Vorlesungsskript

Komplexitätstheorie

Wintersemester 2004/2005

Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin
Lehrstuhl Komplexität und Kryptografie

21. Februar 2005

Inhaltsverzeichnis

1 Einführung **1**

2 Rechenmodelle **4**

2.1 Deterministische Turingmaschinen 4

2.2 Nichtdeterministische Berechnungen 6

2.3 Zeitkomplexität 7

2.4 Platzkomplexität 8

3 Grundlegende Beziehungen **10**

4 Hierarchiesätze **20**

5 Reduktionen **26**

6 Probabilistische Berechnungen **39**

6.1 Reduktion der Fehlerwahrscheinlichkeit 43

7 Die Polynomialzeithierachie **47**

7.1 Anzahl-Operatoren 47

8 Das Graphisomorphieproblem **52**

9 Turing-Operatoren **59**

1 Einführung

In der Komplexitätstheorie werden algorithmische Probleme daraufhin untersucht, wieviel Rechenressourcen zu ihrer Lösung benötigt werden. Naturgemäß bestehen daher enge Querbezüge zu

- Algorithmen (obere Schranken)
- Automatentheorie (Rechenmodelle)
- Berechenbarkeit (Was ist überhaupt algorithmisch lösbar?)
- Logik (liefert viele algorithmische Probleme, mit ihrer Hilfe kann auch die Komplexität von Problemen charakterisiert werden)
- Kryptographie (Wieviel Rechenressourcen benötigt ein Gegner, um ein Kryptosystem zu brechen?)

Zur weiteren Motivation betrachten wir eine Reihe von konkreten algorithmischen Problemstellungen.

Erreichbarkeitsproblem in Graphen (REACH):

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$ und $E \subseteq V \times V$.

Gefragt: Gibt es in G einen Weg von Knoten 1 zu Knoten n?

Zur Erinnerung: Eine Folge $(v_1, \ldots, v_k)$ von Knoten heißt **Weg** in G, falls für $j = 1, \ldots, k-1$ gilt: $(v_j, v_{j+1}) \in E$.

Da als Antwort nur "ja" oder "nein" möglich ist, handelt es sich um ein **Entscheidungsproblem**. Ein solches lässt sich formal durch eine Sprache beschreiben, die alle positiven (mit "ja" zu beantwortenden) Problemeingaben enthält:

$$\text{REACH} = \{G \mid \text{es ex. ein Weg von } 1 \text{ nach } n\}$$

Hierbei setzen wir eine Kodierung von Graphen durch Wörter über einem geeigneten Alphabet Σ voraus. Wir könnten G beispielsweise durch eine Binärfolge der Länge n^2 kodieren, die aus den n Zeilen der Adjazenzmatrix von G gebildet wird.

Um REACH zu entscheiden, markieren wir nach und nach alle Knoten, die vom Knoten 1 aus erreichbar sind. Dabei speichern wir alle markierten Knoten solange in einer Menge S, bis wir auch deren Nachbarknoten markiert haben. Genaueres ist folgendem Algorithmus zu entnehmen:

```
1  Eingabe: G = (V, E)
2    S ← {1}
3    markiere 1
4    repeat
5      wähle Knoten u ∈ S
6      S ← S − {u}
7      for all (u, v) ∈ E do
8        if v ist nicht markiert then
9          markiere v
10         S ← S ∪ {v}
11       end
12     end
13   until S = ∅
14   if n ist markiert then accept else reject end
```

Diskussion: (informal)

Es ist üblich, den Ressourcenverbrauch von Algorithmen (wie z.B. Rechenzeit oder Speicherplatz) in Abhängigkeit von der Größe der Problemeingabe zu messen. Falls die Eingabe aus einem Graphen besteht, kann beispielsweise die Anzahl n der Knoten (oder auch die Anzahl m der Kanten) als Bezugsgröße dienen. Genau genommen hängt die Eingabegröße davon ab, welche Kodierung wir für die Eingaben verwenden (vgl. Definition 12).

$$\begin{aligned} \textsc{Reach} &\in \text{DTIME}(n^3) \\ \textsc{Reach} &\in \text{NSPACE}(\log n) \\ &\subseteq \text{DSPACE}(\log^2 n) \qquad \text{(Satz von Savitch)} \end{aligned}$$

Als nächstes betrachten wir das Problem einen maximalen Fluss in einem Netzwerk zu bestimmen.

Maximaler Fluß (MAXFLOW):

Gegeben: Ein gerichteter Graph $G = (V, E)$, $V = \{1, \ldots, n\}$, $E \subseteq V^2$ mit einer Kapazitätsfunktion $c : E \to \mathbb{N}$.

Gesucht: Ein Fluss $f : E \to \mathbb{N}$ von 1 nach n in G, d.h.

- $\forall e \in E : f(e) \leq c(e)$
- $\forall v \in V - \{1, n\} : \sum_{(v,u)\in E} f(v, u) = \sum_{(u,v)\in E} f(u, v)$

mit maximalem Wert $w(f) := \sum_{(1,v)\in E} f(1, v) = \sum_{(v,n)\in E} f(v, n)$.

Da hier nach einer Lösung (Fluss) mit maximalem Wert gesucht wird, handelt es sich um ein **Optimierungsproblem**. Im Gegensatz hierzu wird bei vielen Entscheidungsproblemen nach der Existenz einer Lösung (mit gewissen Eigenschaften) gefragt.

Diskussion: (informal)

$$\begin{aligned} \text{MAXFLOW} &\in \text{DTIME}(n^5) \\ \text{MAXFLOW} &\in \text{DSPACE}(n^2) \end{aligned}$$

Das folgende Problem scheint zwar auf den ersten Blick nur wenig mit dem Problem MAXFLOW gemein zu haben. In Wirklichkeit entpuppt es sich jedoch als ein Spezialfall von MAXFLOW.

Perfektes Matching in bipartiten Graphen (MATCHING):

Gegeben: Ein bipartiter Graph $G = (U, V, E)$, $U = V = \{1, \ldots, n\}$, $E \subseteq U \times V$.

Gesucht: Besitzt G ein perfektes Matching?

Zur Erinnerung: Eine Kantenmenge $M \subseteq E$ heißt **Matching**, falls für alle Kanten $e = (u, v), e' = (u', v') \in M$ mit $e \neq e'$ gilt: $u \neq u'$ und $v \neq v'$. Gilt zudem $\|M\| = n$, so heißt M **perfekt**.

Diskussion: (informal)

$$\begin{aligned} \text{MATCHING} &\in \text{DTIME}(n^3) \\ \text{MATCHING} &\in \text{DSPACE}(n^2) \end{aligned}$$

Die bisher betrachteten Probleme können in deterministischer Polynomialzeit gelöst werden und gelten daher als effizient lösbar. Zum Schluss dieses Abschnitts betrachten wir ein Problem, für das vermutlich nur ineffiziente Algorithmen existieren.

Travelling Salesman Problem (TSP):

Gegeben: Eine symmetrische $n \times n$-Distanzmatrix $D = (d_{ij})$ mit $d_{ij} \in \mathbb{N}$.

Gesucht: Eine kürzeste Rundreise, d.h. eine Permutation $\pi \in S_n$ mit minimalem Wert $w(\pi) := \sum_{i=1}^{n} d_{\pi(i),\pi(i+1)}$, wobei $\pi(n+1) := \pi(1)$.

Diskussion (informal)

$\text{TSP} \in \text{DTIME}(n!)$ (Ausprobieren aller Rundreisen)

$\text{TSP} \in \text{DSPACE}(n)$

$\text{TSP} \in \text{DTIME-SPACE}(n!, n)$

$\text{TSP} \in \text{DTIME-SPACE}(n^2 \cdot 2^n, n \cdot 2^n)$ (dynamisches Programmieren*)

* Hierzu berechnen wir für alle Teilmengen $S \subseteq \{2, \ldots, n\}$ und alle $j \in S$ die Länge $l(S, j)$ eines kürzesten Pfades von 1 nach j, der alle Städte in S genau einmal besucht.

2 Rechenmodelle

2.1 Deterministische Turingmaschinen

Definition 1 (Mehrband-Turingmaschine)
Eine **deterministische k-Band-Turingmaschine** (k-DTM oder einfach DTM) ist ein Quadrupel $M = (Q, \Sigma, \Gamma, \delta, q_0)$. Dabei ist

- Q eine endliche Menge von Zuständen,
- Σ eine endliche Menge von Symbolen (das Eingabealphabet) mit $\sqcup, \triangleright \notin \Sigma$ ($\sqcup$ heißt Blank und $\triangleright$ heißt Anfangssymbol),
- Γ das Arbeitsalphabet mit $\Sigma \cup \{\sqcup, \triangleright\} \subseteq \Gamma$,
- $\delta : Q \times \Gamma^k \to (Q \cup \{q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}\}) \times (\Gamma \times \{L, R, N\})^k$ die Überführungsfunktion (q_{h} heißt Haltezustand, q_{ja} akzeptierender und q_{nein} verwerfender Endzustand) und
- q_0 der Startzustand.

Befindet sich M im Zustand q und stehen die Schreib-Lese-Köpfe auf Feldern mit der Inschrift a_i (auf dem i-ten Band), so bedeutet die Anweisung $\delta(q, a_1, \ldots, a_k) = (q', a'_1, D_1, \ldots, a'_k, D_k)$, dass M in den Zustand q' übergeht, auf Band i das Symbol a_i durch a'_i ersetzt und den Kopf gemäß D_i bewegt (im Fall $D_i = L$ um ein Feld nach links, im Fall $D_i = R$ um ein Feld nach rechts und im Fall $D_i = N$ wird der Kopf nicht bewegt).

Außerdem verlangen wir von der Überführungsfunktion δ, dass im Fall $\delta(q, a_1, \ldots, a_k) = (q', a'_1, D_1, \ldots, a'_k, D_k)$ und $a_i = \triangleright$ immer $a'_i = \triangleright$ und $D_i = R$ gilt (wird also auf einem Band das Anfangszeichen gelesen, so darf dieses nicht durch ein anderes Zeichen überschrieben und der Kopf muss auf diesem Band nach rechts bewegt werden).

Definition 2 (Konfiguration)
Eine Konfiguration ist ein $(2k+1)$-Tupel $K = (q, u_1, v_1, \ldots, u_k, v_k) \in Q \times (\Gamma^* \times \Gamma^+)^k$ und besagt, dass

- q der momentane Zustand und
- $u_i v_i \sqcup \sqcup \cdots$ die Inschrift des i-ten Bandes ist, sowie
- sich der Kopf auf Band i auf dem ersten Zeichen von v_i befindet.

Definition 3 (Folgekonfiguration)
Eine Konfiguration $K' = (q', u'_1, v'_1, \ldots, u'_k, v'_k)$ heißt Folgekonfiguration von K (kurz: $K \underset{M}{\longrightarrow} K'$), falls $D_i \in \{L, R, N\}$, $a'_i, b_i \in \Gamma$ existieren mit $\delta(q, a_1, \ldots, a_k) = (q', a'_1, D_1, \ldots, a'_k, D_k)$ und für $i = 1, \ldots, k$ gilt jeweils eine der drei Bedingungen

1. $D_i = \text{L}$, $u_i = u'_i b_i$ und $v'_i = b_i a'_i v_i$,
2. $D_i = \text{R}$, $u'_i = u_i a'_i$ und $v'_i = \begin{cases} \sqcup, & v_i = \varepsilon, \\ v_i, & \text{sonst}, \end{cases}$
3. $D_i = \text{N}$, $u'_i = u_i$ und $v'_i = a'_i v_i$.

Wir schreiben $K \underset{M}{\longrightarrow}^t K'$, falls Konfigurationen $K_1, \ldots, K_t$ existieren mit $K_1 = K$ und $K_t = K'$, sowie $K_i \underset{M}{\longrightarrow} K_{i+1}$ für $i = 1, \ldots, t-1$. Die reflexive, transitive Hülle von $\underset{M}{\longrightarrow}$ bezeichnen wir mit $\underset{M}{\longrightarrow}^*$, d.h. $K \underset{M}{\longrightarrow}^* K'$ bedeutet, dass ein $t \geq 0$ existiert mit $K \underset{M}{\longrightarrow}^t K'$.

Definition 4 (Startkonfiguration)
Sei $x \in \Sigma^*$ eine Eingabe. Die zugehörige Startkonfiguration ist

$$K_x = (q_0, \varepsilon, \triangleright x, \underbrace{\varepsilon, \triangleright, \ldots, \varepsilon, \triangleright}_{(k-1)\text{-mal}}).$$

Definition 5 (Endkonfiguration)
Eine Konfiguration $K = (q, u_1, v_1, \ldots, u_k, v_k)$ mit $q \in \{q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}\}$ heißt **Endkonfiguration**. Im Fall $q = q_{\text{ja}}$ (bzw. $q = q_{\text{nein}}$) heißt K **akzeptierende** (bzw. **verwerfende**) **Endkonfiguration**.

Definition 6 (Resultat einer Rechnung)
Eine DTM M **hält** bei Eingabe $x \in \Sigma^*$ im Zustand q (kurz: $M(x)$ hält im Zustand q), falls es eine Endkonfiguration $K = (q, u_1, v_1, \ldots, u_k, v_k)$ gibt mit

$$K_x \underset{M}{\longrightarrow}^* K.$$

Weiter definieren wir das **Resultat** $M(x)$ der Rechnung von M bei Eingabe x,

$$M(x) = \begin{cases} \text{ja}, & M(x) \text{ hält im Zustand } q_{\text{ja}}, \\ \text{nein}, & M(x) \text{ hält im Zustand } q_{\text{nein}}, \\ y, & M(x) \text{ hält im Zustand } q_{\text{h}}, \\ \uparrow, & \text{sonst}. \end{cases}$$

Dabei ergibt sich y aus $u_k v_k$, indem das erste Symbol $\triangleright$ und sämtliche Blanks am Ende entfernt werden, d. h. $u_k v_k = \triangleright y \sqcup^i$. Für $M(x) =$ ja sagen wir auch „$M(x)$ akzeptiert“ und für $M(x) =$ nein „$M(x)$ verwirft“.

Definition 7 (Akzeptieren einer Sprache durch DTMs)
Die von einer DTM M **akzeptierte Sprache** ist

$$L(M) = \{x \in \Sigma^* \mid M(x) \text{ akzeptiert}\}.$$

Eine DTM, die eine Sprache L akzeptiert, darf also bei Eingaben $x \notin L$ unendlich lange rechnen. In diesem Fall heißt L **rekursiv aufzählbar**. Dagegen muss eine DTM, die eine Sprache L entscheidet, bei jeder Eingabe halten.

Definition 8 (entscheidbare Sprachen)
Sei $L \subseteq \Sigma^*$. Eine DTM M **entscheidet** L, falls für alle $x \in \Sigma^*$ gilt:

$$x \in L \Rightarrow M(x) \text{ akz.}$$
$$x \notin L \Rightarrow M(x) \text{ verw.}$$

In diesem Fall heißt L **entscheidbar** (oder **rekursiv**).

Definition 9 (berechenbare Funktionen)
Sei $f : \Sigma^* \to \Sigma^*$ eine Funktion. Eine DTM M **berechnet** f, falls für alle $x \in \Sigma^*$ gilt:

$$M(x) = f(x).$$

f heißt dann **berechenbar** (oder **rekursiv**).

Aus dem Grundstudium wissen wir, dass eine nichtleere Sprache $L \subseteq \Sigma^*$ genau dann rekursiv aufzählbar ist, wenn eine rekursive Funktion $f : \Sigma^* \to \Sigma^*$ existiert, deren Bild $\{f(x) \mid x \in \Sigma^*\}$ die Sprache L ist.

2.2 Nichtdeterministische Berechnungen

Anders als eine DTM, für die in jeder Konfiguration höchstens eine Anweisung ausführbar ist, hat eine nichtdeterministische Turingmaschine in jedem Rechenschritt die Wahl unter einer endlichen Anzahl von Anweisungen.

Definition 10 (nichtdeterministische Mehrband-Turingmaschine)
Eine **nichtdeterministische k-Band-Turingmaschine** (kurz k-NTM oder einfach NTM) ist ein 5-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0)$, wobei Q, Σ, Γ, q_0 genau wie bei einer k-DTM definiert sind und

$$\delta : Q \times \Gamma^k \to \mathcal{P}(Q \cup \{q_{\text{h}}, q_{\text{ja}} q_{\text{nein}}\} \times (\Gamma \times \{R, L, N\})^k)$$

eine Funktion mit der Eigenschaft ist, dass im Fall $(q', a'_1, D_1, \ldots, a'_k, D_k) \in \delta(q, a_1, \ldots, a_k)$ und $a_i = \triangleright$ immer $a'_i = \triangleright$ und $D_i = R$ gilt.

Die Begriffe **Konfiguration**, **Start-** und **Endkonfiguration** übertragen sich unmittelbar von DTMs auf NTMs. Der Begriff der **Folgekonfiguration** lässt sich übertragen, indem wir $\delta(q, a_1, \ldots, a_k) = (q', a'_1, D_1, \ldots, a'_k, D_k)$ durch $(q', a'_1, D_1, \ldots, a'_k, D_k) \in$

$\delta(q, a_1, \ldots, a_k)$ ersetzen. Entsprechend erhalten wir die Relationen $\underset{M}{\longrightarrow}$, $\underset{M}{\longrightarrow}^k$ und $\underset{M}{\longrightarrow}^*$.

Definition 11 (Akzeptieren einer Sprache durch NTMs)

Eine NTM M **akzeptiert** eine Eingabe $x \in \Sigma^*$ (kurz: $M(x)$ akzeptiert), falls eine akzeptierende Endkonfiguration K existiert mit $K_x \rightarrow^* K$. Die von einer NTM M **akzeptierte Sprache** ist genau so definiert wie bei einer DTM.

2.3 Zeitkomplexität

Der Zeitverbrauch $time_M(x)$ einer Turingmaschine M bei Eingabe x ist die maximale Anzahl an Rechenschritten, die M ausgehend von der Startkonfiguration K_x ausführen kann (bzw. ∞, falls unendlich lange Rechnungen existieren).

Definition 12 (Rechenzeit)

Sei M eine TM und sei $x \in \Sigma^*$ eine Eingabe. Dann ist

$$time_M(x) = \max\{t \geq 0 \mid \exists K : K_x \vdash^t K\}$$

die **Rechenzeit** von M bei Eingabe x, wobei $\max \mathbb{N} := \infty$.

Sei $t : \mathbb{N} \rightarrow \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $t(n)$-**zeitbeschränkt**, falls für alle $x \in \Sigma^*$ gilt:

$$\mathrm{time}_M(x) \leq t(|x|).$$

Alle Sprachen, die in (nicht-)deterministischer Zeit $t(n)$ entscheidbar sind, fassen wir in den Komplexitätsklassen

$$\text{DTIME}(t(n)) := \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte DTM}\}$$

und

$$\text{NTIME}(t(n)) := \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte NTM}\}$$

zusammen. Ferner sei

$$\text{FTIME}(t(n)) := \{f \mid f \text{ wird von einer } t(n)\text{-zeitbeschränkten DTM berechnet}\}.$$

Die wichtigsten Zeitkomplexitätsklassen sind

$$\begin{aligned}
\text{LINTIME} &= \bigcup_{c \geq 1} \text{DTIME}(cn), \\
\text{P} &= \bigcup_{c \geq 1} \text{DTIME}(n^c), \\
\text{E} &= \bigcup_{c \geq 1} \text{DTIME}(2^{cn}), \\
\text{EXP} &= \bigcup_{c \geq 1} \text{DTIME}(2^{n^c}).
\end{aligned}$$

Die Klassen NP, NE, NEXP und FP, FE, FEXP sind analog definiert.

2.4 Platzkomplexität

Zur Definition von Platzkomplexitätsklassen verwenden wir sogenannte Offline-Turingmaschinen und Transducer. Diese haben die Eigenschaft, dass sie das erste Band nur als Eingabeband (also nur zum Lesen) bzw. das k-te Band nur als Ausgabeband (also nur zum Schreiben) benutzen dürfen. Der Grund für diese Einschränkungen liegt darin, sinnvolle Definitionen für Komplexitätsklassen mit einem sublinearen Platzverbrauch zu erhalten.

Definition 13 (Offline-Turingmaschine, Transducer)

Eine **Offline-DTM** ist eine DTM M mit der Eigenschaft, dass im Fall $\delta(q, a_1, \ldots, a_k) = (q', a'_1, D_1, \ldots, a'_k, D_k)$ immer

- $a'_1 = a_1$
- $a_1 = \sqcup \Rightarrow D_1 = L$

gilt. Gilt weiterhin immer $D_k \neq L$, so heißt M **Transducer**.

Dies bedeutet, dass eine Offline-Turingmaschine nicht auf das Eingabeband schreiben darf (*read-only*). Beim Transducer dient das letzte Band als Ausgabeband, auch hier können keine Berechnungen durchgeführt werden (*write-only*).

Der Zeitverbrauch $time_M(x)$ von Offline-DTMs und von Transducern ist genauso definiert wie bei DTMs. Als nächstes definieren wir den Platzverbrauch einer TM als die Summe aller während einer Rechnung besuchten Bandfelder.

Definition 14 (Platzverbrauch)

Sei M eine Turingmaschine und sei $x \in \Sigma^*$ eine Eingabe. Dann ist

$$\mathrm{space}_M(x) = \max\{s \geq 1 \mid \exists K = (q, u_1, v_1, \ldots, u_k, v_k) \text{ mit } s = \sum_{i=1}^{k} |u_i v_i| \text{ und } K_x \vdash^t K\}$$

der **Platzverbrauch** von M bei Eingabe x. Für eine Offline-DTM ersetzen wir $\sum_{i=1}^{k} |u_i v_i|$ durch $\sum_{i=2}^{k} |u_i v_i|$ und für einen Transducer durch $\sum_{i=2}^{k-1} |u_i v_i|$. Sei $s : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $s(n)$-**platzbeschränkt**, falls für alle $x \in \Sigma^*$ gilt:

$$\mathrm{space}_M(x) \leq s(|x|).$$

Alle Sprachen, die in (nicht-) deterministischem Platz $s(n)$ entscheidbar sind, fassen wir in den Komplexitätsklassen

$$\mathrm{DSPACE}(s(n)) := \{L(M) \mid M \text{ ist eine } s(n)\text{-platzbeschränkte Offline-DTM}\}$$

und

$$\text{NSPACE}(s(n)) := \{L(M) \mid M \text{ ist eine } s(n)\text{-platzbeschränkte Offline-NTM}\}$$

zusammen. Ferner sei

$$\text{FSPACE}(s(n)) := \{f \mid f \text{ wird von einem } s(n)\text{-platzbeschränkten Transducer berechnet}\}.$$

Die wichtigsten Platzkomplexitätsklassen sind

$$\begin{aligned}
\text{L} &= \text{DSPACE}(\log n) \\
\text{L}^c &= \text{DSPACE}(\log^c n) \\
\text{LINSPACE} &= \text{DSPACE}(n) = \bigcup_{c>0} \text{DSPACE}(cn) \\
\text{PSPACE} &= \bigcup_{c>0} \text{DSPACE}(n^c) \\
\text{ESPACE} &= \bigcup_{c>0} \text{DSPACE}(2^{cn}) \\
\text{EXPSPACE} &= \bigcup_{c>0} \text{DSPACE}(2^{n^c})
\end{aligned}$$

Die Klassen NL, NLINSPACE und NPSPACE, sowie FL, FLINSPACE und FPSPACE sind analog definiert, wobei NPSPACE mit PSPACE zusammenfällt (wie wir bald sehen werden).