

Vorlesungsskript
Komplexitätstheorie

Wintersemester 2010/11

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

21. Oktober 2010

Inhaltsverzeichnis

1 Einführung	1
2 Rechenmodelle	3
2.1 Deterministische Turingmaschinen	3
2.2 Nichtdeterministische Berechnungen	4

1 Einführung

In der Komplexitätstheorie werden algorithmische Probleme daraufhin untersucht, welche 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)
- Kryptografie (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 $V = \{1, \dots, n\}$ 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, \dots, v_k) von Knoten heißt **Weg** in G , falls für $j = 1, \dots, k - 1$ 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:

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

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.

Wir entscheiden REACH durch einen Wegsuche-Algorithmus. Dieser markiert nach und nach alle Knoten, die vom Knoten 1 aus erreichbar sind. Hierzu speichert er jeden markierten Knoten solange in einer Menge S bis er sämtliche Nachbarknoten markiert hat. Genaueres ist folgendem Algorithmus zu entnehmen:

Algorithmus suche-Weg(G)

```
1 Input: Gerichteter Graph  $G = (V, E)$  mit  $V = \{1, \dots, n\}$ 
2    $S := \{1\}$ 
3   markiere Knoten 1
4   repeat
5     wähle einen Knoten  $u \in S$ 
6      $S := S - \{u\}$ 
7     for all  $(u, v) \in E$  do
8       if  $v$  ist nicht markiert then
9         markiere  $v$ 
10         $S := S \cup \{v\}$ 
11  until  $S = \emptyset$ 
12  if  $n$  ist markiert then accept else reject
```

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.

Komplexitätsbetrachtungen:

- REACH ist in Zeit n^3 entscheidbar.

1 Einführung

- REACH ist nichtdeterministisch in Platz $\log n$ entscheidbar (und daher deterministisch in Platz $\log^2 n$; Satz von Savitch).

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

Maximaler Fluß (MAXFLOW):

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \dots, n\}$, $E \subseteq V \times V$ und einer Kapazitätsfunktion $c : E \rightarrow \mathbb{N}$.

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

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

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

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

Komplexitätsbetrachtungen:

- MAXFLOW ist in Zeit n^5 lösbar.
- MAXFLOW ist in Platz n^2 lösbar.

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 $G = (U, V, E)$ mit $U = V = \{1, \dots, n\}$ und $E \subseteq U \times V$.

Gefragt: 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**.

Komplexitätsbetrachtungen:

- MATCHING ist in Zeit n^3 entscheidbar.
- MATCHING ist in Platz n^2 entscheidbar.

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 wir $\pi(n+1) = \pi(1)$ setzen.

Komplexitätsbetrachtungen:

- TSP ist in Zeit $n!$ lösbar (Ausprobieren aller Rundreisen).
- TSP ist in Platz n lösbar (mit demselben Algorithmus, der TSP in Zeit $n!$ löst).
- Durch dynamisches Programmieren^a lässt sich TSP in Zeit $n^2 \cdot 2^n$ lösen, der Platzverbrauch erhöht sich dabei jedoch auf $n \cdot 2^n$ (siehe Übungen).

^aHierzu berechnen wir für alle Teilmengen $S \subseteq \{2, \dots, n\}$ 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 $\Sigma \cup \{\sqcup, \triangleright\} \subseteq \Gamma$,
- $\delta : Q \times \Gamma^k \rightarrow (Q \cup \{q_h, q_{ja}, q_{nein}\}) \times (\Gamma \times \{L, R, N\})^k$ die **Überföhrungsfunktion** (q_h heißt **Haltezustand**, q_{ja} **akzeptierender** und q_{nein} **verwerfender Endzustand**
- und q_0 der **Startzustand**.

Befindet sich M im Zustand $q \in Q$ und stehen die Schreib-Lese-Köpfe auf Feldern mit den Inschriften a_1, \dots, a_k (a_i auf Band i), so geht M bei Ausführung der Anweisung $\delta : (q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$ in den Zustand q' über, ersetzt auf Band i das Symbol a_i durch a'_i und bewegt den Kopf gemäß D_i (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 δ , dass für jede Anweisung $(q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$ mit $a_i = \triangleright$ die Bedingung $a'_i = \triangleright$ und $D_i = R$ erfüllt ist (d.h. das Anfangszeichen \triangleright darf nicht durch ein anderes Zeichen überschrieben werden und der Kopf muss nach dem Lesen von \triangleright immer nach rechts bewegt werden).

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

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

Definition 3. Eine Konfiguration $K' = (q', u'_1, v'_1, \dots, u'_k, v'_k)$ heißt **Folgekonfiguration** von $K = (q, u_1, a_1 v_1, \dots, u_k, a_k v_k)$ (kurz: $K \xrightarrow{M} K'$), falls eine Anweisung

$$(q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$$

in δ und $b_1, \dots, b_k \in \Gamma$ existieren, so dass für $i = 1, \dots, k$ jeweils eine der folgenden drei Bedingungen gilt:

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

Wir schreiben $K \xrightarrow{M}^t K'$, falls Konfigurationen K_0, \dots, K_t existieren mit $K_0 = K$ und $K_t = K'$, sowie $K_i \xrightarrow{M} K_{i+1}$ für $i = 0, \dots, t - 1$. Die reflexive, transitive Hülle von \xrightarrow{M} bezeichnen wir mit \xrightarrow{M}^* , d.h. $K \xrightarrow{M}^* K'$ bedeutet, dass ein $t \geq 0$ existiert mit $K \xrightarrow{M}^t K'$.

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

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

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

Definition 6.

Eine DTM M **hält** bei Eingabe $x \in \Sigma^*$ (kurz: $M(x)$ hält), falls es eine Endkonfiguration $K = (q, u_1, v_1, \dots, u_k, v_k)$ gibt mit

$$K_x \xrightarrow[M]{*} K.$$

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

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

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 ein $i \geq 0$. Für $M(x) = \text{ja}$ sagen wir auch „ $M(x)$ akzeptiert“ und für $M(x) = \text{nein}$ „ $M(x)$ verwirft“.

Definition 7. 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** (oder **semi-entscheidbar**). Dagegen muss eine DTM, die eine Sprache L entscheidet, bei jeder Eingabe halten.

Definition 8. Sei $L \subseteq \Sigma^*$. Eine 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 heißt L **entscheidbar** (oder **rekursiv**).

Definition 9. Sei $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f : \Sigma^* \rightarrow \Sigma^*$ existiert, deren Bild $\text{range}(f) = \{f(x) \mid x \in \Sigma^*\}$ 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. 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

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

die Eigenschaft hat, dass für $(q', a'_1, D_1, \dots, a'_k, D_k) \in \delta(q, a_1, \dots, a_k)$ im Fall $a_i = \triangleright$ 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 $\delta(q, a_1, \dots, a_k) = (q', a'_1, D_1, \dots, a'_k, D_k)$ durch $(q', a'_1, D_1, \dots, a'_k, D_k) \in \delta(q, a_1, \dots, a_k)$ ersetzen (in beiden Fällen schreiben wir auch oft

$$\delta : (q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$$

oder einfach $(q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$.

Wir werden NTMs nur zum Erkennen von Sprachen (d.h. als Akzeptoren) und nicht zum Berechnen von Funktionen benutzen.

Definition 11.

- a) Sei M eine NTM. Wir sagen $M(x)$ **akzeptiert**, falls $M(x)$ nur endlich lange Rechnungen ausführt und eine akzeptierende Endkonfiguration K existiert mit $K_x \rightarrow^* K$.
- b) Akzeptiert $M(x)$ nicht und hat $M(x)$ nur endlich lange Rechnungen, so **verwirft** $M(x)$.
- c) Falls $M(x)$ unendlich lange Rechnungen ausführt, ist $M(x) = \uparrow$ (undefiniert).
- d) Die von M **akzeptierte Sprache** ist

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

- e) M **entscheidet** $L(M)$, falls M alle Eingaben $x \notin L(M)$ verwirft.