Vorlesung: Algorithmische Diskrete Mathematik

TECHNISCHE UNIVERSITÄT DARMSTADT

Raymond Hemmecke

Vertretungsprofessor für Algorithmische Diskrete Mathematik

Inhalt der Vorlesung

- **Komplexitätstheorie**
  - Datenstrukturen und Kodierungsschemata
  - asymptotische Notation, untere und obere Schranken
  - Klassen $\mathcal{P}, \mathcal{NP}, \mathcal{NP}$-vollständig
- **Algorithmen auf Graphen**
  - DFS-Algorithmus (aufspannende Bäume)
  - Greedy-Algorithmus (minimale aufspannende Bäume)
  - Dijkstra, Moore-Bellman, Yen-Variante (kürzeste Wege in Graphen)
  - Ford-Fulkerson (maximale Flüsse in Netzwerken, Matching in bipartiten Graphen)
- **Sortieren in Arrays**
  - Mergesort, Quicksort, Heapsort
  - Divide-and-Conquer
  - untere Komplexitätsschranken für das Sortieren

TECHNISCHE UNIVERSITÄT DARMSTADT

Grundlegende Begriffe der Graphentheorie

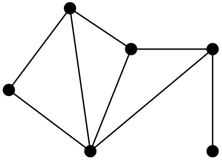
Was ist ein Graph?

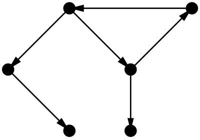
Graph

- Ein ungerichter **Graph** ist ein Paar $G = (V, E)$ disjunkter Mengen, wobei die Elemente von E ungeordnete Paare von Elementen aus V sind.
- Die Elemente aus V heißen **Knoten**, die Elemente aus E **Kanten**.

Digraph

- Ein **gerichter Graph** ist ein Paar $G = (V, A)$ disjunkter Mengen, wobei die Elemente von A geordnete Paare von Elementen aus V sind.
- Die Elemente aus A heißen **Bögen** oder **gerichtete Kanten**.





Nachbarschaftsbeziehungen

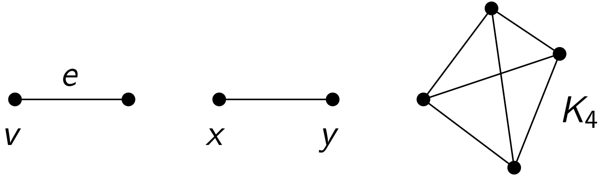
TECHNISCHE UNIVERSITÄT DARMSTADT

Inzidenz, Grad/Valenz

- Ein Knoten v heißt mit einer Kante e **inzident**, wenn $v \in e$ gilt.
- Der **Grad**/die **Valenz** eines Knoten v ist die Anzahl der mit v inzidenten Kanten.

Adjazenz, vollständiger Graph

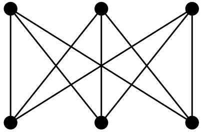
- Zwei Knoten x und y heißen **adjazent** oder **benachbart** in G, wenn $xy \in E(G)$ ist.
- Sind je zwei Knoten von G benachbart, so heißt G **vollständig**.



Bipartite Graphen

Definition

- Ein Graph heißt **bipartit**, wenn sich V disjunkt in zwei Teile V_1 und V_2 zerteilen lässt, so dass jede Kante in E einen Endknoten in V_1 und einen Endknoten in V_2 besitzt.



$K_{3,3}$

Fakt → Beweis als Übungsaufgabe

- Ein Graph ist genau dann bipartit, wenn er nur Kreise gerader Länge enthält.

Was sind Kreise?

TECHNISCHE UNIVERSITÄT DARMSTADT

Pfad, Weg, Kreis

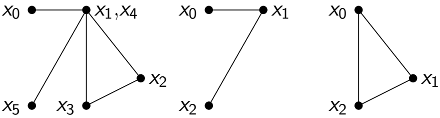
- Ein **Pfad** ist ein nichtleerer Graph $P = (V, E)$ der Form

$$V = \{x_0, x_1, \dots, x_k\}, E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}.$$

- Ein **Weg** ist ein Pfad, in dem alle x_i paarweise verschieden sind.
- Ein **Kreis** ist ein nichtleerer Graph $P = (V, E)$ der Form

$$V = \{x_0, x_1, \dots, x_k\}, E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k, x_kx_0\},$$

  wobei die x_i paarweise verschieden sind.



Pfad Weg Kreis

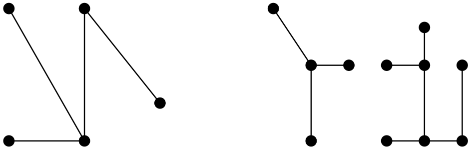
Zusammenhangskomponenten, Bäume und Wälder

Zusammenhängende Graphen

- Ein Graph G heißt **zusammenhängend**, falls es zu jedem Paar $v, w \in V$ einen Weg von v nach w in G gibt.
- Die zusammenhängenden Teile von G heißen **Zusammenhangskomponenten**.

Bäume und Wälder

- Ein **Baum** ist ein zusammenhängender Graph, der keinen Kreis besitzt.
- Ein **Wald** ist ein Graph, der keinen Kreis besitzt.

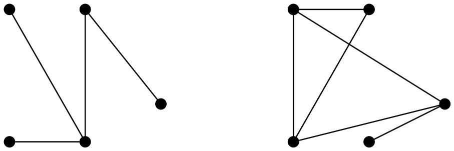


Baum   Wald, 2 Komponenten

Komplementärgraph

Komplementärgraph

- Der zu G **komplementäre Graph** $\bar{G}$ ist der Graph $\bar{G} = (V, \bar{E})$, mit $ij \in \bar{E} \Leftrightarrow ij \notin E$.



Graph Komplement

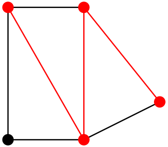
Fakt $\longrightarrow$ Beweis als Übungsaufgabe

- Mindestens einer der Graphen G oder $\bar{G}$ ist zusammenhängend.

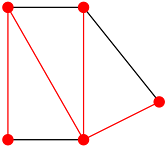
Untergraphen, aufspannende Untergraphen

Untergraph, aufspannende Untergraphen

- $G' = (V', E')$ heißt **Untergraph** von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$ gilt.
- $G' \subseteq G$ heißt **aufspannender Untergraph** von G, falls $V' = V$ gilt.



Graph mit Untergraph



aufspannender Baum

Finden eines aufspannenden Waldes $\rightarrow$ DFS-Algorithmus

- Für gegebenes $G = (V, E)$ finde man einen aufspannenden Wald.

Probleme auf Graphen (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

Finden eines Kreises/Hamiltonischen Kreises

- Entscheide für einen gegebenen Graphen $G = (V, E)$, ob er einen Kreis enthält. Falls ja, gib einen solchen an.
- Entscheide für einen gegebenen Graphen $G = (V, E)$, ob er einen hamiltonischen Kreis enthält. Falls ja, gib einen solchen an. (Ein **hamiltonischer Kreis** ist ein Kreis, der jeden Knoten genau einmal durchläuft.)

Finden eines aufspannenden Waldes $\longrightarrow$ DFS-Algorithmus

- Für gegebenes $G = (V, E)$ finde man einen aufspannenden Wald.

Maximales Matching

- Finde für gegebenen Graphen $G = (V, E)$ die maximale Anzahl von Kanten aus E, so dass je zwei Kanten nicht inzident sind.

Probleme auf Graphen (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Maximale stabile Mengen/Cliquen

- Finde für gegebenen Graphen $G = (V, E)$ die maximale Anzahl von Knoten aus V, so dass je zwei Knoten nicht benachbart sind. $\rightarrow$ stabile Menge
- Finde für gegebenen Graphen $G = (V, E)$ die maximale Anzahl von Knoten aus V, so dass je zwei Knoten benachbart sind. $\rightarrow$ Clique

Kürzeste Wege

- Für gegebenes $G = (V, E)$ und gegebenen Kantengewichten c_{ij} finde man einen kürzesten Weg zwischen zwei gegebenen Knoten v und w.

Färbungsproblem

- Finde für gegebenen Graphen $G = (V, E)$ die minimale Anzahl k von Farben, so dass sich die Knoten von G so mit k Farben färben lassen, dass benachbarte Knoten unterschiedlich gefärbt sind.

Asymptotische Notation

Asymptotische Notation

TECHNISCHE UNIVERSITÄT DARMSTADT

Obere Schranken: O-Notation

- Sei $g : \mathbb{N} \to \mathbb{R}$. Dann bezeichnet

$$O(g) := \{f : \mathbb{N} \to \mathbb{R} : \exists c > 0, n_0 \in \mathbb{N} \text{ mit } |f(n)| \leq c|g(n)| \; \forall n \geq n_0\}$$

die Menge aller Funktionen $f : \mathbb{N} \to \mathbb{R}$, für die zwei positive Konstanten $c \in \mathbb{R}$ und $n_0 \in \mathbb{N}$ existieren, so dass für alle $n \geq n_0$ gilt $|f(n)| \leq c|g(n)|$.

Bemerkungen

- Die asymptotische Notation vernachlässigt Konstanten und Terme niedrigerer Ordnung (wie z.B. die Terme $a_k n^k$ mit $k < m$ im Polynom).

Satz

- Für ein Polynom $f(n) = a_m n^m + ... + a_1 n + a_0$ vom Grade m gilt $f \in O(n^m)$.

Vergleich

TECHNISCHE UNIVERSITÄT DARMSTADT

Die nachfolgende Tabelle zeigt, wie sechs typischen Funktionen anwachsen, wobei die Konstante gleich 1 gesetzt wurde. Wie man feststellt, zeigen die Zeiten vom Typ $O(n)$ und $O(n \log n)$ ein wesentlich schwächeres Wachstum also die anderen.

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

Untere Schranken

Ω- und Θ-Notation

- Seien $f, g : \mathbb{N} \to \mathbb{R}$. Dann ist $f \in \Omega(g)$, wenn zwei positive Konstanten $c \in \mathbb{R}$ und $n_0 \in \mathbb{N}$ existieren, so dass für alle $n \geq n_0$ gilt:

$$|f(n)| \geq c|g(n)|.$$

- $f \in \Theta(g)$, wenn es positive Konstanten $c_1, c_2 \in \mathbb{R}$ und $n_0 \in \mathbb{N}$ gibt, so dass für alle $n \geq n_0$ gilt $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$.
- Falls $f \in \Theta(g)$, dann ist g sowohl eine obere als auch eine untere Schranke für f.

Beispiel: Sequentielle Suche

- Sei $f(n)$ die Anzahl von Vergleichen bei der sequentiellen Suche nach einem bestimmten Wert in einem unsortierten Array mit n Komponenten. Dann ist $f \in O(n)$, da man ja mit n Vergleichen auskommt.
- Andererseits muss man auch jede Komponente überprüfen, denn ihr Wert könnte ja der gesuchte Wert sein. Also $f \in \Omega(n)$ und damit $f \in \Theta(n)$.

Ein weiteres Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Matrixmultiplikation

- Bei der Matrixmultiplikation von $n \times n$ Matrizen ergibt sich die Berechnung eines Eintrags c_{ij} von $C = A \cdot B$ gemäß $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$. Sie erfordert also n Multiplikationen und $n - 1$ Additionen.
- Insgesamt sind für ganz C also n^2 Einträge c_{ij} zu berechnen, und somit $n^2(n + n - 1) = 2n^3 - n^2 = O(n^3)$ arithmetische Operationen insgesamt auszuführen.
- Da jeder Algorithmus für die Matrixmultiplikation n^2 Einträge berechnen muss, folgt andererseits, dass jeder Algorithmus zur Matrixmultiplikation von zwei $n \times n$ Matrizen $\Omega(n^2)$ Operationen benötigt.
- Es klafft zwischen $\Omega(n^2)$ und $O(n^3)$ also noch eine "Komplexitätslücke". Die schnellsten bekannten Algorithmen zur Matrixmultiplikation kommen mit $O(n^{2,376})$ Operationen aus.

Komplexitätstheorie

Was ist ein Problem?

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

- Ein **Problem** ist eine allgemeine Fragestellung, bei der mehrere Parameter offen gelassen sind und für die eine Lösung oder Antwort gesucht wird.
- Ein Problem ist dadurch definiert, dass alle seine Parameter beschrieben werden und dass genau angegeben wird, welche Eigenschaften eine Antwort (Lösung) haben soll.
- Sind alle Parameter eines Problems mit expliziten Daten belegt, dann sprechen wir von einem Problembeispiel (**Instanz**).

Beispiel

- Finde einen kürzesten Hamiltonischen Kreis.
- Offene Parameter: Anzahl Städte, Entfernungen

Algorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

Was ist ein Algorithmus?

- Ein **Algorithmus** ist eine Anleitung zur schrittweisen Lösung eines Problems. Wir sagen, ein Algorithmus A löst ein Problem Π, falls A für alle Problembeispiele $I \in \Pi$, eine Lösung in einer endlichen Anzahl an Schritten findet.
- Ein Schritt ist eine elementare Operation: Addieren, Subtrahieren, Vergleichen, Multiplikation, Division.
- Die Laufzeit eines Algorithmus ist die Anzahl der Schritte, die notwendig sind zur Lösung des Problembeispiels.

Forschungsschwerpunkte zu Algorithmen

- Entwurf von Algorithmen
- Berechenbarkeit: Was kann durch einen Algorithmus berechnet werden?
- Komplexität von Algorithmen
- Korrektheit von Algorithmen

Komplexität von Algorithmen

Was ist Effizienz?

TECHNISCHE UNIVERSITÄT DARMSTADT

Was ist Effizienz?

- Komplexitätstheorie
- Speicher- und Laufzeitkomplexität

Trivial

- Laufzeit eines Algorithmus hängt ab von der "Größe" der Eingabedaten
- Laufzeitanalyse erfordert eine Beschreibung, wie Problembeispiele dargestellt werden (Kodierungsschema)
- Notwendigkeit exakter Definitionen
  - geeignetes Rechnermodell → **Turing-Maschine**

Kodierungsschema

TECHNISCHE UNIVERSITÄT DARMSTADT

Ganze Zahlen

- Ganze Zahlen werden binär dargestellt, d.h. wir schreiben

$$n = \pm \sum_{i=0}^{k} x_i \cdot 2^i,$$

mit $x_i \in \{0,1\}$ und $k = \lfloor \log_2(|n|) \rfloor$.

- D.h. die Kodierungslänge $\langle n \rangle$ einer ganzen Zahl n ist gegeben durch die Formel

$$\langle n \rangle = \lceil \log_2(|n| + 1) \rceil + 1 = \lfloor \log_2(|n|) \rfloor + 2,$$

wobei $+1$ wegen des Vorzeichens $+$ oder $-$.

Rationale Zahlen

- Sei $r \in \mathbb{Q}$. Dann existieren $p \in \mathbb{Z}$ und $q \in \mathbb{Z}$, teilerfremd, mit $r = \frac{p}{q}$.

$$\langle r \rangle = \langle p \rangle + \langle q \rangle$$

Kodierungsschema (2)

Vektoren

- Für $x = (x_1, \dots, x_n)^\mathsf{T} \in \mathbb{Q}^n$ ist
$$\langle x \rangle = \sum_{i=1}^{n} \langle x_i \rangle .$$

Matrizen

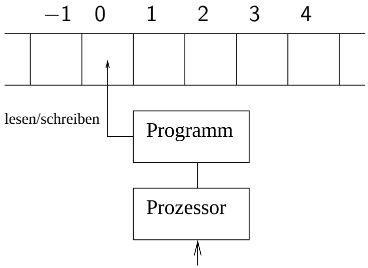
- Für $A \in \mathbb{Q}^{m \times n}$ ist
$$\langle A \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle .$$

Rechnermodell

Nach Festlegung der Kodierungsvorschrift muss ein Rechnermodell entworfen werden, auf dem unsere Speicher- und Laufzeitberechnungen durchgeführt werden.

In der Komplexitätstheorie: **Turing-Maschine** (ein ganz normaler Computer)

Unendliches Band, auf dem Lese-/Schreiboperationen durchgeführt werden



stellt Operationen zur Verfügung:
+, −, *, /, vergleichen, löschen, schreiben, lesen

Ablauf eines Algorithmus auf der Turing-Maschine

Algorithmus A soll Problembeispiel I des Problems Π lösen. Alle Problembeispiele liegen in kodierter Form vor.

Inputlänge

- Die Anzahl der Speicherplätze, die nötig sind, um I vollständig anzugeben, heißt **Inputlänge**, $\langle I \rangle$.

Der Algorithmus liest die Daten und beginnt dann, Operationen auszuführen, d.h. Zahlen zu berechnen, zu speichern, zu löschen.

Speicherbedarf

- Die Anzahl der Speicherplätze, die während der Ausführung des Algorithmus mindestens einmal benutzt werden, nennen wir **Speicherbedarf** von A zur Lösung von I.

Laufzeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Informell

- Die **Laufzeit** von A zur Lösung von I ist die Anzahl elementarer Operationen. Dazu zählen $+$, $-$, $*$, $/$, Vergleich, Löschen, Schreiben, Lesen von rationalen Zahlen.

Dies ist jedoch zu unpräzise!

- Zur Darstellung der entsprechenden Zahlen werden mehrere Bits benötigt.

$\longrightarrow$ Für jede Operation müssen wir mehrere Bits zählen.

Korrekte Definition

- Die **Laufzeit** von A zur Lösung von I ist definiert durch die Anzahl elementarer Operationen, die A ausgeführt hat, um I zu lösen, multipliziert mit der größten Kodierungslänge der während der Berechnung aufgetretenen ganzen oder rationalen Zahl.

Laufzeitfunktion, Speicherplatzfunktion (worst-case)

TECHNISCHE UNIVERSITÄT DARMSTADT

Laufzeitfunktion, polynomielle Laufzeit

- Sei A ein Algorithmus zur Lösung eines Problems Π. Die Funktion $f_A : \mathbb{N} \to \mathbb{N}$,

$$f_A(n) = \max\{\text{Laufzeit von A zur Lösung von } I : I \in \Pi, \langle I \rangle \leq n\}$$

heißt **Laufzeitfunktion** von A.

- Der Algorithmus A hat eine **polynomiale Laufzeit** (kurz: A ist ein polynomialer Algorithmus), wenn es ein Polynom $p : \mathbb{N} \to \mathbb{N}$ gibt mit $f_A(n) \leq p(n)$ für alle $n \in \mathbb{N}$.

Speicherplatzfunktion, polynomieller Speicherbedarf

- Die **Speicherplatzfunktion** $s_A : \mathbb{N} \to \mathbb{N}$ von A ist definiert durch

$$s_A(n) = \max\{\text{Speicherbedarf von } A \text{ zur Lösung von } I : I \in \Pi, \langle I \rangle \leq n\}.$$

- Der Algorithmus A hat polynomialen Speicherbedarf, falls es ein Polynom $q : \mathbb{N} \to \mathbb{N}$ gibt mit $s_A(n) \leq q(n)$ für alle $n \in \mathbb{N}$.

Weitere Bemerkungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Bemerkungen

- Nehmen wir an, wir ermitteln die Rechenzeit $f(n)$ für einen bestimmten Algorithmus. Die Variable n kann z.B. die Anzahl der Ein- und Ausgabewerte sein, ihre Summe, oder auch die Größe eines dieser Werte. Da $f(n)$ maschinenabhängig ist, genügt eine a priori Analyse nicht. Jedoch kann man mit Hilfe einer a priori Analyse ein g bestimmen, so dass $f \in O(g)$.
- Wenn wir sagen, dass ein Algorithmus eine Rechenzeit $O(g)$ hat, dann meinen wir damit folgendes: Wenn der Algorithmus auf unterschiedlichen Computern mit den gleichen Datensätzen läuft, und diese Größe n haben, dann werden die resultierenden Laufzeiten immer kleiner sein als eine Konstante mal $|g(n)|$. Bei der Suche nach der Größenordnung von f werden wir darum bemüht sein, das kleinste g zu finden, so dass $f \in O(g)$ gilt.

Die Klassen $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-vollständig

Die Klasse $\mathcal{P}$

TECHNISCHE UNIVERSITÄT DARMSTADT

Was ist ein Entscheidungsproblem?

- Problem, das nur zwei mögliche Antworten besitzt, "ja" oder "nein".
- Beispiele: "Ist n eine Primzahl?" oder "Enthält G einen Kreis?"

Die Klasse $\mathcal{P}$: informelle Definition

- Klasse der Entscheidungsprobleme, für die ein polynomialer Lösungsalgorithmus existiert

Die Klasse $\mathcal{P}$: formale Definition

- Gegeben ein Kodierungsschema E und ein Rechnermodell M.
- Π sei ein Entscheidungsproblem, wobei jede Instanz aus Π durch Kodierungsschema E kodierbar sei.
- Π gehört zur Klasse $\mathcal{P}$ (bzgl. E und M), wenn es einen auf M implementierbaren Algorithmus zur Lösung aller Problembeispiele aus Π gibt, dessen Laufzeitfunktion auf M polynomial ist.

Die Klasse $\mathcal{NP}$

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

- Enthält G einen Kreis? $\longrightarrow$ "einfach" $\longrightarrow \mathcal{P}$
- Enthält G einen hamiltonischen Kreis? $\longrightarrow$ "schwieriger" $\longrightarrow \mathcal{NP}$

Die Klasse $\mathcal{NP}$: informelle Definition

- Entscheidungsproblem Π gehört zur Klasse $\mathcal{NP}$, wenn es folgende Eigenschaft hat:
  - Ist die Antwort "ja" für $I \in \Pi$, dann kann Korrektheit dieser Aussage mit Hilfe eines geeigneten Zusatzobjekts in polynomialer Laufzeit überprüft werden.

Beispiel: "Enthält G einen hamiltonischen Kreis?"

- Geeignetes Zusatzobjekt wäre ein hamiltonischer Kreis.
- Es muss nun lediglich geprüft werden, ob der angegebene Kreis tatsächlich ein hamiltonischer Kreis von G ist.

Die Klasse $\mathcal{NP}$ (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Formale Definition

- Entscheidungsproblem Π gehört zu $\mathcal{NP}$, wenn es die folgende Eigenschaften hat:
  (a) Für jedes Problembeispiel $I \in \Pi$, für das die Antwort "ja" lautet, gibt es mindestens ein Objekt Q, mit dessen Hilfe die Korrektheit der "ja"-Antwort überprüft werden kann.
  (b) Es gibt einen Algorithmus, der Problembeispiele $I \in \Pi$ und Zusatzobjekte Q als Input akzeptiert und der in einer Laufzeit, die polynomial in $\langle I \rangle$ ist, überprüft, ob Q ein Objekt ist, aufgrund dessen Existenz eine "ja"-Antwort für I gegeben werden muss.

Bemerkungen

- Es wird keine Aussage über die Berechenbarkeit eines geeigneten Q gemacht. Q kann auch geraten werden.
- Da der Überprüfungsalgorithmus Q lesen muss, muss $\langle Q \rangle$ polynomial in $\langle I \rangle$ sein!
- Es wird keine Aussage zu möglicher "nein"-Antwort gemacht. $\rightarrow$ coNP

Charakterisierung besonders schwieriger Probleme in $\mathcal{NP}$

Polynomiale Transformation von Problemen

- Gegeben seien zwei Entscheidungsprobleme Π und Π'.
- Eine polynomiale Transformation von Π in Π' ist ein polynomialer Algorithmus, der, gegeben ein (kodiertes) Problembeispiel $I \in \Pi$, ein (kodiertes) Problembeispiel $I' \in \Pi'$ produziert, so dass folgendes gilt:
  - Die Antwort auf I ist genau dann "ja", wenn die Antwort auf I' "ja" ist.

Die Klasse $\mathcal{NP}$-vollständig

- Ein Entscheidungsproblem Π heißt $\mathcal{NP}$-**vollständig**, falls $\Pi \in \mathcal{NP}$ und falls jedes andere Problem aus $\mathcal{NP}$ polynomial in Π transformiert werden kann.

Bemerkung und Fragen

TECHNISCHE UNIVERSITÄT DARMSTADT

Bemerkung

- Falls ein $\mathcal{NP}$-vollständiges Entscheidungsproblem Π in polynomialer Zeit gelöst werden kann, so gilt das auch für jedes andere Problem aus $\mathcal{NP}$:

$$\Pi \quad \mathcal{NP} - \text{vollständig und } \Pi \in \mathcal{P} \Rightarrow \mathcal{P} = \mathcal{NP}.$$

- Diese Eigenschaft zeigt, dass (bzgl. polynomialer Lösbarkeit) kein Problem in $\mathcal{NP}$ schwieriger ist als ein $\mathcal{NP}$-vollständiges.

Existieren $\mathcal{NP}$-vollständige Probleme? Ja.

- 3-SAT: $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\ldots) \wedge \ldots = \text{TRUE}$

Offene Fragen

$$\begin{aligned} \mathcal{P} &= \mathcal{NP} \cap \text{co}\mathcal{NP} \\ \mathcal{NP} &= \text{co}\mathcal{NP} \\ \mathcal{P} &\neq \mathcal{NP} \end{aligned}$$

Komplexität von Optimierungsproblemen

Die Klasse $\mathcal{NP}$-schwer

TECHNISCHE UNIVERSITÄT DARMSTADT

Transformation in Entscheidungsproblem

- Sei O ein Maximierungsproblem (Minimierungsproblem). Man legt zusätzlich zur Instanz I noch eine Schranke B fest und stellt die Frage:
  - Gibt es für I eine Lösung, deren Wert nicht kleiner (größer) als B ist?
- Damit ist ein Optimierungsproblem polynomial in ein Entscheidungsproblem transformierbar.

Die Klasse $\mathcal{NP}$-schwer: formale Definition

- Ein Optimierungsproblem O heißt $\mathcal{NP}$-**schwer**, falls es ein $\mathcal{NP}$-vollständiges Entscheidungsproblem Π gibt, so dass Π in polynomialer Zeit gelöst werden kann, wenn O in polynomialer Zeit gelöst werden kann.

Beispiele

- Rucksackproblem, Maschinenbelegung, stabile Menge maximaler Kardinalität, Clique maximaler Kardinalität, minimale Knotenüberdeckung

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Traveling Salesman Problem

- Input: Instanz I ist charakterisiert durch eine Distanzmatrix $c_{ij} \in \mathbb{Z}\ \forall i, j \in V$, $|V| = n$.
- Problem: "Finde einen bzgl. c minimalen Hamiltonischen Kreis in K_n."
- TSP-Entscheidungsproblem:
  - Input wie beim TSP, $B \in \mathbb{Z}$.
  - Problem: "Existiert eine TSP-Tour der Länge $\leq B$?"

Lemma

- Polynomialer Algorithmus zur Lösung des TSP-Entscheidungsproblems würde polynomialen Algorithmus zur Lösung des TSP implizieren.

Lösung: Binäre Suche

TECHNISCHE UNIVERSITÄT DARMSTADT

Beweis

- Länge eines hamiltonischen Kreises in K_n ist ein ganzzahliger Wert im Intervall $[ns, nt]$, wobei $s = \min\{c_{ij} : i, j \in V, i \neq j\}$ und $t = \max\{c_{ij} : i, j \in V, i \neq j\}$.
- Rufe TSP-Entscheidungsproblem mit $B = \lfloor \frac{n(s+t)}{2} \rfloor$ auf, passe obere Schranke nt oder untere Schranke ns je nach Ergebnis an und iteriere.
- Insgesamt reichen demnach $\log(nt - ns)$ Iterationen aus. $\square$