

Theoretische Grundlagen der Informatik

Vorlesung am 22. November 2011

INSTITUT FÜR THEORETISCHE INFORMATIK



Fragestellung bisher:

- Ist eine Sprache L entscheidbar oder nicht?
- Ist eine Funktion berechenbar oder nicht?
- Benutzung von deterministischen Turing-Maschinen.

In diesem Kapitel:

- Wie effizient kann ein Problem gelöst werden?
- Betrachtung von nichtdeterministischen Turing-Maschinen.

Frage (P vs. NP):

Gibt es einen wesentlichen Effizienzgewinn beim Übergang der deterministischen Turing-Maschine zur nichtdeterministischen Turing-Maschine?

- **Sprachen**
- **Probleme**
- **Zeitkomplexität**

Beispiel: Traveling Salesman Problem (TSP)

Gegeben sei ein vollständiger Graph $G = (V, E, c)$, d.h.

- $V := \{1, \dots, n\}$
- $E := \{\{u, v\} \mid u, v \in V, u \neq v\}$
- $c: E \rightarrow \mathbb{Z}^+$.

Wir betrachten folgende Problemvarianten

- **Optimierungsproblem:**
Gesucht ist eine Tour (Rundreise), die alle Elemente aus V enthält und minimale Gesamtlänge unter allen solchen Touren hat.
- **Optimalwertproblem:**
Gesucht ist die Länge einer minimalen Tour.
- **Entscheidungsproblem:**
Gegeben sei zusätzlich auch ein Parameter $k \in \mathbb{Z}^+$. Die Frage ist nun: Gibt es eine Tour, deren Länge höchstens k ist?

Wir betrachten folgende Problemvarianten

- **Optimierungsproblem:**

Gesucht ist eine Tour (Rundreise), die alle Elemente aus V enthält und minimale Gesamtlänge unter allen solchen Touren hat.

- **Optimalwertproblem:**

Gesucht ist die Länge einer minimalen Tour.

- **Entscheidungsproblem:**

Gegeben sei zusätzlich auch ein Parameter $k \in \mathbb{Z}^+$. Die Frage ist nun: Gibt es eine Tour, deren Länge höchstens k ist?

Bemerkung:

- Mit einer Lösung des Optimierungsproblems kann man leicht auch das Optimalwertproblem und das Entscheidungsproblem lösen.
- Mit einer Lösung des Optimalwertproblems kann man leicht auch das Entscheidungsproblem lösen.

Definition: Problem

Ein **Problem** Π ist gegeben durch:

- eine allgemeine Beschreibung aller vorkommenden Parameter;
- eine genaue Beschreibung der Eigenschaften, die die Lösung haben soll.

Ein **Problembispiel** / (**Instanz**) von Π erhalten wir, indem wir die Parameter von Π festlegen.

Definition: Kodierungsschema

- Wir interessieren uns für die **Laufzeit** von Algorithmen.
- Diese wird in der Größe des Problems gemessen.

Die Größe eines Problems ist abhängig von der Beschreibung oder Kodierung der Problembeispiele

- Ein **Kodierungsschema** s ordnet jedem Problembeispiel eines Problems eine Zeichenkette oder *Kodierung* über einem Alphabet Σ zu.
- Die **Inputlänge** eines Problembeispiels ist die Anzahl der Symbole seiner Kodierung.

Es gibt verschiedene Kodierungsschemata für ein bestimmtes Problem.

Beispiel:

- Zahlen können dezimal, binär, unär, usw. kodiert werden.
- Die Inputlänge von 5127 beträgt dann 4 für dezimal, 13 für binär und 5127 für unär.

Wir werden uns auf vernünftige Schemata festlegen:

- Die Kodierung eines Problembeispiels sollte keine überflüssigen Informationen enthalten.
- Zahlen sollen binär (oder k -är für $k \neq 1$) kodiert sein.

Dies bedeutet, die Kodierungslänge

- einer ganzen Zahl n ist $\lfloor \log_k |n| + 1 \rfloor + 1 =: \langle n \rangle$
(eine 1 benötigt man für das Vorzeichen);
- einer rationalen Zahl $r = \frac{p}{q}$ ist $\langle r \rangle = \langle p \rangle + \langle q \rangle$;
- eines Vektors $X = (x_1, \dots, x_n)$ ist $\langle X \rangle := \sum_{i=1}^n \langle x_i \rangle$;
- einer Matrix $A \in \mathbb{Q}^{m \times n}$ ist $\langle A \rangle := \sum_{i=1}^m \sum_{j=1}^n \langle a_{ij} \rangle$.
- eines Graphen $G = (V, E)$ kann zum Beispiel durch die Kodierung seiner *Adjazenzmatrix*, die eines gewichteten Graphen durch die Kodierung der *Gewichtsmatrix* beschrieben werden.

Äquivalenz von Kodierungsschemata

Zwei Kodierungsschemata s_1, s_2 heißen **äquivalent** bezüglich eines Problems Π , falls es Polynome p_1, p_2 gibt, so dass gilt:

$$(|s_1(I)| = n \Rightarrow |s_2(I)| \leq p_2(n)) \text{ und } (|s_2(I)| = m \Rightarrow |s_1(I)| \leq p_1(m))$$

für alle Problembeispiele I von Π .

- Ein Entscheidungsproblem Π können wir als Klasse von Problembeispielen D_{Π} auffassen.
- Eine Teilmenge dieser Klasse ist $J_{\Pi} \subseteq D_{\Pi}$, die Klasse der **Ja-Beispiele**, d.h. die Problembeispiele deren Antwort Ja ist.
- Der Rest der Klasse $N_{\Pi} \subseteq D_{\Pi}$ ist die Klasse der **Nein-Beispiele**.

Ein Problem Π und ein Kodierungsschema $s: D_{\Pi} \rightarrow \Sigma^*$ zerlegen Σ^* in drei Klassen:

- Wörter aus Σ^* , die *nicht* Kodierung eines Beispiels aus D_{Π} sind,
- Wörter aus Σ^* , die Kodierung eines Beispiels $I \in N_{\Pi}$ sind,
- Wörter aus Σ^* , die Kodierung eines Beispiels $I \in J_{\Pi}$ sind.

Die dritte Klasse ist die Sprache, die zu Π im Kodierungsschema s korrespondiert.

Die zu einem Problem Π und einem Kodierungsschema s **zugehörige Sprache** ist

$$L[\Pi, s] := \left\{ x \in \Sigma^* \mid \begin{array}{l} \Sigma \text{ ist das Alphabet zu } s \text{ und } x \text{ ist Kodierung} \\ \text{eines Ja-Beispiels } I \text{ von } \Pi \text{ unter } s, \text{ d.h. } I \in J_{\Pi} \end{array} \right\}$$

- Wir betrachten im folgenden deterministische Turing-Maschinen mit zwei Endzuständen q_J, q_N , wobei q_J akzeptierender Endzustand ist.
- Dann wird die Sprache $L_{\mathcal{M}}$ akzeptiert von der Turing-Maschine \mathcal{M} , falls

$$L_{\mathcal{M}} = \{x \in \Sigma^* \mid \mathcal{M} \text{ akzeptiert } x\} .$$

- Eine deterministische Turing-Maschine \mathcal{M} **löst** ein Entscheidungsproblem Π unter einem Kodierungsschema s , falls \mathcal{M} bei jeder Eingabe über dem Eingabe-Alphabet in einem Endzustand endet und $L_{\mathcal{M}} = L[\Pi, s]$ ist.

Für eine deterministische Turing-Maschine \mathcal{M} , die für alle Eingaben über dem Eingabe-Alphabet Σ hält, ist die **Zeitkomplexitätsfunktion**

$T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ definiert durch

$$T_{\mathcal{M}}(n) = \max \left\{ m \mid \begin{array}{l} \text{es gibt eine Eingabe } x \in \Sigma^* \text{ mit } |x| = n, \text{ so} \\ \text{dass die Berechnung von } \mathcal{M} \text{ bei Eingabe } x \\ m \text{ Berechnungsschritte (Übergänge) benötigt,} \\ \text{bis ein Endzustand erreicht wird} \end{array} \right\}$$

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

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 $i, j \in V := \{1, \dots, n\}$,
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 $m := \max_{1 \leq i, j \leq n} c_{ij}$;
setze $L(\text{ow}) := 0$ und $H(\text{igh}) := n \cdot m$;
- solange $H - L > 1$ gilt, führe aus:
 - falls $\mathcal{A}\left(n, c, \left\lceil \frac{1}{2}(H + L) \right\rceil\right) = \text{nein}$ ist,
setze $L := \left\lceil \frac{1}{2}(H + L) \right\rceil + 1$;
 - sonst
setze $H := \left\lceil \frac{1}{2}(H + L) \right\rceil$;

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

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 $H \neq \left\lfloor \frac{1}{2}(H + L) \right\rfloor$ ist.
- Die Differenz verkleinert sich also mit jedem Durchlauf
- Da H und L ganzzahlig sind, tritt der Fall $H - L \leq 1$ ein.

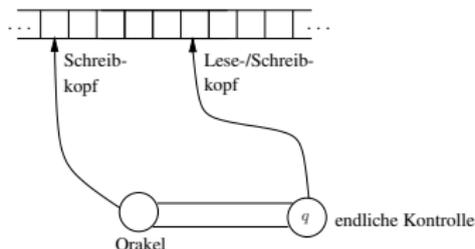
- Nach Abbruch der Schleife gilt $H - L \geq 0$:
 - Eine Differenz zwischen H und L von 0 kann genau durch Erhöhen von L bei einer aktuellen Differenz von 2 bzw. 3 erreicht werden
 - Eine Differenz zwischen H und L von 0 minimal ist (bei einer Differenz von weniger als 2 wird die Schleife nicht mehr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

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

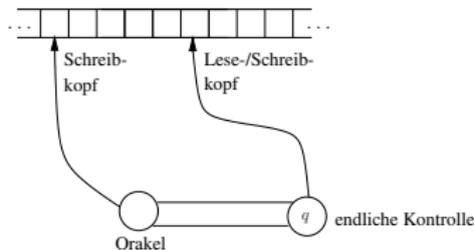
- Bei der nichtdeterministischen Turing-Maschine wird die Übergangsfunktion δ zu einer Relation erweitert.
- Dies ermöglicht Wahlmöglichkeiten und ε -Übergänge (vergleiche endliche Automaten).
- Wir betrachten ein äquivalentes Modell einer nichtdeterministischen Turing-Maschine (NTM), die auf einem **Orakel**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



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

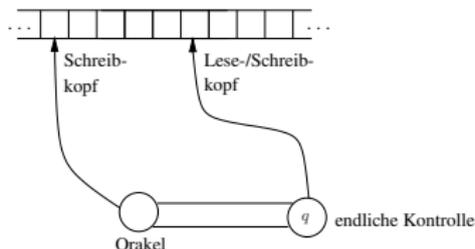
■ 1. Stufe:

- Das Orakelmodul schreibt zunächst ein Trennzeichen in die Zelle direkt vor der Eingabe
- Dann weist es 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_f endet.
- \mathcal{M} akzeptiert die Sprache $L \subseteq \Sigma^*$ genau dann, wenn sie gerade die Wörter aus L akzeptiert.

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.
- **2. 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. $(\sigma(1), \dots, \sigma(n))$, $G = (V, E)$, c und k bilden die Eingabe.
- **2. Stufe:** Es wird nun überprüft, ob $\sigma(V)$ eine Tour in G enthält, deren Länge bezüglich c nicht größer als k ist.

- 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.

- 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_f überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine \mathcal{M} ist definiert durch

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

- 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_f überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine \mathcal{M} ist definiert durch

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

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 wird von diesen kürzesten die längste bestimmt.
- Somit ergibt sich eine *worst-case* Abschätzung.

- 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_f überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine \mathcal{M} ist definiert durch

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

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

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.