Vorlesungsskript

Komplexitätstheorie

Wintersemester 2001/2002

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

14. Februar 2002

Inhaltsverzeichnis

1 Einführung

In der Komplexitätstheorie werden algorithmische Problemstellungen daraufhin untersucht, welche Rechenressourcen zu ihrer Lösung benötigt werden.

Naturgemäß bestehen daher enge Querbezüge zu

- Algorithmen (obere Schranken)
- Automatentheorie (Rechenmodelle)
- Berechenbarkeit (Welche Probleme sind überhaupt algorithmisch lösbar?)
- Logik (liefert viele algorithmische Probleme, mit ihrer Hilfe kann auch die Komplexität von Problemklassen charakterisiert werden)
- Kryptografie (Welche Rechenressourcen muss ein Gegner aufbringen, um ein Kryptosystem zu brechen?)

Zur weiteren Motivation betrachten wir eine Reihe von konkreten Problemstellungen.

Definition 1 (**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:

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

Hierbei setzen wir eine Kodierung von Graphen durch Wörter über einem geeigneten Alphabet Σ voraus. Wir können 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 **??**).

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

Definition 2 **(Maximaler Fluß: MAXFLOW)**

Gegeben: ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$ und $E \subseteq V^2 - (V \times \{1\} \cup \{n\} \times V)$, sowie eine Kapazitätsfunktion $c : E \to \mathbb{N}^+$.

Gesucht: ein Fluß $f : E \to \mathbb{N}^+$ von 1 nach n, 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 $W(f) := \sum_{(1,v)\in E} f(1, v) = \sum_{(v,n)\in E} f(v, n)$.

Wir haben es hier mit einem Optimierungsproblem zu tun, das wir bereits aus dem Grundstudium kennen.

Diskussion: (informal)

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

Definition 3 (**Perfektes Matching in bipartiten Graphen: MATCHING**)

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

Gefragt: Hat G ein perfektes Matching?

Zur Erinnerung: Eine Teilmenge $M \subseteq E$, $\|M\| = n$, ist ein **perfektes Matching** in $G = (V, U, E)$, falls für je zwei Kanten $e = (u, v) \neq e' = (u', v') \in M$ gilt: $u \neq u'$ und $v \neq v'$.

Diskussion: (informal)

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

Definition 4 (**Travelling Salesman Problem: TSP**)

Gegeben: eine $n \times n$-Distanzmatrix $D = (d_{ij})$ mit

- $d_{ij} \in \mathbb{N}$
- $d_{ij} = d_{ji}$

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)

$$\begin{aligned} \text{TSP} &\in \mathbf{DTIME}(n!) \\ \text{TSP} &\in \mathbf{DSPACE}(n) \\ \text{TSP} &\in \mathbf{DTIME\text{-}SPACE}(n!, n) \\ \text{TSP} &\in \mathbf{DTIME\text{-}SPACE}(n^2 \cdot 2^n, n \cdot 2^n) \\ &\quad \text{(dynamisches Programmieren)} \end{aligned}$$

Die entscheidende Frage ist nun: Sind diese oberen Komplexitätsschranken optimal?

2 Rechenmodelle

2.1 Deterministische Turingmaschinen

Definition 5 (Mehrband-Turingmaschine)

Eine **deterministische k-Band-Turingmaschine** (k-DTM) ist ein Quintupel $M = (Q, \Sigma, \Gamma, \delta, q_0)$. Dabei ist

- Q eine endliche Menge von Zuständen,
- Σ eine endliche Menge von Symbolen (das Eingabealphabet) mit $\sqcup, \rhd \notin \Sigma$ ($\sqcup$ heißt Blank und $\rhd$ heißt Anfangssymbol),
- Γ das Arbeitsalphabet mit $\Sigma \cup \{\sqcup, \rhd\} \subseteq \Gamma$,
- $\delta : Q \times \Gamma^k \to (Q \cup \{q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}\}) \times (\Gamma \times \{L, R, N\})^k$ 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 $a_i = \rhd$ immer $a'_i = \rhd$ 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 6 (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 und dass
- sich der Kopf auf Band i auf dem ersten Zeichen von v_i befindet.

Definition 7 (Folgekonfiguration)

Eine Konfiguration $K' = (q', u_1', v_1', \ldots, u_i', v_i')$ ist **Folgekonfiguration** von $K = (q, u_1, a_1 v_1, \ldots, u_k, a_k v_k)$ (kurz $K \underset{M}{\longrightarrow} K'$), falls $D_i \in \{L, R, N\}$ und $a_i', b_i \in \Gamma$ 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 $v_i' = a_i' v_i$.

gilt. Mit $\underset{M}{\longrightarrow}^*$ bezeichnen wir die reflexive, transitive Hülle von $\underset{M}{\longrightarrow}$, d. h. $K \underset{M}{\longrightarrow}^* K'$, falls Konfigurationen $K_0, K_1, \ldots, K_m$ existieren mit $K_0 = K$, $K_m = K'$ und $K_i \underset{M}{\longrightarrow} K_{i+1}$ für $i = 0, \ldots, m-1$.

Definition 8 (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 9 (Halten einer Turingmaschine)

- 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**.
- Eine k-DTM M **hält bei Eingabe** $x \in \Sigma^*$ (kurz: $M(x)$ **hält**), falls es eine Endkonfiguration K gibt mit

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

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

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

  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 10 (Entscheidbare Sprachen)

Sei $L \subseteq \Sigma^*$. Eine k-DTM M **entscheidet** L, falls für alle $x \in \Sigma^*$ gilt:

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

In diesem Fall nennen wir L **entscheidbar**.

Definition 11 (Rekursiv aufzählbare Sprachen)

Eine k-DTM M **akzeptiert** L (kurz: $L = L(N)$), falls für alle $x \in \Sigma^*$ gilt:

$$x \in L \Leftrightarrow M(x) \text{ akz.}$$

In diesem Fall heißt L **rekursiv aufzählbar**.

Definition 12 (Rekursive Funktionen)

Sei $f : \Sigma^* \to \Sigma^*$ eine Funktion. M **berechnet** f, falls für alle $x \in \Sigma^*$ gilt:

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

In diesem Fall heißt f **rekursiv** (oder **berechenbar**).

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 Wertebereich die Sprache L ist.

2.1.1 Zeitkomplexität

Der Zeitverbrauch $time_M(x)$ einer k-DTM M bei Eingabe x ist die Anzahl der Schritte, die diese Maschine ausgehend von der Startkonfiguration K_x ausführt (bzw. ∞, falls sie nicht stoppt).

Definition 13 (Zeitkomplexität von DTMs)

Sei M eine k-DTM und $x \in \Sigma^*$. Dann ist der **Zeitverbrauch** von M bei Eingabe x,

$$time_M(x) = \begin{cases} t, & \text{falls es eine Endkonfiguration } K \text{ gibt mit } K_x \xrightarrow[M]{}^{t} K, \\ \infty, & \text{sonst.} \end{cases}$$

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

$$time_M(x) \leq \max\{t(|x|), |x| + 2\}.$$

(Die Zeit $|x| + 2$ wird benötigt, um die Eingabe x zu lesen.)

Definition 14 (Zeitkomplexitätsklassen)

$$\begin{aligned}
\mathbf{DTIME}(t(n)) &= \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte } k\text{-DTM}\} \\
\mathbf{P} &= \bigcup_{c>0} \mathbf{DTIME}(n^c) \\
&\quad \text{„in Polynomialzeit (effizient) entscheidbare Probleme“} \\
\mathbf{E} &= \bigcup_{c>0} \mathbf{DTIME}(2^{cn}) \\
&\quad \text{„in linearer Exponentialzeit entscheidbare Probleme“} \\
\mathbf{EXP} &= \bigcup_{c>0} \mathbf{DTIME}(2^{n^c}) \\
&\quad \text{„in Exponentialzeit entscheidbare Probleme“}
\end{aligned}$$

Definition 15 (Offline-Turingmaschine, Transducer)

Eine **Offline-k-DTM** ist eine k-DTM M mit der Zusatzeigenschaft, 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-k-DTMs und von Transducern ist genauso definiert wie bei k-DTMs.

2.1.2 Platzkomplexität

Definition 16 (Platzverbrauch)

- Sei M eine k-DTM und $x \in \Sigma^*$. Dann ist

$$space_M(x) := \begin{cases} \sum_{i=1}^{k} |u_i, v_i|, & \text{falls eine Endkonfiguration } K = (q, u_1, v_1, \ldots, u_k, v_k) \\ & \text{mit } K_x \xrightarrow[M]{}{}^* K \text{ existiert,} \\ \uparrow, & \text{sonst} \end{cases}$$

Für eine Offline-k-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 heißt M $s(n)$-platzbeschränkt, falls für alle $x \in \Sigma^*$ gilt:
  - $space_M(x) \neq \uparrow$,
  - $space_M(x) \leq s(|x|)$.

Definition 17 (Platzkomplexitätsklassen)

$$\begin{aligned}
\mathbf{DSPACE}(s(n)) &:= \{L \mid \text{es gibt eine } s(n)\text{-platzbeschränkte Offline-}k\text{-DTM, die } L \text{ entscheidet}\} \\
\mathbf{LOGSPACE} &= \bigcup_{c>0} \mathbf{DSPACE}(c \cdot \log^c n) \\
\mathbf{LINSPACE} &= \bigcup_{c>0} \mathbf{DSPACE}(c \cdot n) \\
\mathbf{PSPACE} &= \bigcup_{c>0} \mathbf{DSPACE}(n^c + c) \\
\mathbf{ESPACE} &= \bigcup_{c>0} \mathbf{DSPACE}(2^{cn}) \\
\mathbf{EXPSPACE} &= \bigcup_{c>0} \mathbf{DSPACE}(2^{n^c+c})
\end{aligned}$$

2.2 Nichtdeterministische Turingmaschinen

Definition 18 (Nichtdeterministische Mehrband-Turingmaschine)

Eine **nichtdeterministische k-Band-Turingmaschine** (k-NTM) ist ein 5-Tupel $N = (Q, \Sigma, \Gamma, \delta, q_0)$, wobei Q, Σ, Γ, q_0 genau wie bei einer k-DTM definiert sind und δ eine Funktion

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

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, Folge- und Startkonfiguration übertragen sich von k-DTMs auf k-NTMs, indem man $\delta(q, \ldots) = (q', \ldots)$ durch $(q', \ldots) \in \delta(q, \ldots)$ ersetzt. Entsprechend erhalten wir dann die Relationen $\xrightarrow[N]{}$, $\xrightarrow[N]{}^*$ und $\xrightarrow[N]{}^k$.

Der Zeitverbrauch $time_N(x)$ von N bei Eingabe x ist die maximale Länge einer Rechnung von M bei Eingabe x ($\max \mathbb{N} := \infty$).

Definition 19 (Zeitkomplexität von NTMs)

Sei N eine k-NTM und $x \in \Sigma^*$. Dann ist der **Zeitverbrauch** von N bei Eingabe x,

$$time_N(x) \quad := \quad \max\{t \geq 0 \mid \exists K : K_x \xrightarrow[N]{}^t K\}$$

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

$$time_N(x) \quad \leq \quad \max\{t(|x|), |x| + 2\}.$$

Definition 20 (Akzeptieren einer Sprache durch NTMs)

Sei $L \subseteq \Sigma^*$. Eine k-NTM N **akzeptiert** L (kurz: $L = L(N)$), falls für alle $x \in \Sigma^*$ gilt:

- $time_N(x) < \infty$
- $x \in L \Leftrightarrow$ es gibt eine akz. Endkonf. K mit $K_x \xrightarrow[N]{}^* K$.

Die nichtdet. Zeitkomplexitätsklassen sind wie folgt definiert:

$$\mathbf{NTIME}(t(n)) \quad := \quad \{L(N) \mid N \text{ ist eine } t(n)\text{-zeitbeschränkte } k\text{-NTM}\}$$

$\rightsquigarrow$ **NP**, **NE**, **NEXP**.

Definition 21 (Platzverbrauch von NTMs)

Sei M eine k-NTM und $x \in \Sigma^*$. Dann ist

$$space_N(x) \quad = \quad \max\{s \mid \text{es ex. eine Konfiguration } K = (q, u_1, v_1, \ldots, u_k, v_k) \text{ mit } K_x \xrightarrow[N]{}^* K \text{ und } s = \sum_{i=1}^{k} |u_i v_i|\}.$$

Für eine Offline-k-NTM N ersetzen wir wieder $\sum_{i=1}^{k} |u_i v_i|$ durch $\sum_{i=2}^{k} |u_i v_i|$ und für einen Transducer durch $\sum_{i=2}^{k-1} |u_i v_i|$.

Entsprechend definieren wir $s(n)$-**platzbeschränkt** und $\mathbf{NSPACE}(s(n))$.

$\rightsquigarrow$ **NL**, **NLINSPACE**, **NPSPACE**.