathbf{NSPACE}(S(n)) = \mathbf{NSPACE}(cS(n))$$

Beweis

Eine Richtung ist trivial. Die andere geht, indem man eine feste Anzahl r ($> \frac{2}{c}$) von benachbarten Zellen auf dem Band als **ein neues Symbol** darstellt. **Die Zustände der neuen Maschine simulieren die alten Kopfbewegungen als Zustandsübergänge** (innerhalb des neuen Symbols). D.h. für r Zellen der alten Maschine werden nun nur maximal 2 benutzt: im ungünstigsten Fall, wenn man von einem Block in den benachbarten geht.

Theorem 16.4 (Zeitbeschleunigung)

Für jedes $c \in \mathbb{R}^+$ und jede Zeitfunktion $T(n)$ mit $\lim_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$ gilt:

$$\mathbf{DTIME}(T(n)) = \mathbf{DTIME}(cT(n))$$

$$\mathbf{NTIME}(T(n)) = \mathbf{NTIME}(cT(n))$$

Beweis

Eine Richtung ist trivial. Der Beweis der anderen läuft wieder über die Repräsentation einer festen Anzahl $r (> \frac{4}{c})$ benachbarter Bandzellen durch **neue Symbole**. Im Zustand der neuen Maschine wird wieder kodiert, welches Zeichen und welche Kopfposition (der simulierten, alten Maschine) aktuell ist.

Wenn die alte Maschine simuliert wird, muss die neue nur 4 statt r Schritte machen: 2 um die neuen Felder zu drucken und weitere 2 um den Kopf auf das neue und wieder zurück auf das alte (im schlimmsten Fall) zu bewegen.

Lemma 16.5 (Wachstumsrate von DTIME, DSPACE)

Es sei $T(n)$ eine Zeitfunktion mit $\lim_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$, und es sei $S(n)$ eine Speicherfunktion.

- (a) Es sei $f(n) = \mathbf{O}(T(n))$. Dann gilt:
 $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DTIME}(T(n))$.
- (b) Es sei $g(n) = \mathbf{O}(S(n))$. Dann gilt:
 $\mathbf{DSPACE}(g(n)) \subseteq \mathbf{DSPACE}(S(n))$.

Definition 16.6 (P, NP, PSPACE)

$$\begin{aligned} \mathbf{P} &:= \bigcup_{i \geq 1} \mathbf{DTIME}(n^i) \\ \mathbf{NP} &:= \bigcup_{i \geq 1} \mathbf{NTIME}(n^i) \\ \mathbf{PSPACE} &:= \bigcup_{i \geq 1} \mathbf{DSPACE}(n^i) \end{aligned}$$

Intuitiv

- Probleme in **P** sind effizient lösbar, jene aus **NP** können in exponentieller Zeit gelöst werden.
- **PSPACE** ist eine sehr große Klasse, weit größer als **P** oder **NP**.

Komplexitätsklassen für Funktionen

Eine Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ ist in **P**, falls es eine DTM \mathcal{M} und ein Polynom $p(n)$ gibt, so daß für jedes n der Funktionswert $f(n)$ in höchstens $p(\text{länge}(n))$ Schritten von \mathcal{M} berechnet wird. Dabei gilt $\text{länge}(n) = \lg n$, denn man braucht $\lg n$ Zeichen um die Zahl n binär darzustellen.

Analog funktioniert dies für alle anderen Komplexitätsklassen.

Frage

Was sind die genauen Beziehungen zwischen den Komplexitätsklassen P, NP, PSPACE?

$$P \subseteq NP \subseteq PSPACE$$

Frage

Wie zeigen wir, daß ein gegebenes Problem in einer bestimmten Klasse ist?

Antwort

Reduktion auf ein bekanntes!

Wir brauchen eines, mit dem wir anfangen können: SAT

Frage

Können wir in **NP** Probleme finden, die **die schwierigsten in NP** sind?

Antwort

Es gibt mehrere Wege, ein schwerstes Problem zu definieren. Sie hängen davon ab, welchen **Begriff von Reduzierbarkeit** wir benutzen.

Für einen gegebenen Begriff von Reduzierbarkeit ist die Antwort: **Ja**. Solche Probleme werden **vollständig in der gegebenen Klasse** bezüglich des Begriffs der Reduzierbarkeit genannt.

Definition 16.7 (Polynomial-Zeit-Reduzibilität)

Seien L_1, L_2 Sprachen.

Polynomial-Zeit: L_2 ist Polynomial-Zeit reduzibel auf L_1 , bezeichnet mit $L_2 \preceq_{\text{pol}} L_1$, wenn es eine **Polynomial-Zeit beschränkte DTM** gibt, die für jede Eingabe w eine Ausgabe $f(w)$ erzeugt, so daß

$$w \in L_2 \text{ gdw } f(w) \in L_1$$

Lemma 16.8 (Polynomial-Zeit-Reduktionen)

① Sei L_2 Polynomial-Zeit-reduzibel auf L_1 . Dann gilt

L_2 ist in **NP** wenn L_1 in **NP** ist

L_2 ist in **P** wenn L_1 in **P** ist

② Die Komposition zweier Polynomial-Zeit-Reduktionen ist wieder eine Polynomial-Zeit-Reduktionen.

Theorem 16.9 (Charakterisierung von NP)

Eine Sprache L ist in **NP** genau dann wenn es eine Sprache L' in **P** und ein $k \geq 0$ gibt, so dass für alle $w \in \Sigma$ gilt:

$$w \in L \text{ gdw. es gibt ein } c : \langle w, c \rangle \in L' \text{ und } |c| < |w|^k.$$

c wird **Zeuge** (witness oder Zertifikat/certificate) von w in L genannt. Eine DTM, die die Sprache L' akzeptiert, wird **Prüfer** (verifier) von L genannt.

Wichtig:

Ein Entscheidungsproblem ist in **NP** genau dann wenn **jede Ja-Instanz ein kurzes Zertifikat** hat (d.h. seine Länge polynomial in der Länge der Eingabe ist), welche in polynomial-Zeit verifiziert werden kann.

Teil VI

Komplexitätstheorie

- 1 Die Struktur von PSPACE
- 2 Vollständige und harte Probleme**
- 3 Beispiele

Definition 17.1 (NP-vollständig, NP-hart)

- Eine Sprache L heißt **NP-hart (NP-schwer)** wenn jede Sprache $L' \in \mathbf{NP}$ polynomial-zeit-reduzibel auf L ist.
- Eine Sprache L heißt **NP-vollständig** wenn sie
 - 1 in \mathbf{NP} ist ($L \in \mathbf{NP}$), und
 - 2 **NP-hart** ist

Definition 17.2 (PSPACE-vollständig, PSPACE-hart)

- Eine Sprache L heißt **PSPACE-hart (PSPACE-schwer)** wenn jede Sprache $L' \in \mathbf{PSPACE}$ polynomial-zeit-reduzibel auf L ist.
- Eine Sprache L heißt **PSPACE-vollständig** wenn sie
 - 1 in \mathbf{PSPACE} ist ($L \in \mathbf{PSPACE}$) und
 - 2 **PSPACE-hart** ist

Bemerkenswert

- Wenn gezeigt werden kann, daß auch nur ein einziges **NP**-hartes Problem in **P** liegt, dann ist **P = NP**.
- Wenn **P ≠ NP** gilt, dann ist kein einziges **NP**-vollständiges Problem in polynomieller Zeit lösbar.

Eine Million Euro für den, der das „P = NP“-Problem löst!
(Millenium Probleme)

Wie zeigt man NP-Vollständigkeit?

Um zu zeigen, dass eine Sprache L **NP**-vollständig ist:

Finde bekanntermaßen **NP**-vollständige Sprache L' und reduzieren sie auf L :

$$L' \preceq L$$

Das genügt, da jede Sprache aus **NP** auf L' reduzierbar ist und wegen $L' \preceq L$ dann auch auf L .

Hierfür häufig verwendet:
SAT-Problem, d.h.

$$L' = L_{\text{sat}} = \{w \mid w \text{ ist eine erfüllbare aussagenlogische Formel}\}$$

P, PSPACE abgeschlossen unter Komplement

Alle Komplexitätsklassen, die mittels **deterministischer Turing-Maschinen** definiert sind, sind **abgeschlossen unter Komplement-Bildung**

Denn:

Wenn eine Sprache L dazu gehört, dann auch ihr Komplement (einfach die alte Maschine ausführen und die Ausgabe invertieren).

Abgeschlossenheit von NP unter Komplement

Frage:

Ist **NP** abgeschlossen unter Komplementbildung?

Antwort:

Keiner weiß es!

Definition 17.3 (co-NP)

co-NP ist die Klasse der Sprachen deren Komplemente in **NP** liegen:

$$\mathbf{co-NP} = \{L \mid \bar{L} \in \mathbf{NP}\}$$

Die folgenden Beziehungen sind momentan noch unbekannt

- 1 $P \neq NP$.
- 2 $NP \neq \text{co-NP}$.
- 3 $P \neq \text{PSPACE}$.
- 4 $NP \neq \text{PSPACE}$.

Teil VI

Komplexitätstheorie

- 1 Die Struktur von PSPACE
- 2 Vollständige und harte Probleme
- 3 Beispiele**

Beispiel 18.1 (NP vollständige Probleme)

- Ist ein (un-) gerichteter Graph hamiltonsch? (**Hamiltonian circle**)
- Ist eine logische Formel erfüllbar? (**Satisfiability**)
- Gibt es in einem Graphen eine Clique der Größe k ? (**Clique of size k**)
- Ist ein Graph mit drei Farben zu färben? (**3-colorability**)
- Gibt es in einer Menge von ganzen Zahlen eine Teilmenge mit der Gesamtsumme x ? (**Subset Sum**)

Definition 18.2 (Hamilton Circle)

Hamilton-Kreis:

Weg entlang der Kanten in einem Graphen, der jeden Knoten genau einmal besucht

- $L_{\text{Ham}_{\text{undir}}}$: Die Sprache, die aus allen ungerichteten Graphen besteht, in denen es einen Hamilton-Kreis gibt.
- $L_{\text{Ham}_{\text{dir}}}$: Die Sprache, die aus allen gerichteten Graphen besteht, in denen es einen Hamilton-Kreis gibt.

Definition 18.3 (Maximale Clique: L_{Clique_k})

Eine **Clique** in einem Graphen ist ein **vollständiger Teilgraph von G** .

Für $k \in \mathbb{N}$:

L_{Clique_k} Die Sprache, die aus allen ungerichteten Graphen besteht, die eine Clique der Größe k enthalten.

Definition 18.4 (k -colorability: $L_{\text{Color}_{\leq k}}$)

Ein (ungerichteter) Graph heißt **k -färbbar**, falls jeder Knoten mit einer von k Farben so gefärbt werden kann, daß benachbarte Knoten verschiedene Farben haben.

Für $k \in \mathbb{N}$:

$L_{\text{Color}_{\leq k}}$ Die Sprache, die aus allen ungerichteten, mit höchstens k Farben färbbaren Graphen besteht.

Definition 18.5 (SAT, k -CNF, k -DNF)

DNF: Eine Formel ist in **disjunktiver Normalform**, wenn sie von folgender Form ist:

$$(l_{11} \wedge \dots \wedge l_{1n_1}) \vee \dots \vee (l_{m1} \wedge \dots \wedge l_{mn_m})$$

CNF: Eine Formel ist in **konjunktiver Normalform**, wenn sie von folgender Form ist:

$$(l_{11} \vee \dots \vee l_{1n_1}) \wedge \dots \wedge (l_{m1} \vee \dots \vee l_{mn_m})$$

Definition (Fortsetzung)

- k -DNF:** Eine Formel ist in k -DNF wenn sie in DNF ist und jede ihrer Konjunktionen genau k Literale hat.
- k -CNF:** Eine Formel ist in k -CNF wenn sie in CNF ist und jede ihrer Disjunktion genau k Literale hat.

Theorem 18.6 (NP-vollständige Probleme)

Die folgenden Probleme liegen in **NP** und sind **NP-vollständig**:

- L_{sat}
- CNF
- k - CNF für $k \geq 3$

Theorem 18.7 (Probleme in P)

Die folgenden Probleme liegen in **P**:

- DNF
- k - DNF für alle k
- 2 - CNF

Einige Beispiel-Reduktionen

- $L_{\text{CNF-SAT}} \preceq_{\text{pol}} L_{\text{Clique}_{\leq k}}$,
- $L_{\text{Ham}_{\text{dir}}} \preceq_{\text{pol}} L_{\text{Ham}_{\text{undir}}}$,
- $L_{\text{Ham}_{\text{undir}}} \preceq_{\text{pol}} L_{\text{Ham}_{\text{cost} \leq k}}, L_{\text{Clique}_k}$,
- $L_{\text{SAT}} \preceq_{\text{pol}} L_{\text{3-CNF}}$,
- $L_{\text{SAT}} \preceq_{\text{pol}} L_{\text{CNF-SAT}}$,
- $L_{\text{3-CNF}} \preceq_{\text{pol}} L_{\text{Color}_{\leq k}}$.

Beispiel 18.8 (CNF-SAT \preceq_{pol} Clique $_{\leq k}$)

Gegeben eine Instanz von CNF

(eine Konjunktion von Klauseln $C_1 \wedge C_2 \wedge \dots \wedge C_k$)

Wir konstruieren daraus einen Graphen:

Knoten: die Paare (x, i) , so dass x ein Literal ist, dass in der Klausel C_i vorkommt.

Kanten: Es gibt eine Kante zwischen (x, i) und (y, j) falls:

- (1) $i \neq j$, und
- (2) x, y sind nicht komplementär.

Es gilt dann:

Die CNF-Formel ist erfüllbar genau dann, wenn der zugeordnete Graph eine Clique der Größe k hat.