Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik

Vorlesung am 16.11.2010

INSTITUT FÜR THEORETISCHE INFORMATIK

Die Klasse $\mathcal{P}$

Die Klasse $\mathcal{P}$ist die Menge aller Sprachen L (Probleme), für die eine deterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial ist, d.h. es existiert ein Polynom p mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

Schwierigkeit von Entscheidungs und Optimierungsproblem

Karlsruher Institut für Technologie

Satz:
Falls es einen Algorithmus $\mathcal{A}$ gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

Schwierigkeit von Entscheidungs und Optimierungsproblem

Satz:
Falls es einen Algorithmus $\mathcal{A}$ gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

Beweis: Algorithmus, der das Optimierungsproblem löst.

Input: $G = (V, E)$, $c_{ij} = c(\{i, j\})$ für $i, j \in V := \{1, \ldots, n\}$, Algorithmus $\mathcal{A}$

Output: d_{ij} $(1 \leq i, j \leq n)$, so dass alle bis auf n der d_{ij}-Werte den Wert $\left(\max_{i,j} c_{ij}\right) + 1$ haben. Die restlichen n d_{ij}-Werte haben den Wert c_{ij} und geben genau die Kanten einer optimalen Tour an.

Algorithmus OPT-TOUR (als Beweis)

- berechne $m := \max_{1 \le i,j \le n} c_{ij}$;
  setze L(ow) := 0 und H(igh) := $n \cdot m$;
- solange $H - L > 1$ gilt, führe aus:

$$\left.\begin{array}{l} \text{falls } \mathcal{A}\left(n, c, \left\lceil \frac{1}{2}(H+L) \right\rceil\right) = \text{nein ist,} \\ \qquad \text{setze } L := \left\lceil \frac{1}{2}(H+L) \right\rceil + 1; \\ \text{sonst} \\ \qquad \text{setze } H := \left\lceil \frac{1}{2}(H+L) \right\rceil; \end{array}\right\} (* \text{ binäre Suche } *)$$

- falls $\mathcal{A}(n, c, L) =$„nein“ ist,setze $OPT := H$; sonst setze $OPT := L$;
- für $i = 1 \ldots n$ führe aus
- für $j = 1 \ldots n$ führe aus
- setze $R := c_{ij}$; $c_{ij} := m + 1$;
  falls $\mathcal{A}(n, c, OPT) =$ nein ist, setze $c_{ij} := R$;
- setze $d_{ij} = c_{ij}$;

Bemerkungen zum Algorithmus

Die Schleife in Schritt 2 bricht ab, und danach ist die Differenz $H - L$ gleich 1 oder 0, denn:

- Solange $H - L > 1$, ändert sich bei jedem Schleifendurchlauf einer der Werte H, L:
  - Für $H - L > 1$ gilt, dass $L \neq \left\lceil \frac{1}{2}(H + L) \right\rceil + 1$ und $H \neq \left\lceil \frac{1}{2}(H + L) \right\rceil$ ist.
- Die Differenz verkleinert sich also mit jedem Durchlauf
- Da H und L ganzzahlig sind, tritt der Fall $H - L \leq 1$ ein.

- Nach Abbruch der Schleife gilt $H - L \geq 0$:
  - Eine Differenz zwischen H und L von 0 kann genau durch Erhöhen von L bei einer aktuellen Differenz von 2 bzw. 3 erreicht werden
  - Eine Differenz zwischen H und L von 0 minimal ist (bei einer Differenz von weniger als 2 wird die Schleife nicht mehr betreten).

Laufzeit des Algorithmus

- In 2. wird $\mathcal{A}(n, c, k)$ etwa $\log(n \cdot m)$–mal aufgerufen
- In 4. wird $\mathcal{A}(n, c, OPT)$ etwa n^2–mal aufgerufen.
- Es finden also $\mathcal{O}(n^2 + \log(nm))$ Aufrufe von $\mathcal{A}$ statt.
- Die Inputlänge ist $\mathcal{O}(n^2 \cdot \log(\max c_{ij}))$.
- Da $\mathcal{A}$ polynomial ist, ist dies also auch OPT-TOUR.

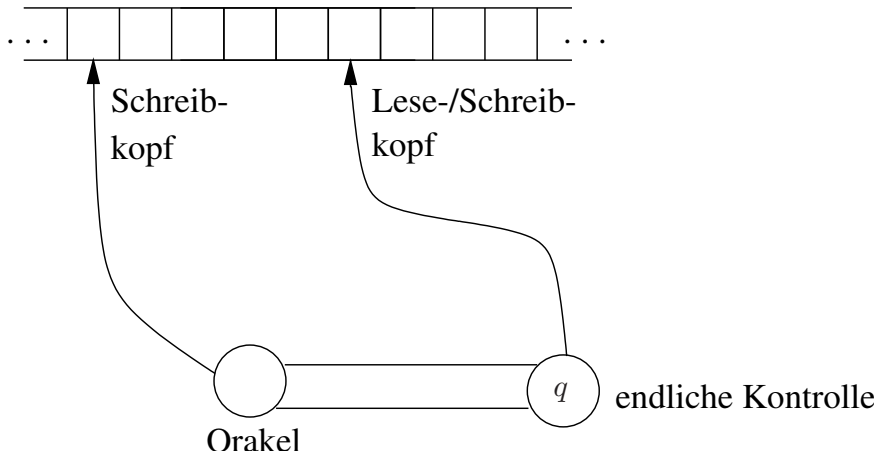
Kapitel

- **Nichtdeterministische Turingmaschinen**
- **Die Klasse NP**

Die Nichtdeterministische Turingmaschine

- Bei der nichtdeterministischen Turing-Maschine wird die Übergangsfunktion δ zu einer Relation erweitert.
- Dies ermöglicht Wahlmöglichkeiten und ε–Übergänge (vergleiche endliche Automaten).
- Wir betrachten ein äquivalentes Modell einer nichtdeterministischen Turing-Maschine (NTM), die auf einem **Orakel** basiert
- Dies kommt der Intuition näher.

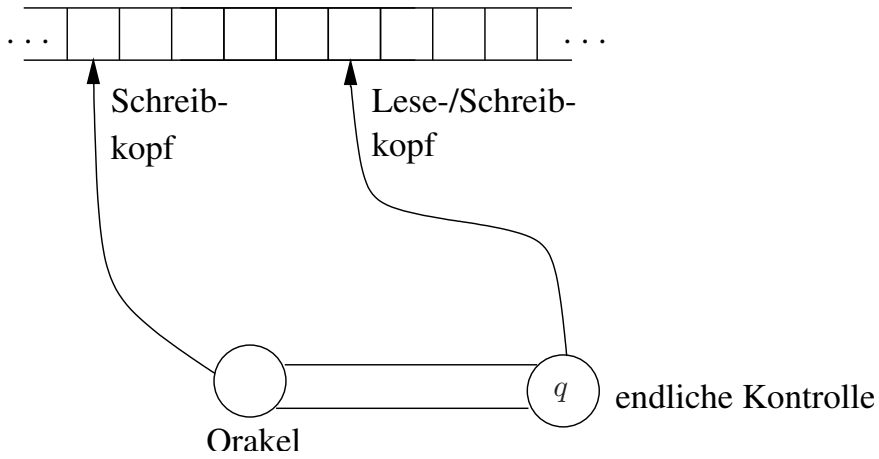
Die Nichtdeterministische Turingmaschine



Nichtdeterministische Turingmaschinen (NTM)

- werden analog zur DTM durch das Oktupel $(Q, \Sigma, \sqcup, \Gamma, s, \delta, q_J, q_N)$ beschrieben.
- haben zusätzlich zu der endlichen Kontrolle mit dem Lese-/Schreibkopf ein **Orakelmodul** mit einem eigenen Schreibkopf
- NTMs haben genau zwei Endzustände q_J und q_N, wobei q_J der akzeptierende Endzustand ist.

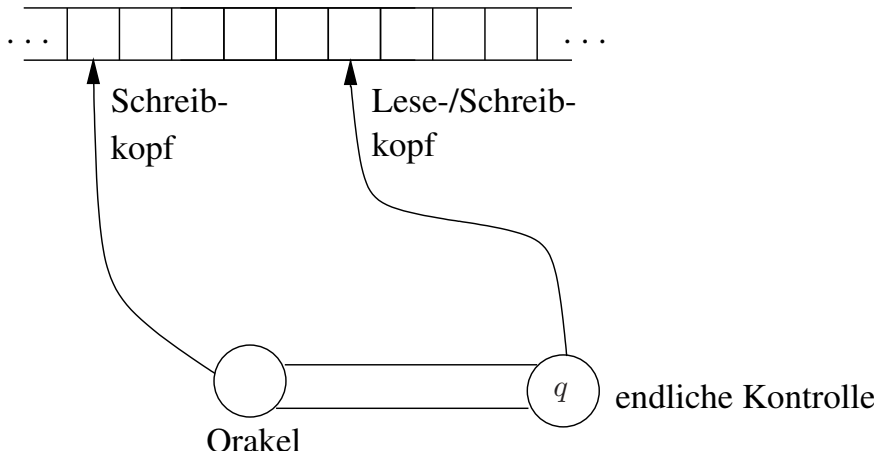
Die Nichtdeterministische Turingmaschine



Berechnung bei einer nichtdeterministischen Turing-Maschine

- **1. Stufe:**
  - Das Orakelmodul weist seinen Schreibkopf an, Schritt für Schritt entweder ein Symbol zu schreiben und nach links zu gehen oder anzuhalten.
  - Falls der Schreibkopf anhält, wird das Orakelmodul inaktiv, und die endliche Zustandskontrolle wird aktiv.
- **2. Stufe:**
  - Ab jetzt genau wie bei DTM.
  - Das Orakelmodul und sein Schreibkopf sind nicht weiter beteiligt.

Die Nichtdeterministische Turingmaschine



- Eine nichtdeterministische Turing-Maschine $\mathcal{M}$ **akzeptiert** ein Wort $x \in \Sigma^*$ genau dann, wenn es eine Berechnung gibt, die in q_J endet.
- $\mathcal{M}$ akzeptiert die Sprache $L \subseteq \Sigma^*$ genau dann, wenn sie gerade die Wörter aus L akzeptiert.

Übertragung auf Entscheidungsprobleme Π

Die Eingabe ist ein Wort aus Σ^*, zum Beispiel eine Kodierung eines Problembeispiels $I \in D_\Pi$.

- **1. Stufe:** Es wird ein Orakel aus Γ^* berechnet, zum Beispiel ein Lösungsbeispiel für I, also ein Indikator, ob $I \in J_\Pi$ oder $I \in N_\Pi$ gilt.
- **1. Stufe:** Hier wird nun dieser Lösungsvorschlag überprüft, d.h. es wird geprüft ob $I \in J_\Pi$.

Beispiel TSP

- **1. Stufe:** Es wird zum Beispiel eine Permutation σ auf der Knotenmenge V vorgeschlagen. D.h. $(\sigma(1), \ldots, \sigma(n))$, $G = (V, E)$, c und k bilden die Eingabe.
- **1. Stufe:** Es wird nun überprüft, ob $\sigma(V)$ eine Tour in G enthält, deren Länge bezüglich c nicht größer als k ist.

Bemerkungen zur NTM

- Das Orakel kann ein beliebiges Wort aus Γ^* sein.
- Darum muss in der Überprüfungsphase (2.Stufe) zunächst geprüft werden, ob das Orakel ein zulässiges Lösungsbeispiel ist.
- Ist dies nicht der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort „Nein“ beendet werden.

- Jede NTM $\mathcal{M}$ hat zu einer gegebenen Eingabe x eine unendliche Anzahl möglicher Berechnungen, eine zu jedem Orakel aus Γ^*.
- Endet mindestens eine in q_J, so wird x akzeptiert.

Zeitkomplexität für NTM

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine $\mathcal{M}$ benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die $\mathcal{M}$ in den Zustand q_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine $\mathcal{M}$ ist definiert durch

$$T_{\mathcal{M}}(n) := \max\left(\{1\} \cup \left\{ m \;\middle|\; \begin{array}{l}\text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so}\\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,}\\ \text{um } x \text{ zu akzeptieren, } m \text{ ist}\end{array} \right\}\right)$$

Zeitkomplexität für NTM

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine $\mathcal{M}$ benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die $\mathcal{M}$ in den Zustand q_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine $\mathcal{M}$ ist definiert durch

$$T_{\mathcal{M}}(n) := \max\left(\{1\} \cup \left\{ m \;\middle|\; \begin{array}{l}\text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so}\\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,}\\ \text{um } x \text{ zu akzeptieren, } m \text{ ist}\end{array} \right\}\right)$$

Bemerkung 1

- Zur Berechnung von $T_{\mathcal{M}}(n)$ wird für jedes $x \in L_{\mathcal{M}}$ mit $|x| = n$ die kürzeste akzeptierende Berechnung betrachtet.
- Anschließend wir von diesen kürzesten die längste bestimmt.
- Somit ergibt sich eine *worst–case* Abschätzung.

Zeitkomplexität für NTM

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine $\mathcal{M}$ benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die $\mathcal{M}$ in den Zustand q_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine $\mathcal{M}$ ist definiert durch

$$T_{\mathcal{M}}(n) := \max\left(\{1\} \cup \left\{ m \;\middle|\; \begin{array}{l}\text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so}\\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,}\\ \text{um } x \text{ zu akzeptieren, } m \text{ ist}\end{array} \right\}\right)$$

Bemerkung 2

- Die Zeitkomplexität hängt nur von der Anzahl der Schritte ab, die bei einer akzeptierenden Berechnung auftreten.
- Hierbei umfasst die Anzahl der Schritte auch die Schritte der Orakelphase.
- Per Konvention ist $T_{\mathcal{M}}(n) = 1$, falls es keine Eingabe x der Länge n gibt, die von $\mathcal{M}$ akzeptiert wird.

Die Klasse NP

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L, für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

($\mathcal{NP}$ steht für **nichtdeterministisch polynomial**.)

Die Klasse NP

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L, für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

($\mathcal{NP}$ steht für **nichtdeterministisch polynomial**.)

Bemerkungen

- Alle Sprachen in $\mathcal{NP}$ sind entscheidbar.
- Informell ausgedrückt: Π gehört zu $\mathcal{NP}$, falls Π folgende Eigenschaft hat: Ist die Antwort bei Eingabe eines Beispiels I von Π Ja, dann kann die Korrektheit der Antwort in polynomialer Zeit überprüft werden.

Die Klasse NP

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L, für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

($\mathcal{NP}$ steht für **nichtdeterministisch polynomial**.)

Beispiel: TSP$\in \mathcal{NP}$:

Denn zu gegebenem $G = (V, E), c, k$ und einer festen Permutation σ auf V kann man in $O(|V| \cdot \log C)$, wobei C die größte vorkommende Zahl ist, überprüft werden, ob

$$\sum_{i=1}^{n-1} c(\{\sigma(i), \sigma(i+1)\}) + c(\{\sigma(n), \sigma(1)\}) \leq k$$

gilt.

Kapitel

Karlsruher Institut für Technologie

NP-vollständige Probleme

Große Frage der Theoretischen Informatik

Karlsruher Institut für Technologie

- Trivialerweise gilt: $\mathcal{P} \subseteq \mathcal{NP}$.
- **Frage:** Gilt $\mathcal{P} \subset \mathcal{NP}$ oder $\mathcal{P} = \mathcal{NP}$?
- Die Vermutung ist, dass $\mathcal{P} \neq \mathcal{NP}$ gilt.

- Dazu betrachten wir Probleme, die zu den schwersten Problemen in $\mathcal{NP}$ gehören.

- Dabei ist am schwersten im folgenden Sinne gemeint:
- Wenn ein solches Problem trotzdem in $\mathcal{P}$ liegt, so kann man folgern, dass alle Probleme aus $\mathcal{NP}$ in $\mathcal{P}$ liegen, d.h. $\mathcal{P} = \mathcal{NP}$.

- Diese Probleme sind also Kandidaten, um $\mathcal{P}$ und $\mathcal{NP}$ zu trennen.
- Es wird sich zeigen, dass alle diese schwersten Probleme im wesentlichen gleich schwer sind.

NP-Vollständigkeit

Eine **polynomiale Transformation** einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
- für alle $x \in \Sigma_1^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Wir schreiben dann $L_1 \propto L_2$ (L_1 ist polynomial transformierbar in L_2).

Eine Sprache L heißt $\mathcal{NP}$**–vollständig** , falls gilt:

- $L \in \mathcal{NP}$ und
- für alle $L' \in \mathcal{NP}$ gilt $L' \propto L$.

NP-Vollständigkeit

Karlsruher Institut für Technologie

Wir formulieren nun die Begriffe polynomial transformierbar und $\mathcal{NP}$-vollständig für Entscheidungsprobleme.

Ein Entscheidungsproblem Π_1 ist **polynomial transformierbar** in das Entscheidungsproblem Π_2, wenn eine Funktion $f\colon D_{\Pi_1} \to D_{\Pi_2}$ existiert mit folgenden Eigenschaften:

- f ist durch einen polynomialen Algorithmus berechenbar;
- $\forall\, I \in D_{\Pi_1} : I \in J_{\Pi_1} \iff f(I) \in J_{\Pi_2}$.

Wir schreiben dann $\Pi_1 \propto \Pi_2$.

Ein Entscheidungsproblem Π heißt **$\mathcal{NP}$-vollständig**, falls gilt:

- $\Pi \in \mathcal{NP}$ und
- für alle $\Pi' \in \mathcal{NP}$ gilt $\Pi' \propto \Pi$.

Transitivität der poly. Transformation

Eine **polynomiale Transformation** einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
- für alle $x \in \Sigma_1^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Wir schreiben dann $L_1 \propto L_2$ (L_1 ist polynomial transformierbar in L_2).

Lemma. $\propto$ ist transitiv, d.h. aus $L_1 \propto L_2$ und $L_2 \propto L_3$ folgt $L_1 \propto L_3$.

Beweis. Die Hintereinanderausführung zweier polynomialer Transformationen ist wieder eine polynomiale Transformation.

Korollar

Korollar. Falls $L_1, L_2 \in \mathcal{NP}$, $L_1 \propto L_2$ und L_1 $\mathcal{NP}$-vollständig, dann ist auch L_2 $\mathcal{NP}$-vollständig.

Bedeutung.
Um also zu zeigen, dass ein Entscheidungsproblem Π $\mathcal{NP}$-vollständig ist, gehen wir folgendermaßen vor. Wir beweisen:

- $\Pi \in \mathcal{NP}$
- für ein bekanntes $\mathcal{NP}$-vollständiges Problem Π' gilt: $\Pi' \propto \Pi$.

Problem.

- Wir wissen noch für kein einziges Problem, dass es $\mathcal{NP}$-vollständig ist.
- Das erste $\mathcal{NP}$-vollständige Problem ist das Erfüllbarkeitsproblem SAT (satisfiability).

Das Problem SAT (satisfiability)

Sei $U = \{u_1, \ldots, u_m\}$ eine Menge von booleschen Variablen
Es heißen $u_i, \overline{u_i}$ Literale.
Eine Wahrheitsbelegung für U ist eine Funktion $t\colon U \to \{\texttt{wahr}, \texttt{falsch}\}$.
Eine **Klausel** ist ein Boole'scher Ausdruck der Form

$$y_1 \vee \ldots \vee y_s \quad \text{mit} \quad y_i \in \underbrace{\{u_1, \ldots, u_m\} \cup \{\overline{u_1}, \ldots, \overline{u_m}\}}_{\text{Literalmenge}} \cup \{\texttt{wahr}, \texttt{falsch}\}$$

Problem SAT
Gegeben: Menge U von Variablen, Menge C von Klauseln über U.
Frage: Existiert eine Wahrheitsbelegung von U, so dass C erfüllt wird, d.h. dass alle Klauseln aus C den Wahrheitswert `wahr` annehmen?

Beispiel: $U = \{u_1, u_2\}$ mit $C = \{u_1 \vee \overline{u_2},\ \overline{u_1} \vee u_2\}$ ist Ja-Beispiel von SAT. Mit der Wahrheitsbelegung $t(u_1) = t(u_2) = \texttt{wahr}$ wird C erfüllt.

Der Satz von Cook (Steven Cook, 1971)

SAT ist $\mathcal{NP}$-vollständig.