Technische Universität Dresden

Theoretische Informatik und Logik

7. Vorlesung: Beziehungen zwischen Komplexitätsklassen / Effizient lösbare Probleme

Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 7. Mai 2026

Rückblick

Die wichtigsten Ressourcen zur Messung von Komplexität sind Rechenzeit und Speicher.

Die wichtigsten deterministischen Komplexitätsklassen sind:

$$\mathsf{P} = \mathsf{PTime} = \bigcup_{d \geq 1} \mathsf{DTime}(n^d) \qquad \text{polynomielle Zeit}$$

$$\mathsf{Exp} = \mathsf{ExpTime} = \bigcup_{d \geq 1} \mathsf{DTime}(2^{n^d}) \qquad \text{exponentielle Zeit}^*$$

$$\mathsf{L} = \mathsf{LogSpace} = \mathsf{DSpace}(\log n) \qquad \text{logarithmischer Speicher}$$

$$\mathsf{PSpace} = \bigcup_{d \geq 1} \mathsf{DSpace}(n^d) \qquad \text{polynomieller Speicher}$$

Robustheit von Zeitklassen

Zwei wichtige Erkenntnisse zur Robustheit von Zeitklassen:

Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine zeitbeschränkte Mehrband-TM lösen kann, sofern mindestens lineare Zeit erlaubt ist (Linear Speedup Theorem).

Anmerkung: Wenn nicht wenigstens lineare Zeit zur Verfügung steht, dann kann die TM nicht einmal die Eingabe lesen. Das ergibt also bei einer herkömmlichen TM wenig Sinn.

Robustheit von Zeitklassen

Zwei wichtige Erkenntnisse zur Robustheit von Zeitklassen:

Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine zeitbeschränkte Mehrband-TM lösen kann, sofern mindestens lineare Zeit erlaubt ist (Linear Speedup Theorem).

Anmerkung: Wenn nicht wenigstens lineare Zeit zur Verfügung steht, dann kann die TM nicht einmal die Eingabe lesen. Das ergibt also bei einer herkömmlichen TM wenig Sinn.

Die Anzahl der Bänder hat lediglich einen polynomiellen (quadratischen) Einfluss auf die Probleme, die eine zeitbeschränkte TM lösen kann.

Das hatten wir in Formale Systeme durch Simulation mehrerer Bänder auf einem gezeigt.

Robustheit von Speicherklassen

Bei speicherbeschränkten TMs ist die Situation sogar etwas einfacher:

Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine speicherbeschränkte TM lösen kann.

Robustheit von Speicherklassen

Bei speicherbeschränkten TMs ist die Situation sogar etwas einfacher:

> Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine speicherbeschränkte TM lösen kann.

Beweis: Ähnlich zu Linear Speedup, aber viel einfacher.

Robustheit von Speicherklassen

Bei speicherbeschränkten TMs ist die Situation sogar etwas einfacher:

Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine speicherbeschränkte TM lösen kann.

Beweis: Ähnlich zu Linear Speedup, aber viel einfacher.

Die Anzahl der Bänder hat keinen Einfluss auf die Probleme, die eine speicherbeschränkte TM lösen kann.

Robustheit von Speicherklassen

Bei speicherbeschränkten TMs ist die Situation sogar etwas einfacher:

Konstante Faktoren haben keinen Einfluss auf die Probleme, die eine speicherbeschränkte TM lösen kann.

Beweis: Ähnlich zu Linear Speedup, aber viel einfacher.

Die Anzahl der Bänder hat keinen Einfluss auf die Probleme, die eine speicherbeschränkte TM lösen kann.

Beweis: Reduktion von k Bändern auf 1 Band wie gehabt, kombiniert mit einer $1/k$ Speicherreduktion.

Beziehung von Zeit und Raum (1)

Ist die Berechnungszeit beschränkt, so kann auch nur beschränkt viel Speicher genutzt werden:

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\mathrm{DTIME}(f) \subseteq \mathrm{DSPACE}(f).$$

Beweis: Die TM benötigt immer mindestens einen Schritt um eine zusätzliche Speicherstelle zu beschreiben. □

Beziehung von Zeit und Raum (1)

Ist die Berechnungszeit beschränkt, so kann auch nur beschränkt viel Speicher genutzt werden:

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\text{DTIME}(f) \subseteq \text{DSPACE}(f).$$

Beweis: Die TM benötigt immer mindestens einen Schritt um eine zusätzliche Speicherstelle zu beschreiben. □

Daraus folgt zum Beispiel PTime ⊆ PSpace.

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\mathsf{DSPACE}(f) \subseteq \mathsf{DTIME}(2^{O(f)}).$$

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\mathrm{DSPACE}(f) \subseteq \mathrm{DTIME}(2^{O(f)}).$$

Beweis: Gegeben sei eine Eingabe w der Länge $|w| = n$.

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\mathsf{DSPACE}(f) \subseteq \mathsf{DTIME}(2^{O(f)}).$$

Beweis: Gegeben sei eine Eingabe w der Länge $|w| = n$.

- Es gibt $|\Gamma|^{f(n)} = 2^{\log_2(|\Gamma|) \cdot f(n)}$ Speicherbelegungen der Länge $f(n)$

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\text{DSPACE}(f) \subseteq \text{DTIME}(2^{O(f)}).$$

Beweis: Gegeben sei eine Eingabe w der Länge $|w| = n$.

- Es gibt $|\Gamma|^{f(n)} = 2^{\log_2(|\Gamma|) \cdot f(n)}$ Speicherbelegungen der Länge $f(n)$
- Hinzu kommen $f(n)$ mögliche Kopfpositionen und $|Q|$ Zustände

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\text{DSPACE}(f) \subseteq \text{DTIME}(2^{O(f)}).$$

Beweis: Gegeben sei eine Eingabe w der Länge $|w| = n$.

- Es gibt $|\Gamma|^{f(n)} = 2^{\log_2(|\Gamma|)\cdot f(n)}$ Speicherbelegungen der Länge $f(n)$
- Hinzu kommen $f(n)$ mögliche Kopfpositionen und $|Q|$ Zustände
- Es gibt also $|Q| \cdot f(n) \cdot 2^{\log_2(|\Gamma|)\cdot f(n)} \in O(2^{O(f)})$ TM-Konfigurationen.

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\text{DSPACE}(f) \subseteq \text{DTIME}(2^{O(f)}).$$

Beweis: Gegeben sei eine Eingabe w der Länge $|w| = n$.

- Es gibt $|\Gamma|^{f(n)} = 2^{\log_2(|\Gamma|) \cdot f(n)}$ Speicherbelegungen der Länge $f(n)$
- Hinzu kommen $f(n)$ mögliche Kopfpositionen und $|Q|$ Zustände
- Es gibt also $|Q| \cdot f(n) \cdot 2^{\log_2(|\Gamma|) \cdot f(n)} \in O(2^{O(f)})$ TM-Konfigurationen.
- Aus diesen kann man für eine gegebene Eingabe in polynomieller Zeit einen Konfigurationsgraph berechnen, in dem (gerichtete) Kanten die möglichen Übergänge darstellen.

Beziehung von Zeit und Raum (2)

Andererseits ist Speicher mächtiger als Zeit, da man Speicher mehrfach verwenden kann (Zeit leider nicht):

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt

$$\text{DSPACE}(f) \subseteq \text{DTIME}(2^{O(f)}).$$

Beweis: Gegeben sei eine Eingabe w der Länge $|w| = n$.

- Es gibt $|\Gamma|^{f(n)} = 2^{\log_2(|\Gamma|) \cdot f(n)}$ Speicherbelegungen der Länge $f(n)$
- Hinzu kommen $f(n)$ mögliche Kopfpositionen und $|Q|$ Zustände
- Es gibt also $|Q| \cdot f(n) \cdot 2^{\log_2(|\Gamma|) \cdot f(n)} \in O(2^{O(f)})$ TM-Konfigurationen.
- Aus diesen kann man für eine gegebene Eingabe in polynomieller Zeit einen Konfigurationsgraph berechnen, in dem (gerichtete) Kanten die möglichen Übergänge darstellen.
- Daraus kann man die Akzeptanz der Eingabe in polynomieller Zeit ermitteln („Ist von der Startkonfiguration aus eine akzeptierende Endkonfiguration erreichbar?“).

Damit hat man das Wortproblem in Zeit $O(2^{O(f)})$ entschieden. □

Nichtdeterministische Komplexitätsklassen

Ressourcen nichtdeterministischer TMs

Bei NTMs gibt es viele mögliche Berechnungspfade.

$\rightsquigarrow$ Welche Pfade meinen wir, wenn wir Ressourcen beschränken?

Ressourcen nichtdeterministischer TMs

Bei NTMs gibt es viele mögliche Berechnungspfade.
$\rightsquigarrow$ Welche Pfade meinen wir, wenn wir Ressourcen beschränken?

– Alle!

Sei $f : \mathbb{N} \to \mathbb{R}$ eine Funktion und $\mathcal{M}$ eine nichtdeterministische TM.

- $\mathcal{M}$ heißt $O(f)$-zeitbeschränkt wenn es eine Funktion $g \in O(f)$ gibt, so dass $\mathcal{M}$ für eine beliebige Eingabe $w \in \Sigma^*$ auf jedem Berechnungspfad nach maximal $g(|w|)$ Schritten anhält.
- $\mathcal{M}$ heißt $O(f)$-speicherbeschränkt wenn es eine Funktion $g \in O(f)$ gibt, so dass $\mathcal{M}$ für eine beliebige Eingabe $w \in \Sigma^*$ hält und zuvor auf jedem Berechnungspfad maximal $g(|w|)$ Speicherzellen verwendet.

Eine zeit- oder speicherbeschränkte NTM muss also auch auf erfolglosen Pfaden („falsch geratene Übergänge“) garantiert innerhalb der Ressourcengrenzen halten.

Zeit und Raum, nichtdeterministisch

Die entsprechenden Sprachklassen werden genau wie bei deterministischen TMs definiert:

Sei $f : \mathbb{N} \to \mathbb{R}$ eine Funktion.

- NTIME (f(n)) ist die Klasse aller Sprachen **L**, welche durch eine $O(f)$-zeitbeschränkte NTM entschieden werden können.
- NSPACE (f(n)) ist die Klasse aller Sprachen **L**, welche durch eine $O(f)$-speicherbeschränkte NTM entschieden werden können.

Nichtdeterministische Komplextitätsklassen

Auch hier beschränken wir uns auf einige wichtige Fälle:

$$\mathsf{NP} = \mathsf{NPTime} = \bigcup_{d \geq 1} \mathsf{NTime}(n^d)$$ nichtdet. polynomielle Zeit

$$\mathsf{NExp} = \mathsf{NExpTime} = \bigcup_{d \geq 1} \mathsf{NTime}(2^{n^d})$$ nichtdet. exponentielle Zeit

$$\mathsf{NL} = \mathsf{NLogSpace} = \mathsf{NSpace}(\log n)$$ nichtdet. logarithmischer Speicher

$$\mathsf{NPSpace} = \bigcup_{d \geq 1} \mathsf{NSpace}(n^d)$$ nichtdet. polynomieller Speicher

Nichtdeterministische Komplextitätsklassen

Auch hier beschränken wir uns auf einige wichtige Fälle:

$$\mathsf{NP} = \mathsf{NPTime} = \bigcup_{d \geq 1} \mathsf{NTime}(n^d)$$ nichtdet. polynomielle Zeit

$$\mathsf{NExp} = \mathsf{NExpTime} = \bigcup_{d \geq 1} \mathsf{NTime}(2^{n^d})$$ nichtdet. exponentielle Zeit

$$\mathsf{NL} = \mathsf{NLogSpace} = \mathsf{NSpace}(\log n)$$ nichtdet. logarithmischer Speicher

$$\mathsf{NPSpace} = \bigcup_{d \geq 1} \mathsf{NSpace}(n^d)$$ nichtdet. polynomieller Speicher

Beispiel: Die Existenz eines Hamiltonpfads ist in NP entscheidbar. Wenn ein Hamiltonpfad existiert, dann kann er in polynomieller Zeit erraten und überprüft werden.

Einfache Beobachtungen

Die folgenden Konstruktionen funktionieren wie im deterministischen Fall:

- Zeitreduktion durch linear Speedup
- Lineare Speicherreduktion
- Bandreduktion von Mehrband-TMs

Einfache Beobachtungen

Die folgenden Konstruktionen funktionieren wie im deterministischen Fall:

- Zeitreduktion durch linear Speedup
- Lineare Speicherreduktion
- Bandreduktion von Mehrband-TMs

Die Beziehungen von Zeit und Speicher bleiben ebenfalls erhalten, mit einer Besonderheit:

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt:

$$\mathsf{NTIME}(f) \subseteq \mathsf{NSPACE}(f)$$
$$\mathsf{NSPACE}(f) \subseteq \mathsf{DTIME}(2^{O(f)})$$

Einfache Beobachtungen

Die folgenden Konstruktionen funktionieren wie im deterministischen Fall:

- Zeitreduktion durch linear Speedup
- Lineare Speicherreduktion
- Bandreduktion von Mehrband-TMs

Die Beziehungen von Zeit und Speicher bleiben ebenfalls erhalten, mit einer Besonderheit:

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt:

$$\text{NTIME}(f) \subseteq \text{NSPACE}(f)$$
$$\text{NSPACE}(f) \subseteq \text{DTIME}(2^{O(f)})$$

Beweis: Beide Fälle wie im deterministischen Fall. Der Konfigurationsgraph ist auch hier exponentiell groß, aber kann wie zuvor deterministisch durchsucht werden. □

Deterministisch vs. nichtdeterministisch

Wir haben also nebenbei auch gezeigt: NTIME(f) $\subseteq$ NSPACE(f) $\subseteq$ DTIME($2^{O(f)}$).

Satz: Für jede beliebige Funktion $f : \mathbb{N} \to \mathbb{R}$ gilt:

$$\text{NTIME}(f) \subseteq \text{DTIME}(2^{O(f)})$$

Anmerkung: In Formale Systeme, Vorlesung 19, haben wir dieses Ergebnis durch eine alternative Konstruktion gezeigt (ausgehend von der Simulation einer beliebigen, unbeschränkten NTM durch deterministische Suche im Baum der möglichen Berechnungspfade).

Anmerkung 2: Es ist bis heute nicht bekannt, ob NTIME(f) $\subseteq$ DTIME(g) auch für eine Funktion $g \in o(2^{O(f)})$ gilt (Achtung: Klein-o-Notation!).

Was wir wissen

Aus unseren Beobachtungen folgen verschiedene einfache Beziehungen:

- „DTM $\subseteq$ NTM“: $\mathsf{L} \subseteq \mathsf{NL}$, $\mathsf{P} \subseteq \mathsf{NP}$, $\mathsf{PSpace} \subseteq \mathsf{NPSpace}$, $\mathsf{Exp} \subseteq \mathsf{NExp}$
- „Zeit $\subseteq$ Speicher“: $\mathsf{P} \subseteq \mathsf{PSpace}$, $\mathsf{NP} \subseteq \mathsf{NPSpace}$
- „(N)Speicher $\subseteq 2^{\text{(D)Zeit}}$“: $\mathsf{NL} \subseteq \mathsf{P}$, $\mathsf{NPSpace} \subseteq \mathsf{Exp}$

Was wir wissen

Aus unseren Beobachtungen folgen verschiedene einfache Beziehungen:

- „DTM $\subseteq$ NTM“: $\mathsf{L} \subseteq \mathsf{NL}$, $\mathsf{P} \subseteq \mathsf{NP}$, $\mathsf{PSpace} \subseteq \mathsf{NPSpace}$, $\mathsf{Exp} \subseteq \mathsf{NExp}$
- „Zeit $\subseteq$ Speicher“: $\mathsf{P} \subseteq \mathsf{PSpace}$, $\mathsf{NP} \subseteq \mathsf{NPSpace}$
- „(N)Speicher $\subseteq 2^{\text{(D)Zeit}}$“: $\mathsf{NL} \subseteq \mathsf{P}$, $\mathsf{NPSpace} \subseteq \mathsf{Exp}$

Zudem besagt der berühmte Satz von Savitch, dass speicherbeschränkte NTMs durch DTMs mit nur quadratischen Mehrkosten simuliert werden können. Daraus folgt:

Satz (Savitch): $\mathsf{PSpace} = \mathsf{NPSpace}$.

(ohne Beweis)

Was wir wissen

Aus unseren Beobachtungen folgen verschiedene einfache Beziehungen:

- „DTM $\subseteq$ NTM“: $\mathsf{L} \subseteq \mathsf{NL}$, $\mathsf{P} \subseteq \mathsf{NP}$, $\mathsf{PSpace} \subseteq \mathsf{NPSpace}$, $\mathsf{Exp} \subseteq \mathsf{NExp}$
- „Zeit $\subseteq$ Speicher“: $\mathsf{P} \subseteq \mathsf{PSpace}$, $\mathsf{NP} \subseteq \mathsf{NPSpace}$
- „(N)Speicher $\subseteq 2^{\text{(D)Zeit}}$“: $\mathsf{NL} \subseteq \mathsf{P}$, $\mathsf{NPSpace} \subseteq \mathsf{Exp}$

Zudem besagt der berühmte Satz von Savitch, dass speicherbeschränkte NTMs durch DTMs mit nur quadratischen Mehrkosten simuliert werden können. Daraus folgt:

Satz (Savitch): $\mathsf{PSpace} = \mathsf{NPSpace}$.

(ohne Beweis)

Zusammenfassung der wichtigsten bekannten Beziehungen:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSpace} = \mathsf{NPSpace} \subseteq \mathsf{Exp} \subseteq \mathsf{NExp}$$

Was wir nicht wissen

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser $\subseteq$ sogar $\subsetneq$ ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

Was wir nicht wissen

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser $\subseteq$ sogar $\subsetneq$ ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

Es wird aber vermutet, dass alle $\subseteq$ eigentlich $\subsetneq$ sind. Bekannt ist das aber nur bei exponentiell großen Ressourcenunterschieden:

Satz: Die folgenden Inklusionen sind echt:

- $NL \subsetneq PSpace$
- $P \subsetneq Exp$
- $NP \subsetneq NExp$

(ohne Beweis; folgt aus dem sogenanntem Time (bzw. Space) Hierarchy Theorem)

Effizient lösbare Probleme

Was bedeutet „effizient“?

Intuitiv klar: Lineare Algorithmen sind „effizient“

Was bedeutet „effizient“?

Intuitiv klar: Lineare Algorithmen sind „effizient“

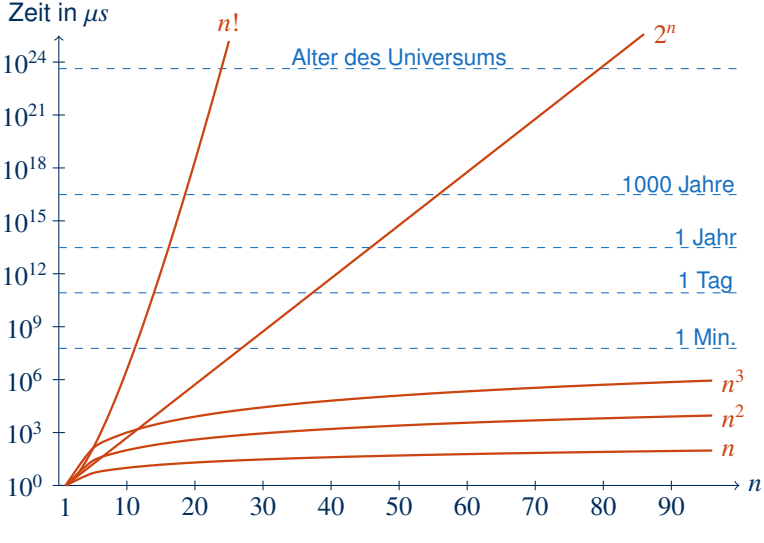
Aber der Begriff „linear“ ist nicht robust:

- Abhänging von Details des Maschinenmodells
- Abhänging von Details der Kodierung

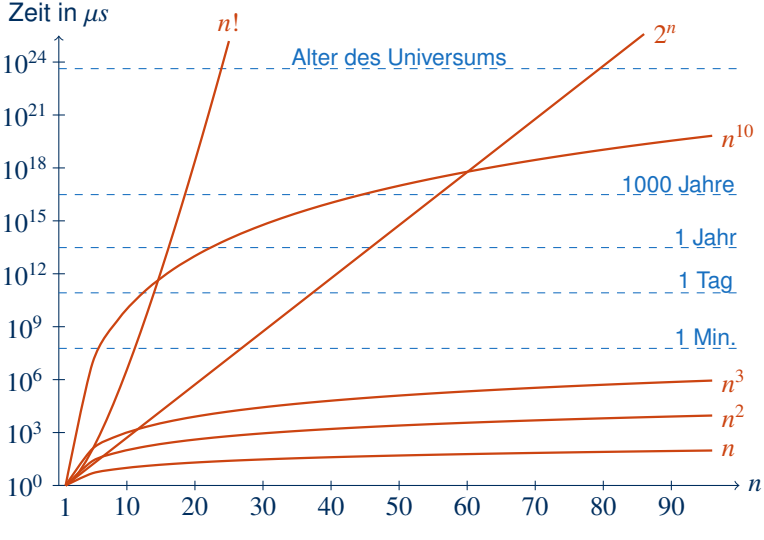
$\rightsquigarrow$ Polynomielle Zeit als robuste Verallgemeinerung von Linearzeit:

$$\mathsf{P} = \mathsf{PTIME} = \bigcup_{d \geq 1} \mathsf{DTIME}(n^d)$$

Wachstum einiger Funktionen



Wachstum einiger Funktionen



Polynomiell = effizient?

Wir verwenden P als mathematisches Modell für die Klasse der praktisch lösbaren Probleme.

Aber: diese Übereinstimmung ist nicht perfekt

- Polynome hohen Grades können sehr schnell wachsen
- Die konstanten Faktoren können auch sehr groß sein (und Linear Speedup hilft in der Praxis wenig)

Polynomiell = effizient?

Wir verwenden P als mathematisches Modell für die Klasse der praktisch lösbaren Probleme.

Aber: diese Übereinstimmung ist nicht perfekt

- Polynome hohen Grades können sehr schnell wachsen
- Die konstanten Faktoren können auch sehr groß sein (und Linear Speedup hilft in der Praxis wenig)

Dennoch: P ist von praktischem wie theoretischem Interesse

- Praxis: Die meisten polynomiellen Probleme erlauben Algorithmen in $O(x^2)$ oder $O(x^3)$, während man zum Beispiel $O(x^{10})$ selten antrifft
- Theorie: Unabhängig von der tatsächlichen Laufzeit liefert uns P tiefe Einsichten in die Struktur eines Problems

Probleme in P

Aus der 4. Vorlesung kennen wir bereits ein typisches P-Problem.

Rückblick:

- Eine Hornklausel ist eine aussagenlogische Formel der Form $p_1 \wedge \ldots \wedge p_n \rightarrow q$ mit $n \geq 0$ (bei $n = 0$ ergibt sich einfach q – ein Fakt)
- Eine Formel ist erfüllbar wenn sie für eine Wertezuweisung auf wahr abgebildet wird
- Eine Menge von Formeln ist erfüllbar, wenn es eine Wertzuweisung gibt, die alle ihre Elemente gleichzeitig wahr macht

Erfüllbarkeit propositionaler Horn-Formeln (**HornSAT**) ist das folgende Entscheidungsproblem:

Gegeben: Eine Menge aussagenlogischer Formeln Hornklauseln
Frage: Ist diese Menge erfüllbar?

2SAT

Ein weiteres Beispiel für ein polynomiell lösbares Problem aus der Aussagenlogik:

Rückblick:

- Ein Literal ist ein aussagenlogisches Atom oder ein negiertes aussagenlogisches Atom
- Eine Klausel ist eine Disjunktion von Literalen, die man oft einfach als Menge darstellt
- Eine Formel in Klauselform ist eine Konjunktion von Klauseln, ebenfalls dargestellt als Menge

2SAT

Ein weiteres Beispiel für ein polynomiell lösbares Problem aus der Aussagenlogik:

Rückblick:

- Ein Literal ist ein aussagenlogisches Atom oder ein negiertes aussagenlogisches Atom
- Eine Klausel ist eine Disjunktion von Literalen, die man oft einfach als Menge darstellt
- Eine Formel in Klauselform ist eine Konjunktion von Klauseln, ebenfalls dargestellt als Menge

2SAT ist das folgende Entscheidungsproblem:

Gegeben: Eine aussagenlogische Formel F in Klauselform, bei der jede Klausel höchstens zwei Literale enthält
Frage: Ist F erfüllbar?

2SAT ist in P

Satz: 2SAT $\in$ P

2SAT ist in P

Satz: 2SAT $\in$ P

Beweis: Dazu wollen wir einen polynomiellen Algorithmus angeben.

2SAT ist in P

Satz: 2SAT $\in$ P

Beweis: Dazu wollen wir einen polynomiellen Algorithmus angeben.

Der Resolutionsalgorithmus aus Vorlesung 4 erfüllt den Zweck ohne jegliche Abwandlungen:

- Ein Resolutionsschritt kombiniert zwei Klauseln $\{p, L_1\}$ und $\{\neg p, L_2\}$ zu einer neuen Klausel $\{L_1, L_2\}$
- Aus Zweier-Klauseln entstehen also immer wieder Klauseln mit höchstens zwei Literalen
- Es gibt nur quadratisch viele Klauseln mit höchstens zwei Literalen
- Der Algorithmus terminiert also nach polynomieller Zeit □

Effizienter als P?

Wenn wir eine robuste Klasse wollen, die Linearzeit-Algorithmen enthält, dann enthält sie auch beliebige polynomiellen Algorithmen.

Gibt es Probleme, die noch einfacher sind?

Effizienter als P?

Wenn wir eine robuste Klasse wollen, die Linearzeit-Algorithmen enthält, dann enthält sie auch beliebige polynomiellen Algorithmen.

Gibt es Probleme, die noch einfacher sind?

- **Sub-lineare Zeit** funktioniert mit dem normalen TM-Modell nicht, da man in dieser Zeit nicht einmal die Eingabe lesen kann (erfordert Rechenmodelle mit einer Form von Parallelverarbeitung ...)
- **Sub-linearer Speicher** ist machbar, wenn man ein getrenntes schreibgeschütztes Eingabeband erlaubt (siehe letzte Vorlesung)

$\rightsquigarrow$ Komplexitätsklassen L und NL

Was kann L?

Intuition: ein Algorithmus mit logarithmischem Speicher kann:

- Eine feste Anzahl an binärkodierten Zählern speichern, die nicht größer als $O(n)$ werden
- Eine feste Anzahl an „Pointern“ auf eine Position der Eingabe speichern
- Den Inhalt von Zählern und Speicherstellen miteinander vergleichen

Damit kann man bereits viele einfache Algorithmen umsetzen

Was kann L?

Intuition: ein Algorithmus mit logarithmischem Speicher kann:

- Eine feste Anzahl an binärkodierten Zählern speichern, die nicht größer als $O(n)$ werden
- Eine feste Anzahl an „Pointern“ auf eine Position der Eingabe speichern
- Den Inhalt von Zählern und Speicherstellen miteinander vergleichen

Damit kann man bereits viele einfache Algorithmen umsetzen

Beispiel: Die Sprache aller Wörter über $\{0, 1\}$, welche die gleiche Anzahl der Symbole 0 und 1 enthalten, ist in L:

- Wir verwenden zwei Zähler für die beiden Zahlen
- Die TM liest das Wort von links nach rechts und erhöht jeweils den entsprechenden Zähler
- Am Ende wird der Wert beider Zähler verglichen

Noch ein Beispiel in L

Beispiel: Die Sprache **Palindrom**, welche Wörter enthält, die von hinten gelesen genauso lauten wie von vorn, ist in L:

- Wir verwenden zwei Pointer in die Eingabe, einer auf die erste und einer auf die letzte Eingabezelle
- Die TM vergleicht die Speicherinhalte bei den Pointern und verschiebt sie anschließend um eine Zelle in Richtung Wortmitte
- Das Wort wird akzeptiert, wenn die Pointer sich treffen und alle Vergleiche erfolgreich waren

Reduktionen

Effizient berechenbare Funktionen

Bisher haben wir Entscheidungsprobleme betrachtet. Man kann unsere Definitionen leicht auf andere Berechnungsprobleme übertragen.

Eine totale Funktion f ist in polynomieller Zeit berechenbar, wenn es eine polynomiell zeitbeschränkte DTM $\mathcal{M}$ gibt, die f berechnet, d.h. mit $f = f_{\mathcal{M}}$.

Effizient berechenbare Funktionen

Bisher haben wir Entscheidungsprobleme betrachtet. Man kann unsere Definitionen leicht auf andere Berechnungsprobleme übertragen.

Eine totale Funktion f ist in polynomieller Zeit berechenbar, wenn es eine polynomiell zeitbeschränkte DTM $\mathcal{M}$ gibt, die f berechnet, d.h. mit $f = f_{\mathcal{M}}$.

Bei logarithmischen Speicherschranken müssen wir vorsichtig sein: Das Ergebnis könnte größer sein als der Arbeitsspeicher!

Eine totale Funktion f ist in logarithmischem Speicher berechenbar, wenn es eine 3-Band DTM $\mathcal{M}$ gibt, die f wie folgt berechnet: (1) die Eingabe befindet sich auf dem schreibgeschützten Eingabeband, (2) die DTM verwendet maximal $O(\log n)$ Speicherzellen auf dem Arbeitsband, (3) die Ausgabe wird auf das Ausgabeband geschrieben (pro Zelle einmaliges Schreiben, kein Lesen).

Polynomielle Many-One-Reduktionen

Mithilfe effizient berechenbarer Funktionen können wir ein Problem effizient auf ein anderes reduzieren.

Eine polynomiell berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ ist eine polynomielle Many-One-Reduktion von einer Sprache **P** auf eine Sprache **Q** (in Symbolen: $\mathbf{P} \leq_p \mathbf{Q}$), wenn für alle Wörter $w \in \Sigma^*$ gilt:

$$w \in \mathbf{P} \quad \text{genau dann wenn} \quad f(w) \in \mathbf{Q}$$

Polynomielle Many-One-Reduktionen

Mithilfe effizient berechenbarer Funktionen können wir ein Problem effizient auf ein anderes reduzieren.

Eine polynomiell berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ ist eine polynomielle Many-One-Reduktion von einer Sprache **P** auf eine Sprache **Q** (in Symbolen: $\mathbf{P} \leq_p \mathbf{Q}$), wenn für alle Wörter $w \in \Sigma^*$ gilt:

$$w \in \mathbf{P} \quad \text{genau dann wenn} \quad f(w) \in \mathbf{Q}$$

Wir sprechen oft einfach von polynomiellen Reduktionen.

Logarithmische Many-One-Reduktionen könnten analog definiert werden (wir werden uns damit nicht näher beschäftigen).

Komplexität durch Reduktion zeigen

Idee: Wenn man ein Problem **A** leicht auf ein leichtes Problem **B** reduzieren kann, dann ist **A** ebenfalls leicht.

Satz: Falls $\mathbf{A} \leq_p \mathbf{B}$ und $\mathbf{B} \in \text{PTime}$, dann ist $\mathbf{A} \in \text{PTime}$.

Beweis: Folgt durch Hintereinanderausführung der polynomiellen Prozeduren, da die Summe und Komposition von Polynomen ein Polynom ist. □

Komplexität durch Reduktion zeigen

Idee: Wenn man ein Problem **A** leicht auf ein leichtes Problem **B** reduzieren kann, dann ist **A** ebenfalls leicht.

Satz: Falls $\mathbf{A} \leq_p \mathbf{B}$ und $\mathbf{B} \in \text{PTIME}$, dann ist $\mathbf{A} \in \text{PTIME}$.

Beweis: Folgt durch Hintereinanderausführung der polynomiellen Prozeduren, da die Summe und Komposition von Polynomen ein Polynom ist. □

Die Umkehrung wird uns später noch interessieren:

Satz: Falls $\mathbf{A} \leq_p \mathbf{B}$ und $\mathbf{A} \notin \text{PTIME}$, dann ist $\mathbf{B} \notin \text{PTIME}$.

Dazu mehr in den kommenden Vorlesungen . . .

Beispiel

2-Färbbarkeit (**2Col**) ist das folgende Problem:
Gegeben: Ein ungerichteter Graph G
Frage: Kann man die Knoten von G mit zwei Farben (rot und blau) so einfärben, dass keine gleichfarbigen Knoten durch Kanten verbunden sind?

Man kann **2Col** $\in$ P leicht durch Reduktion auf **2SAT** zeigen:

- Für jeden Knoten v führen wir ein Atom p_v ein
- Für jede Kante $v \to w$ führen wir zwei Klauseln ein: $\{p_v, p_w\}$ und $\{\neg p_v, \neg p_w\}$

Beispiel

2-Färbbarkeit (**2Col**) ist das folgende Problem:
Gegeben: Ein ungerichteter Graph G
Frage: Kann man die Knoten von G mit zwei Farben (rot und blau) so einfärben, dass keine gleichfarbigen Knoten durch Kanten verbunden sind?

Man kann **2Col** $\in$ P leicht durch Reduktion auf **2SAT** zeigen:

- Für jeden Knoten v führen wir ein Atom p_v ein
- Für jede Kante $v \to w$ führen wir zwei Klauseln ein: $\{p_v, p_w\}$ und $\{\neg p_v, \neg p_w\}$

Dies kann man offenbar in polynomieller Zeit berechnen.

Beispiel

2-Färbbarkeit (**2Col**) ist das folgende Problem:
Gegeben: Ein ungerichteter Graph G
Frage: Kann man die Knoten von G mit zwei Farben (rot und blau) so einfärben, dass keine gleichfarbigen Knoten durch Kanten verbunden sind?

Man kann **2Col** $\in$ P leicht durch Reduktion auf **2SAT** zeigen:

- Für jeden Knoten v führen wir ein Atom p_v ein
- Für jede Kante $v \to w$ führen wir zwei Klauseln ein: $\{p_v, p_w\}$ und $\{\neg p_v, \neg p_w\}$

Dies kann man offenbar in polynomieller Zeit berechnen.

Die Reduktionseigenschaft gilt:

- Wenn der Graph 2-färbbar ist, dann verwenden wir die Farben als Wahrheitswerte und erhalten eine erfüllende Zuweisung
- Wenn die Formel erfüllbar ist, dann erhalten wir umgekehrt aus der Wertzuweisung eine korrekte 2-Färbung. □

Zusammenfassung und Ausblick

Die grundlegenden Beziehungen der Komplexitätsklassen sind

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

Die Klassen P, L und NL sind mathematische Modelle für effiziente Algorithmen

Mit polynomiellen Reduktionen kann man aus der Komplexität eines Problems auf die eines anderen schließen

Was erwartet uns als nächstes?

- NP
- NL