TECHNISCHE UNIVERSITÄT DRESDEN

THEORETISCHE INFORMATIK UND LOGIK

7. Vorlesung: Einführung in die Komplexitätstheorie

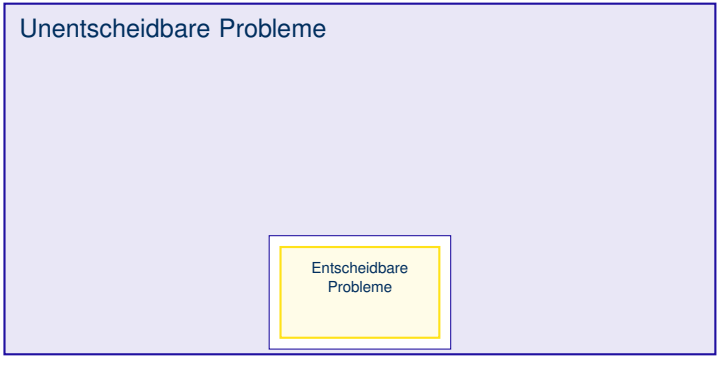
Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 6. Mai 2021

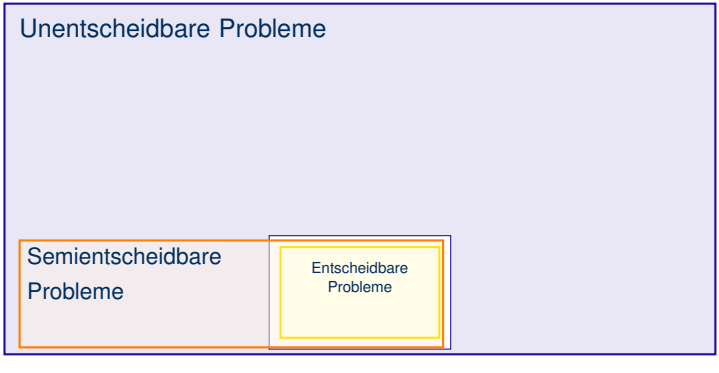
Übersicht

Der Raum der formalen Sprachen (Wortprobleme) lässt sich wie folgt aufteilen:



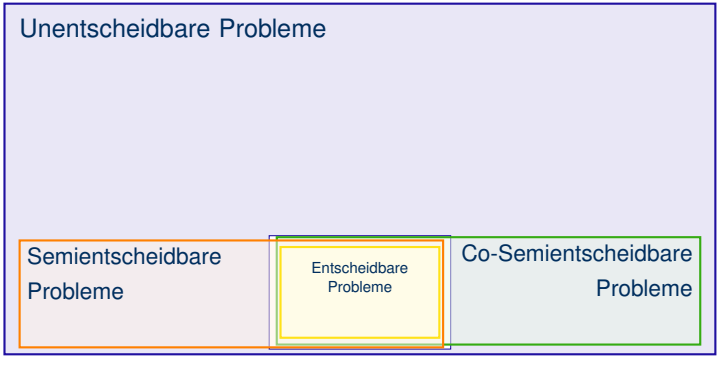
Übersicht

Der Raum der formalen Sprachen (Wortprobleme) lässt sich wie folgt aufteilen:



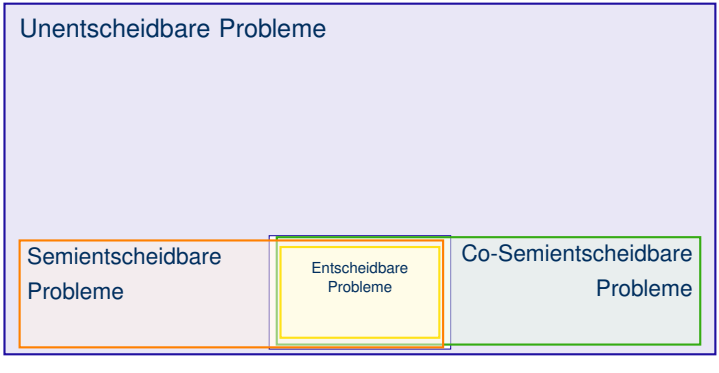
Übersicht

Der Raum der formalen Sprachen (Wortprobleme) lässt sich wie folgt aufteilen:



Übersicht

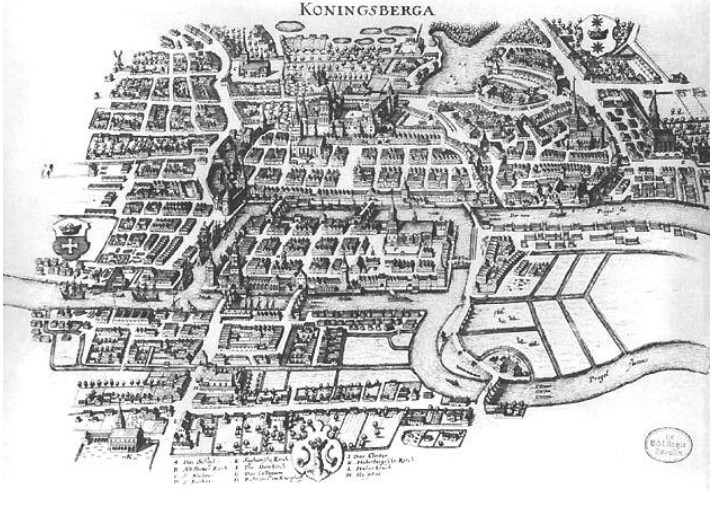
Der Raum der formalen Sprachen (Wortprobleme) lässt sich wie folgt aufteilen:



$\rightsquigarrow$ Wie kann man die entscheidbaren Probleme weiter unterteilen?

Königsberg im 18. Jahrhundert

Königsberg, Preussen (heute Kaliningrad, Russland):

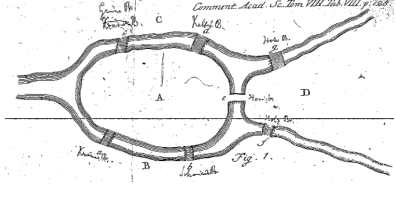


Ein klassisches Problem

Ein populäre Frage der Königsberger:

> Gibt es einen Weg durch die Stadt, auf dem man jede der sieben Brücken von Königsberg genau einmal überquert?

Im Jahr 1735 beschäftigt sich Leonhard Euler (Mathematiker in Sankt Petersburg) mit der Frage ...

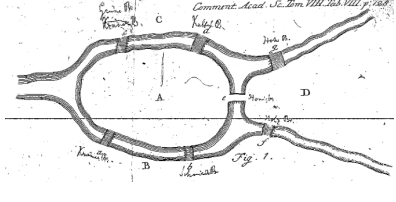


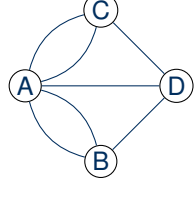
Ein klassisches Problem

Ein populäre Frage der Königsberger:

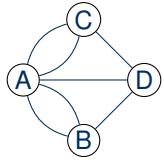
> Gibt es einen Weg durch die Stadt, auf dem man jede der sieben Brücken von Königsberg genau einmal überquert?

Im Jahr 1735 beschäftigt sich Leonhard Euler (Mathematiker in Sankt Petersburg) mit der Frage … und abstrahiert …



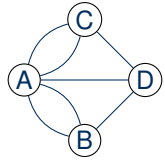


Eulers Einsichten



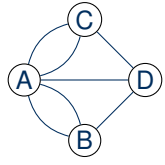
- Lage der Brücken und Wege von einer Brücke zur nächsten sind egal

Eulers Einsichten



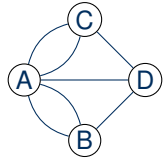
- Lage der Brücken und Wege von einer Brücke zur nächsten sind egal
- Ein Pfad kann als Liste von Brücken dargestellt werden, aber es gibt viele denkbare Listen ($7! = 5040$)

Eulers Einsichten



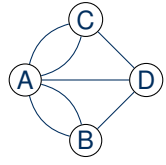
- Lage der Brücken und Wege von einer Brücke zur nächsten sind egal
- Ein Pfad kann als Liste von Brücken dargestellt werden, aber es gibt viele denkbare Listen ($7! = 5040$)
- Wenn man n-mal auf einer Landmasse ankommt, dann muss man sie auch n-mal verlassen – außer sie ist Start oder Ziel

Eulers Einsichten



- Lage der Brücken und Wege von einer Brücke zur nächsten sind egal
- Ein Pfad kann als Liste von Brücken dargestellt werden, aber es gibt viele denkbare Listen ($7! = 5040$)
- Wenn man n-mal auf einer Landmasse ankommt, dann muss man sie auch n-mal verlassen – außer sie ist Start oder Ziel
- Daher muss jede Landmasse – außer der Start und das Ziel – eine gerade Zahl an Brücken besitzen

Eulers Einsichten



- Lage der Brücken und Wege von einer Brücke zur nächsten sind egal
- Ein Pfad kann als Liste von Brücken dargestellt werden, aber es gibt viele denkbare Listen ($7! = 5040$)
- Wenn man n-mal auf einer Landmasse ankommt, dann muss man sie auch n-mal verlassen – außer sie ist Start oder Ziel
- Daher muss jede Landmasse – außer der Start und das Ziel – eine gerade Zahl an Brücken besitzen

$\rightsquigarrow$ Das Rätsel der Königsberger ist unlösbar

Verallgemeinerung

Euler legt damit den Grundstein für die Graphentheorie, und definiert ein heute nach ihm benanntes Konzept:

Ein Eulerpfad ist ein Pfad in einem Graphen, der jede Kante genau einmal durchquert.
Ein Eulerkreis ist ein zyklischer Eulerpfad.

Verallgemeinerung

Euler legt damit den Grundstein für die Graphentheorie, und definiert ein heute nach ihm benanntes Konzept:

Ein Eulerpfad ist ein Pfad in einem Graphen, der jede Kante genau einmal durchquert.
Ein Eulerkreis ist ein zyklischer Eulerpfad.

Euler zeigte also:

Satz (Euler): Ein Graph hat genau dann einen Eulerschen Pfad, wenn er maximal zwei Knoten ungeraden Grades besitzt.

Verallgemeinerung

Euler legt damit den Grundstein für die Graphentheorie, und definiert ein heute nach ihm benanntes Konzept:

Ein Eulerpfad ist ein Pfad in einem Graphen, der jede Kante genau einmal durchquert.
Ein Eulerkreis ist ein zyklischer Eulerpfad.

Euler zeigte also:

Satz (Euler): Ein Graph hat genau dann einen Eulerschen Pfad, wenn er maximal zwei Knoten ungeraden Grades besitzt.

Ein ähnliches Problem

1859 publiziert der Physiker und Astronom Sir William Rowan Hamilton ein Brettspiel.
Es verkauft sich nicht gut, aber es liefert uns ein weiteres Rätsel auf Graphen:

Ein Hamiltonpfad ist ein Pfad in einem Graphen, der jeden Knoten genau einmal durchquert. Ein Hamiltonkreis ist ein zyklischer Hamiltonpfad.

Ein ähnliches Problem

1859 publiziert der Physiker und Astronom Sir William Rowan Hamilton ein Brettspiel. Es verkauft sich nicht gut, aber es liefert uns ein weiteres Rätsel auf Graphen:

Ein Hamiltonpfad ist ein Pfad in einem Graphen, der jeden Knoten genau einmal durchquert. Ein Hamiltonkreis ist ein zyklischer Hamiltonpfad.

Wie bei Eulerpfaden ist die naive Lösung sehr ineffizient, da man alle (exponentiell viele) Pfade systematisch Durchprobieren muss.

Ein ähnliches Problem

1859 publiziert der Physiker und Astronom Sir William Rowan Hamilton ein Brettspiel. Es verkauft sich nicht gut, aber es liefert uns ein weiteres Rätsel auf Graphen:

Ein Hamiltonpfad ist ein Pfad in einem Graphen, der jeden Knoten genau einmal durchquert. Ein Hamiltonkreis ist ein zyklischer Hamiltonpfad.

Wie bei Eulerpfaden ist die naive Lösung sehr ineffizient, da man alle (exponentiell viele) Pfade systematisch Durchprobieren muss.

Aber im Gegensatz zu Eulerpfaden hat bislang niemand eine elegante einfache Lösung gefunden. Die meisten Experten glauben, dass es prinzipiell keine effiziente Lösung geben kann.

Ein ähnliches Problem

1859 publiziert der Physiker und Astronom Sir William Rowan Hamilton ein Brettspiel. Es verkauft sich nicht gut, aber es liefert uns ein weiteres Rätsel auf Graphen:

Ein Hamiltonpfad ist ein Pfad in einem Graphen, der jeden Knoten genau einmal durchquert. Ein Hamiltonkreis ist ein zyklischer Hamiltonpfad.

Wie bei Eulerpfaden ist die naive Lösung sehr ineffizient, da man alle (exponentiell viele) Pfade systematisch Durchprobieren muss.

Aber im Gegensatz zu Eulerpfaden hat bislang niemand eine elegante einfache Lösung gefunden. Die meisten Experten glauben, dass es prinzipiell keine effiziente Lösung geben kann.

Kann man beweisen, dass es keine bessere Lösung gibt?

Leicht oder schwer?

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen kürzesten Weg von A nach B.

Leicht oder schwer?

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen kürzesten Weg von A nach B.

Leicht! Lösbar in polynomieller Zeit, z.B. mit Dijkstras Algorithmus

Leicht oder schwer?

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen kürzesten Weg von A nach B.

Leicht! Lösbar in polynomieller Zeit, z.B. mit Dijkstras Algorithmus

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen längsten Weg von A nach B.

Leicht oder schwer?

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen kürzesten Weg von A nach B.

Leicht! Lösbar in polynomieller Zeit, z.B. mit Dijkstras Algorithmus

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen längsten Weg von A nach B.

Schwer! Keine sub-exponentielle Lösung bekannt

Leicht oder schwer?

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen kürzesten Weg von A nach B.

Leicht! Lösbar in polynomieller Zeit, z.B. mit Dijkstras Algorithmus

Aufgabe: Gegeben einen Graphen mit zwei Knoten A und B, finde einen längsten Weg von A nach B.

Schwer! Keine sub-exponentielle Lösung bekannt

Warum sind manche Probleme leicht und andere schwer?

(obwohl sie sich auf den ersten Blick stark ähneln)

Einleitung

Fragen:

- Warum sind manche Probleme leicht und andere schwer?
- Und sind sie wirklich schwer oder hatten wir nur bisher nicht die richtige Idee zu ihrer Lösung?

Der Weg zu Antworten:

Ein Ziel der Komplexitätstheorie ist die Unterteilung berechenbarer Probleme entsprechend der Menge an Ressourcen, die zu ihrer Lösung nötig sind

- Unterteile Problem in Klassen gleicher „Schwere“
- Entwickle Methoden zur Bestimmung der Komplexität eines Problems

Einleitung

Fragen:

- Warum sind manche Probleme leicht und andere schwer?
- Und sind sie wirklich schwer oder hatten wir nur bisher nicht die richtige Idee zu ihrer Lösung?

Der Weg zu Antworten:

Ein Ziel der Komplexitätstheorie ist die Unterteilung berechenbarer Probleme entsprechend der Menge an Ressourcen, die zu ihrer Lösung nötig sind

- Unterteile Problem in Klassen gleicher „Schwere“
- Entwickle Methoden zur Bestimmung der Komplexität eines Problems

Beschränkung von Zeit und Raum

Turingmaschinen beschränken

Wir wiederholen zunächst einige Grundlagen aus der Vorlesung Formale Systeme ...

TMs verwenden zwei Ressourcen, die man beschränken kann:

- Speicher: die Zahl der verwendeten Speicherzellen
- Zeit: die Zahl der durchgeführten Berechnungsschritte

Turingmaschinen beschränken

Wir wiederholen zunächst einige Grundlagen aus der Vorlesung Formale Systeme ...

TMs verwenden zwei Ressourcen, die man beschränken kann:

- Speicher: die Zahl der verwendeten Speicherzellen
- Zeit: die Zahl der durchgeführten Berechnungsschritte

Feste Schranken ergeben wenig Sinn (endliche Automaten)

$\rightsquigarrow$ Schranken werden als Funktion in der Länge der Eingabe angegeben

Beispiel: LBAs beschränken den verfügbaren Speicher auf die Anzahl der Symbole in der Eingabe. Dies entspricht einer Funktion, welche die Länge n der Eingabe auf den Maximalwert von n Speicherzellen abbildet.

Zur Erinnerung: O-Notation

Die O-Notation (mit großem O!) charakterisiert Funktionen nach ihrem asymptotischen Verhalten und versteckt lineare Faktoren.

Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ schreiben wir $f \in O(g)$ wenn gilt:

Es gibt eine Zahl $c > 0$ und eine Zahl $n_0 \in \mathbb{N}$,
so dass für jedes $n > n_0$ gilt: $f(n) \leq c \cdot g(n)$

Das bedeutet, f wächst nicht wesentlich schneller als g.

Zur Erinnerung: O-Notation

Die O-Notation (mit großem O!) charakterisiert Funktionen nach ihrem asymptotischen Verhalten und versteckt lineare Faktoren.

Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ schreiben wir $f \in O(g)$ wenn gilt:

Es gibt eine Zahl $c > 0$ und eine Zahl $n_0 \in \mathbb{N}$,
so dass für jedes $n > n_0$ gilt: $f(n) \leq c \cdot g(n)$

Das bedeutet, f wächst nicht wesentlich schneller als g.

Notation 1: Manchmal schreibt man statt $f \in O(g)$ auch $f = O(g)$ (allerdings ist = dann eine asymmetrische Relation).

Notation 2: Manchmal schreibt man statt $f \in O(g)$ (oder $f = O(g)$) auch $f(n) \in O(g(n))$ (oder $f(n) = O(g(n))$).

Beispiele:

- $(10n^3 + 42n^2 - n + 100) \in O(n^3)$
- $(2^n + n^{2000}) \in O(2^n)$
- $2^{729} \in O(1)$

Schwestern der O-Notation

Randbemerkung: Es gibt neben der O-Notation noch eine Reihe weiterer asymptotischer Notationen, die in der Informatik verwendet werden:

Notation	$C = \lim_{n\to\infty} \frac{f(n)}{g(n)}$	Intuition
$f \in O(g)$	$C < \infty$	„$f \leq g$“
$f \in \Omega(g)$	$C > 0$	„$f \geq g$“
$f \in \Theta(g)$	$0 < C < \infty$	„$f = g$“
$f \in o(g)$	$C = 0$	„$f < g$“
$f \in \omega(g)$	$C = \infty$	„$f > g$“

Schranken für Zeit und Raum

Die O-Notation wird verwendet, um allgemeine Ressourcenschranken für TMs anzugeben.

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

- $\mathcal{M}$ heißt $O(f)$-zeitbeschränkt wenn es eine Funktion $g \in O(f)$ gibt, so dass $\mathcal{M}$ für eine beliebige Eingabe $w \in \Sigma^*$ 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 maximal $g(|w|)$ Speicherzellen verwendet.

Schranken für Zeit und Raum

Die O-Notation wird verwendet, um allgemeine Ressourcenschranken für TMs anzugeben.

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

- $\mathcal{M}$ heißt $O(f)$-zeitbeschränkt wenn es eine Funktion $g \in O(f)$ gibt, so dass $\mathcal{M}$ für eine beliebige Eingabe $w \in \Sigma^*$ 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 maximal $g(|w|)$ Speicherzellen verwendet.

Beispiel: Ein LBA entspricht einer $O(n)$-speicherbeschränkten TM.

Beispiel: Eine naive Suche nach einem Eulerpfad wäre $O(n!)$-zeitbeschränkt, wenn die Zahl der Kanten n nicht übersteigt.

Lineare Faktoren

Die O-Notation versteckt bei der Abschätzung der Laufzeit beliebig große konstante Faktoren. Werden dadurch nicht zu viele unterschiedlich schwere Probleme in einen Topf geworfen?

Lineare Faktoren

Die O-Notation versteckt bei der Abschätzung der Laufzeit beliebig große konstante Faktoren. Werden dadurch nicht zu viele unterschiedlich schwere Probleme in einen Topf geworfen?

Nein. Im Gegenteil: das TM-Modell der Berechnung kann konstante Faktoren nicht unterscheiden, zumindest wenn man mehrere Bänder erlaubt:

Satz (Linear Speedup Theorem): Sei $\mathcal{M}$ eine TM mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band TM $\mathcal{M}'$, die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.

Lineare Faktoren

Die O-Notation versteckt bei der Abschätzung der Laufzeit beliebig große konstante Faktoren. Werden dadurch nicht zu viele unterschiedlich schwere Probleme in einen Topf geworfen?

Nein. Im Gegenteil: das TM-Modell der Berechnung kann konstante Faktoren nicht unterscheiden, zumindest wenn man mehrere Bänder erlaubt:

Satz (Linear Speedup Theorem): Sei $\mathcal{M}$ eine TM mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band TM $\mathcal{M}'$, die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.

Beispiel: Wenn ein Problem mit einer Zwei-Band-TM in n^3 Schritten gelöst werden kann, so ist das auch in $\frac{n^3}{1000000000} + n + 2$ Schritten möglich.

Linear Speedup Theorem: Beweis (1)

Satz (Linear Speedup Theorem): Sei $\mathcal{M}$ eine TM mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band TM $\mathcal{M}'$, die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.

Linear Speedup Theorem: Beweis (1)

Satz (Linear Speedup Theorem): Sei $\mathcal{M}$ eine TM mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band TM $\mathcal{M}'$, die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.

Beweisskizze: Wenn $\mathcal{M}$ das Arbeitsalphabet Γ hatte, dann verwenden wird für $\mathcal{M}'$ das Arbeitsalphabet $\Gamma' = \Sigma \cup \Gamma^{6c}$.

Wir können Bandinhalte dadurch effizient kodieren:

- $\mathcal{M}'$ liest die Eingabe und erzeugt eine kodierte Kopie auf Band 2
- Dabei werden jeweils $6c$ Zeichen aus Σ in eines aus Γ^{6c} übersetzt
- Diese Transkodierung benötigt $n + 2$ Schritte

Linear Speedup Theorem: Beweis (2)

Satz (Linear Speedup Theorem): Sei $\mathcal{M}$ eine TM mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band TM $\mathcal{M}'$, die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.

Beweisskizze: Wir haben die Eingabe im Alphabet Γ^{6c} kodiert.

Linear Speedup Theorem: Beweis (2)

Satz (Linear Speedup Theorem): Sei $\mathcal{M}$ eine TM mit $k > 1$ Bändern, die bei Eingaben der Länge n nach maximal $f(n)$ Schritten hält. Dann gibt es für jede natürliche Zahl $c > 0$ eine äquivalente k-Band TM $\mathcal{M}'$, die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält.

Beweisskizze: Wir haben die Eingabe im Alphabet Γ^{6c} kodiert.

Jetzt kann man $\mathcal{M}$ simulieren:

- Lies (in vier Schritten) das Γ^{6c}-Symbol an den aktuellen k Bandpositionen, sowie jeweils links und rechts davon.
- Das Ergebnis und die genaue Bandposition von $\mathcal{M}$ wird als Zustand gespeichert: wir verwenden dazu $|Q \times \{1, \ldots, k\} \times \Gamma^{18ck}|$ zusätzliche Zustände
- Simuliere (in zwei Schritten) die nächsten $6c$ Schritte von $\mathcal{M}$ ($\mathcal{M}'$ verändert maximal das aktuelle Bandfeld und einen Nachbarn)

Ergebnis: Simulation von $6c$ $\mathcal{M}$-Schritten und 6 $\mathcal{M}'$-Schritten. □

Linear Speedup: Diskussion

Kann jedes Programm beliebig schnell gemacht werden?

Linear Speedup: Diskussion

Kann jedes Programm beliebig schnell gemacht werden?

In der Praxis: Nein

- Wirkprinzip Linear Speedup: kodiere mehr Information pro Bandfeld und verarbeite diese auf einen Schlag mithilfe einer größeren Zustandsübergangstabelle
- In der Praxis kann man nicht beliebig große Daten in einem Schritt lesen
- In der Praxis kann man nicht beliebig komplexe Zustandsübergänge in konstanter Zeit realisieren

Linear Speedup: Diskussion

Kann jedes Programm beliebig schnell gemacht werden?

In der Praxis: Nein

- Wirkprinzip Linear Speedup: kodiere mehr Information pro Bandfeld und verarbeite diese auf einen Schlag mithilfe einer größeren Zustandsübergangstabelle
- In der Praxis kann man nicht beliebig große Daten in einem Schritt lesen
- In der Praxis kann man nicht beliebig komplexe Zustandsübergänge in konstanter Zeit realisieren

In der Theorie: Nein

- Wir interessieren uns für asymptotisches Verhalten bei beliebig wachsenden Eingaben
- Lineare Faktoren machen meist nur bei relativ kleinen Werten einen Unterschied

Wichtige Komplexitätsklassen

Zeit und Raum, deterministisch

Beschränkte TMs können verwendet werden, um viele weitere Sprachklassen zu definieren.

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

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

Zeit und Raum, deterministisch

Beschränkte TMs können verwendet werden, um viele weitere Sprachklassen zu definieren.

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

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

Beispiel: Die naive Suche nach Eulerpfaden kann in DSPACE(n) implementiert werden (Übung: wie?).

Zeit und Raum, deterministisch

Beschränkte TMs können verwendet werden, um viele weitere Sprachklassen zu definieren.

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

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

Beispiel: Die naive Suche nach Eulerpfaden kann in DSPACE(n) implementiert werden (Übung: wie?).

Beispiel: Das Halteproblem ist in keiner der Klassen DTIME($f(n)$) oder DSPACE($f(n)$), da es durch keine TM entschieden wird.

Maschinenmodelle

Es gibt viele unterschiedliche Versionen von deterministischen TMs und viele alternative Berechnungsmodelle (z.B. Mehrband-Maschinen und WHILE-Programme).

Sind DTIME(f) und DSPACE(f) für jedes TM-Modell gleich?

Maschinenmodelle

Es gibt viele unterschiedliche Versionen von deterministischen TMs und viele alternative Berechnungsmodelle (z.B. Mehrband-Maschinen und WHILE-Programme).

Sind DTIME(f) und DSPACE(f) für jedes TM-Modell gleich?

Antwort: „Nein, aber bei vielen typischen Variationen gibt es nur polynomielle Unterschiede.“

Beispiel: Jede $O(f(n))$-zeitbeschränkte k-Band-TM kann durch eine $O(k \cdot f^2(n))$-zeitbeschränkte 1-Band TM simuliert werden (siehe Formale Systeme, Vorlesung 18). Einfacher gesagt: Der Verzicht auf mehrere Bänder verursacht maximal quadratische Zeitkosten (k ist hier ein linearer Faktor).

Anmerkung: Wir betrachten hier verschiedene Versionen deterministischer Rechenmodelle. Zwischen DTMs und NTMs gibt es vermutlich schon große (nicht-polynomielle) Unterschiede.

Kodierungsdetails

Es gibt viele unterschiedliche Arten, wie Eingaben von Problemen als Wörter kodiert werden können.

Sind DTIME(f) und DSPACE(f) für jede Kodierung gleich?

Kodierungsdetails

Es gibt viele unterschiedliche Arten, wie Eingaben von Problemen als Wörter kodiert werden können.

Sind DTIME(f) und DSPACE(f) für jede Kodierung gleich?

Antwort: „Nein, aber vernünftige Kodierungen unterscheiden sich voneinander in der Regel nur polynomiell.“

Beispiel: Ein Graph kann als Adjazenzmatrix kodiert werden ($O(n^2)$ Speicher) oder z.B. auch als Adjazenliste ($O(e \cdot \log v)$ Speicher für e Kanten und v Knoten). Letzteres ist deutlich effizienter für lichte Graphen, aber der Unterschied bleibt stets polynomiell.

Aber: Wir werden Fälle sehen, in denen eine (besonders ineffiziente) Kodierung die Komplexität verändert.

Implementierungsdetails

Es gibt viele unterschiedliche Arten um ein Problem praktisch zu lösen, z.B. unter Verwendung spezifischer Datenstrukturen.

Sind DTIME(f) und DSPACE(f) für verschiedene Implementierungsdetails gleich?

Implementierungsdetails

Es gibt viele unterschiedliche Arten um ein Problem praktisch zu lösen, z.B. unter Verwendung spezifischer Datenstrukturen.

Sind DTIME(f) und DSPACE(f) für verschiedene Implementierungsdetails gleich?

Antwort: „Nein, aber die meisten Änderungen an der Implementierung haben bestenfalls polynomielle oder konstant-lineare Effekte.“

Ist Komplexitätstheorie praktisch unmöglich?

Die Klassen DTIME(f) und DSPACE(f) unterscheiden sich je nach

- Details des Maschinenmodells
- Details der Eingabekodierung
- Details der Implementierung

Eine exakte Bestimmung solcher Schranken ist oft sehr schwer.

Ist Komplexitätstheorie praktisch unmöglich?

Die Klassen DTIME(f) und DSPACE(f) unterscheiden sich je nach

- Details des Maschinenmodells
- Details der Eingabekodierung
- Details der Implementierung

Eine exakte Bestimmung solcher Schranken ist oft sehr schwer.

Beispiel: Ein naiver Algorithmus zur Matrixmultiplikation liegt in DTIME(n^3).

Ist Komplexitätstheorie praktisch unmöglich?

Die Klassen DTIME(f) und DSPACE(f) unterscheiden sich je nach

- Details des Maschinenmodells
- Details der Eingabekodierung
- Details der Implementierung

Eine exakte Bestimmung solcher Schranken ist oft sehr schwer.

Beispiel: Ein naiver Algorithmus zur Matrixmultiplikation liegt in DTIME(n^3).
Seit Jahrzehnten suchen Forscher nach besseren Lösungen: DTIME($n^{2,808}$) [Strassen, 1969], DTIME($n^{2,796}$) [Pan, 1978], DTIME($n^{2,780}$) [Bini et al., 1979], DTIME($n^{2,522}$) [Schönhage, 1981], DTIME($n^{2,517}$) [Romani, 1982], DTIME($n^{2,496}$) [Coppersmith & Winograd, 1981], DTIME($n^{2,479}$) [Strassen, 1986], DTIME($n^{2,376}$) [Coppersmith & Winograd, 1990], DTIME($n^{2,374}$) [Stothers, 2010] und DTIME($n^{2,373}$) [Williams, 2011].

Ist Komplexitätstheorie praktisch unmöglich?

Die Klassen DTIME(f) und DSPACE(f) unterscheiden sich je nach

- Details des Maschinenmodells
- Details der Eingabekodierung
- Details der Implementierung

Eine exakte Bestimmung solcher Schranken ist oft sehr schwer.

Beispiel: Ein naiver Algorithmus zur Matrixmultiplikation liegt in DTIME(n^3).
Seit Jahrzehnten suchen Forscher nach besseren Lösungen: DTIME($n^{2,808}$) [Strassen, 1969], DTIME($n^{2,796}$) [Pan, 1978], DTIME($n^{2,780}$) [Bini et al., 1979], DTIME($n^{2,522}$) [Schönhage, 1981], DTIME($n^{2,517}$) [Romani, 1982], DTIME($n^{2,496}$) [Coppersmith & Winograd, 1981], DTIME($n^{2,479}$) [Strassen, 1986], DTIME($n^{2,376}$) [Coppersmith & Winograd, 1990], DTIME($n^{2,374}$) [Stothers, 2010] und DTIME($n^{2,373}$) [Williams, 2011].
Vermutete optimale Lösung: DTIME(n^2).

Wie weiter?

Problem:

- Die exakte Bestimmung der Komplexität ist selbst bei einfachsten Algorithmen bisher nicht gelungen.
- Selbst wenn sie gelänge wäre sie von vielen detaillierten Annahmen abhängig, die praktische Computer eventuell nicht erfüllen.

Wie weiter?

Problem:

- Die exakte Bestimmung der Komplexität ist selbst bei einfachsten Algorithmen bisher nicht gelungen.
- Selbst wenn sie gelänge wäre sie von vielen detaillierten Annahmen abhängig, die praktische Computer eventuell nicht erfüllen.

Lösung:

- Wir betrachten noch allgemeinere Sprachklassen, die auch gegenüber polynomiellen Änderungen der Ressourcen robust sind
- Nachteil: Wir können nicht mehr zwischen n und n^{1000} unterscheiden
- Vorteil: Wir müssen nicht mehr zwischen $n^{2,374}$ und $n^{2,373}$ unterscheiden

Wichtige Komplextitätsklassen

Die wichtigen deterministischen Komplexitätsklassen fassen jeweils ganze Familien von zeit- oder speicherbeschränkten Klassen zusammen. Wir erwähnen hier nur die praktisch wichtigsten:

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

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

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

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

*) Anmerkung: Dies ist die praktisch wichtigste Definition von „exponentieller Zeit“. Es gibt daneben auch $\mathsf{E} = \mathsf{ETime} = \bigcup_{d \geq 1} \mathsf{DTime}(2^{dn})$ (exponentielle Zeit mit linearem Exponenten).

LogSpace? Wie soll das gehen?

Für $n > 1$ gilt $\log(n) < n$. Auch beliebige lineare Faktoren können das nur für kleine n kompensieren.

Eine $O(\log(n))$-speicherbeschränkte TM darf also weniger Speicher verwenden als ihre Eingabe benötigt. $\rightsquigarrow$ Wie soll das gehen?

LogSpace? Wie soll das gehen?

Für $n > 1$ gilt $\log(n) < n$. Auch beliebige lineare Faktoren können das nur für kleine n kompensieren.

Eine $O(\log(n))$-speicherbeschränkte TM darf also weniger Speicher verwenden als ihre Eingabe benötigt. $\rightsquigarrow$ Wie soll das gehen?

Man definiert $O(\log(n))$-speicherbeschränkte Turingmaschinen als besondere Mehrband-TMs:

- Das erste Band ist das Eingabeband. Es enthält die Eingabe und darf nur gelesen aber nicht beschrieben werden.
- Das zweite Band ist das Arbeitsband. Es darf beliebig gelesen und beschrieben werden, aber es ist auf $O(\log(n))$-Speicherzellen beschränkt.

Das genügt zur Erkennung von Sprachen. Wenn die TM eine Ausgabe berechnen soll, dann wird dafür ein drittes Ausgabeband verwendet, auf dem man beliebig viele Zeichen einmalig schreiben aber nicht lesen kann.

Beziehungen der Komplexitätsklassen

Eine wichtige Frage der Komplexitätstheorie ist, was man über die Beziehungen der Komplexitätsklassen aussagen kann.

Beziehungen der Komplexitätsklassen

Eine wichtige Frage der Komplexitätstheorie ist, was man über die Beziehungen der Komplexitätsklassen aussagen kann.

Offensichtlich führen (asymptotisch) höhere Ressourcenschranken zu größeren Sprachklassen. Oft ist aber nicht klar, ob man mit mehr Ressourcen auch wirklich mehr (oder einfach nur gleich viele) Probleme lösen kann. Bei unseren Klassen ist das aber bekannt:

Fakt: Es gilt $\mathsf{P} \subsetneq \mathsf{Exp}$ und $\mathsf{LogSpace} \subsetneq \mathsf{PSpace}$.

Beziehungen der Komplexitätsklassen

Eine wichtige Frage der Komplexitätstheorie ist, was man über die Beziehungen der Komplexitätsklassen aussagen kann.

Offensichtlich führen (asymptotisch) höhere Ressourcenschranken zu größeren Sprachklassen. Oft ist aber nicht klar, ob man mit mehr Ressourcen auch wirklich mehr (oder einfach nur gleich viele) Probleme lösen kann. Bei unseren Klassen ist das aber bekannt:

Fakt: Es gilt $\mathsf{P} \subsetneq \mathsf{Exp}$ und $\mathsf{LogSpace} \subsetneq \mathsf{PSpace}$.

Weiterhin kann man Speicher mit Zeit in Beziehung bringen:

- In Zeit n kann man nur n Speicherzellen nutzen
- Alle möglichen Konfigurationen auf n Speicherzellen kann man in exponentieller Zeit (bezüglich n) berechnen

Fakt: Es gilt $\mathsf{LogSpace} \subseteq \mathsf{P} \subseteq \mathsf{PSpace} \subseteq \mathsf{Exp}$.

Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.
Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.
Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Eulers Methode um die Existenz von Eulerpfaden zu entscheiden, kann in LogSpace implementiert werden: wir zählen die Kanten jedes Knotens und speichern die Zahl der Knoten ungeraden Grades, jeweils binär.

Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.
Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Eulers Methode um die Existenz von Eulerpfaden zu entscheiden, kann in LogSpace implementiert werden: wir zählen die Kanten jedes Knotens und speichern die Zahl der Knoten ungeraden Grades, jeweils binär.

Beispiel: Die Suche nach Hamilton-Pfaden ist in ExpTime aber auch in PSpace.

Beispiele

Unsere Klassen sind recht robust: Details der Implementierung haben oft keinen Einfluss auf die Einordnung eines Problems.
Oft genügt eine Implementierungsskizze um zu zeigen, dass eine Sprache in einer dieser Klassen liegt.

Beispiel: Eulers Methode um die Existenz von Eulerpfaden zu entscheiden, kann in LogSpace implementiert werden: wir zählen die Kanten jedes Knotens und speichern die Zahl der Knoten ungeraden Grades, jeweils binär.

Beispiel: Die Suche nach Hamilton-Pfaden ist in ExpTime aber auch in PSpace.

Beispiel: Ein typisches Problem in P haben wir bereits in der Vorlesung Formale Systeme kennengelernt: Erfüllbarkeit von propositionaler Horn-Logik. Unser Resolutionsalgorithmus liefert allerdings keinen Hinweis auf Machbarkeit in LogSpace.

Zusammenfassung und Ausblick

Die Komplexitätstheorie beschäftigt sich mit der Klassifikation entscheidbarer Probleme nach ihrer Schwierigkeit

Um robuste Ergebnisse zu erhalten, die nicht von Implementierungsdetails abhängen, werden oft polynomielle Unterschiede in Kauf genommen

Die wichtigsten deterministischen Komplexitätsklassen sind

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSpace} \subseteq \mathsf{Exp}$$

Was erwartet uns als nächstes?

- Effizient lösbare Probleme: P
- Die kleinste „traditionelle“ Komplexitätsklasse: L
- Weitere Beziehungen zwischen Komplexitäten