VL-12: P versus NP

(Berechenbarkeit und Komplexität, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

Organisatorisches

- Nächste Vorlesungen:
  Freitag, Dezember 14, 16:30–18:00 Uhr, Audimax
  Donnerstag, Dezember 20, 12:30–14:00 Uhr, Aula
  Donnerstag, Januar 11, 12:30–14:00 Uhr, Aula
- **Keine Vorlesung:** Freitag, Dezember 21
- Webseite:
  http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php
  ($\longrightarrow$ Arbeitsheft zur Berechenbarkeit)

Wiederholung

Wdh.: Primitiv rekursive Funktionen

Die primitiv rekursiven Funktionen bilden eine Unterfamilie der Funktionen $\mathbb{N}^k \to \mathbb{N}$ mit $k \geq 1$.

Sie sind induktiv definiert, und werden durch zwei Operationen aus den sogenannten Basisfunktionen zusammengebaut.

Thoralf Skolem (1923):
"Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich",
Skrifter utgit av Videnskapsselskapet i Kristiania 6, pp 1–38.

Wdh.: μ-rekursive Funktionen

Definition

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von (partiellen und totalen) Funktionen,

- die die Basisfunktionen (konstant Funktionen; identischen Abbildungen; Nachfolgerfunktion) enthält und
- die abgeschlossen ist unter Komposition, primitiver Rekursion und Anwendung des μ-Operators.

Wdh.: Äquivalenz zu LOOP und WHILE

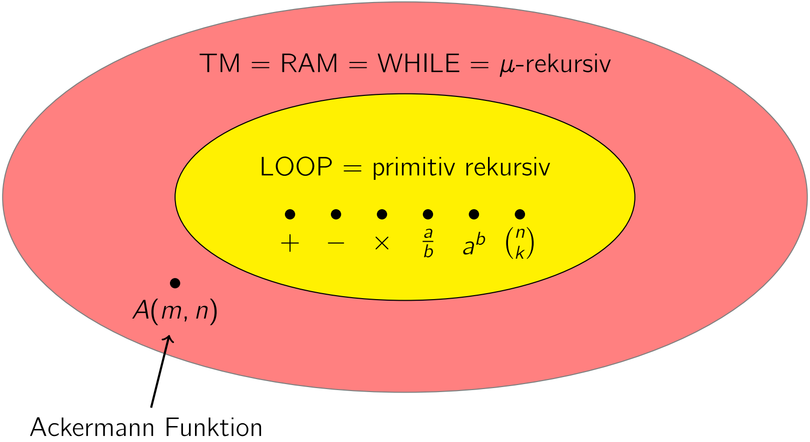
Satz

Die Menge der primitiv rekursiven Funktionen
fällt mit der Menge der LOOP-berechenbaren zusammen.

Satz

Die Menge der μ-rekursiven Funktionen
fällt mit der Menge der WHILE-berechenbaren (Turing-berechenbaren; RAM-berechenbaren) Funktionen zusammen.

Wdh.: Landschaftsbild



Wdh.: Kostenmodelle der RAM

Modelle für die Rechenzeit einer RAM

- Uniformes Kostenmaß:
  Jeder Schritt zählt als eine Zeiteinheit.
- Logarithmisches Kostenmaß:
  Die Laufzeitkosten eines Schrittes sind proportional zur binären Länge der Zahlen in den angesprochenen Registern.

Vorlesung VL-12
P versus NP

- Polynomielle Algorithmen
- Die Komplexitätsklasse P
- Non-deterministische Turingmaschinen
- Die Komplexitätsklasse NP
- Katalog von Problemen in NP
- P versus NP

Komplexitätstheorie

Die Komplexitätstheorie versucht,
(entscheidbare!) algorithmische Probleme
nach ihrem Bedarf an Berechnungsressourcen zu klassifieren
und sie in Komplexitätsklassen einzuteilen.

Berechnungsressourcen:
Rechenzeit und Speicherplatz (als Funktion der Eingabelänge)

Polynomielle Algorithmen

Polynomielle Zeit

Definition: Worst Case Laufzeit eines Algorithmus

Die Worst Case Laufzeit $t_A(n)$ eines Algorithmus A misst
die maximalen Laufzeitkosten auf Eingaben der Länge n
bezüglich des logarithmischen Kostenmaßes der RAM.

Definition: Polynomielle Algorithmen

Die Worst Case Laufzeit $t_A(n)$ eines Algorithmus A ist polynomiell beschränkt, falls gilt

$$\exists \alpha \in \mathbb{N} : \; t_A(n) \in O(n^\alpha)$$

Einen Algorithmus mit polynomiell beschränkter Worst Case Laufzeit bezeichnen wir als polynomiellen Algorithmus oder auch als Polynomialzeitalgorithmus.

Beispiel: $O(n)$; $O(n \log n)$; $O(n^3)$; $O(n^{100})$

Beispiel: Sortieren

Problem: SORTIEREN

Eingabe: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ in Binärdarstellung

Ausgabe: Die aufsteigend sortierte Folge der Eingabezahlen

Satz

SORTIEREN kann in polynomieller Zeit gelöst werden.

Beweis:

- Wir lösen das Problem mit MergeSort oder HeapSort
- Laufzeit im uniformen Kostenmaß: $O(n \log n)$
- Laufzeit im logarithmischen Kostenmaß:
  - $O(\ell n \log n)$, wobei $\ell = \max_{1 \leq i \leq n} \log a_i$
  - Für die Gesamtlänge L der Eingabe gilt $L \geq \ell$ und $L \geq n$
  - Somit ist die Laufzeit durch $\ell n \log n \leq L^3$ beschränkt

Weitere Beispiele

Die folgenden Probleme können in polynomieller Zeit gelöst werden:

- Eulerkreis
- Kürzester Weg
- Minimaler Spannbaum
- Maximaler Fluss
- Maximum Matching
- Grösster gemeinsamer Teiler
- Konvexe Hülle in 2D

- Primzahltest

Die Klasse P

Definition: Komplexitätsklasse P

P ist die Klasse aller Entscheidungsprobleme,
für die es einen polynomiellen Algorithmus gibt.

Anmerkungen:

- **P** steht für **P**olynomiell
- **P** enthält ausschliesslich Entscheidungsprobleme
- Statt der RAM könnte man in der Definition der polynomiellen Laufzeit und der polynomiellen Algorithmen genauso gut die TM verwenden: RAM (mit logarithmischem Kostenmaß) und TM simulieren einander ja mit polynomiellem Zeitverlust
- Polynomielle Algorithmen werden oft als effiziente Algorithmen bezeichnet, und **P** als die Klasse der effizient lösbaren Probleme.

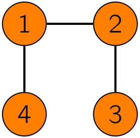
Beispiel: Graphzusammenhang (1)

Problem: Graphzusammenhang

Eingabe: Ein ungerichteter Graph $G = (V, E)$

Frage: Ist G zusammenhängend?

Anmerkung: Bei Graphproblemen gehen wir grundsätzlich davon aus, dass der Graph in Form einer Adjazenzmatrix gegeben ist.



$$\rightsquigarrow \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \rightsquigarrow \quad 0101101001001000$$

Beispiel: Graphzusammenhang (2)

Satz

Graphzusammenhang $\in P$.

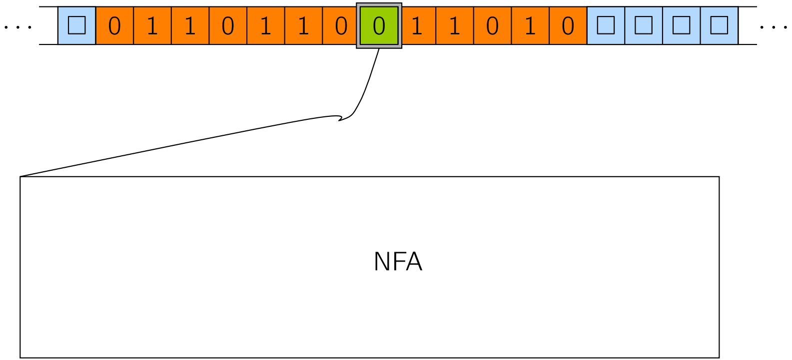
Beweis

- Wir lösen das Problem mit Tiefensuche (DFS).
- Laufzeit im uniformen Kostenmaß: $O(|V| + |E|) = O(|V|^2)$
- Laufzeit im logarithmischen Kostenmaß: $O(|V|^2 \cdot \log |V|)$
- Die Gesamtlänge der Eingabe ist $L = |V|^2$
- Die Gesamtlaufzeit liegt somit in

$$O(|V|^2 \log |V|) \;\subseteq\; O(L \log L) \;\subseteq\; O(L^2)$$

Die non-deterministische Turingmaschine (NTM)

NTM (1): Illustration



NTM (2): Definition

Definition: Non-deterministische Turingmaschine (NTM)

Eine non-deterministische Turingmaschine (NTM) verfügt über

- ein beidseitig unbeschränktes Arbeitsband,
- einen Lese/Schreibkopf, und
- einen Mechanismus, der die Zustandsüberführungen steuert.

Der einzige Unterschied zur deterministischen Turingmaschine TM besteht darin, dass die Zustandsüberführungen bei der NTM nicht durch eine Funktion sondern durch eine Relation gesteuert werden:

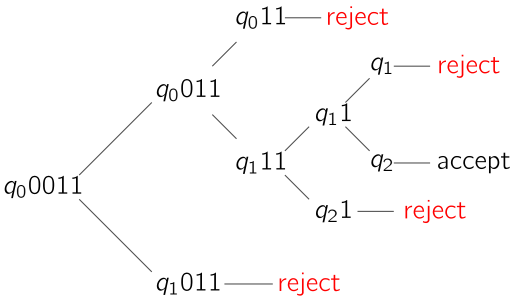
$$\delta \quad \subseteq \quad ((Q \setminus \{\bar{q}\}) \times \Gamma) \;\times\; (Q \times \Gamma \times \{L, R, N\})$$

Berechnungen auf einer NTM sind nicht deterministisch, da es zu einer Konfiguration mehrere direkte Nachfolgekonfigurationen geben kann.

NTM (3a): Berechnungsbaum

δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Diese NTM hat auf der Eingabe $w = 0011$ folgenden Berechnungsbaum:



NTM (3b): Berechnungsbaum

Rechenweg einer NTM = Konfigurationsfolge, die mit Startkonfiguration beginnt und mit Nachfolgekonfigurationen fortgesetzt wird, bis eine Endkonfiguration im Zustand $\bar{q}$ erreicht wird

Die möglichen Rechenwege einer NTM M auf einer Eingabe können in einem Berechnungsbaum zusammengefasst werden:

- Die Knoten des Baumes entsprechen Konfigurationen
- Die Wurzel des Baumes entspricht der Startkonfiguration
- Die Kinder einer Konfiguration entsprechen den möglichen Nachfolgekonfigurationen

Der maximale Verzweigungsgrad des Berechnungsbaumes ist

$$\Delta \; := \; \max \left\{ |\delta(q,a)| \;\middle|\; q \in Q \setminus \{\bar{q}\},\; a \in \Gamma \right\}.$$

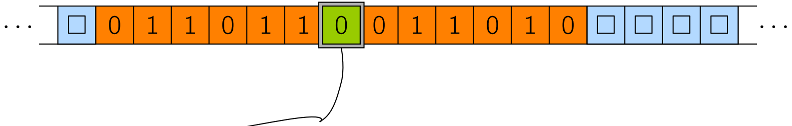
Anmerkung: Δ hängt nur von M und nicht von der Eingabe ab

NTM (4a): Akzeptanzverhalten

Definition: Akzeptanzverhalten der NTM

Eine NTM M akzeptiert die Eingabe $x \in \Sigma^*$,
falls es mindestens einen Rechenweg von M gibt,
der in eine Konfiguration mit akzeptierendem Zustand führt.

Die von der NTM M erkannte Sprache $L(M)$
besteht aus allen von M akzeptierten Wörtern.



δ	0	1	B
q_0	$\{(q_0, B, R), (q_1, B, R)\}$	{reject}	{reject}
q_1	{reject}	$\{(q_1, B, R), (q_2, B, R)\}$	{reject}
q_2	{reject}	{reject}	{accept}

Welche Sprache wird von dieser NTM erkannt?

$$L(M) = \{0^i 1^j \mid i \geq 1, j \geq 1\}$$

NTM (5): Laufzeit

Definition: Laufzeit der NTM

Die Laufzeit einer NTM M auf einer Eingabe x ist wie folgt definiert.

- Falls $x \in L(M)$, so ist die Laufzeit $T_M(x)$ die Länge des kürzesten akzeptierenden Rechenweges von M auf x
- Falls $x \notin L(M)$, so ist die Laufzeit $T_M(x) = 0$.

Die Worst Case Laufzeit $t_M(n)$ der NTM M auf Eingaben der Länge $n \in \mathbb{N}$ ist definiert als $t_M(n) := \max\{T_M(x) \mid x \in \Sigma^n\}$

Die Komplexitätsklasse NP

Die Klasse NP: Definition

Definition: Komplexitätsklasse NP

NP ist die Klasse aller Entscheidungsprobleme,
die durch eine NTM M erkannt werden,
deren Worst Case Laufzeit $t_M(n)$ polynomiell beschränkt ist.

NP steht für **N**on-deterministisch **P**olynomiell

Beispiel

Problem: CLIQUE

Eingabe: Ein ungerichteter Graph $G = (V, E)$; eine Zahl k

Frage: Enthält G eine Clique mit $\geq k$ Knoten?

Satz

CLIQUE $\in NP$

Beweis: Wir beschreiben eine NTM M mit $L(M) = \text{CLIQUE}$.

- Syntaktisch inkorrekte Eingaben werden verworfen
- M rät non-deterministisch einen 0-1-String y der Länge $|V|$
- M akzeptiert, falls der String y genau k Einsen enthält und falls die Knotenmenge $C = \{i \in V \mid y_i = 1\}$ eine Clique bildet

Korrektheit: $\exists$ akzeptierender Rechenweg $\iff$ G enthält k-Clique
Laufzeit: Jede Phase kostet polynomielle Zeit

Alternative Charakterisierung von NP

Satz (Zertifikat Charakterisierung von NP)

Eine Sprache $L \subseteq \Sigma^*$ liegt genau dann in NP,
wenn es einen polynomiellen (deterministischen) Algorithmus V
und ein Polynom p mit der folgenden Eigenschaft gibt:

$$x \in L \iff \exists y \in \{0,1\}^*,\ |y| \le p(|x|) :\ V \text{ akzeptiert } y\#x$$

Anmerkungen:

- Der polynomielle Algorithmus V heisst auch Verifizierer
- Das Wort $y \in \{0,1\}^*$ heisst auch Zertifikat

Beweis: NTM $\Rightarrow$ Zertifikat

$$x \in L \iff \exists y \in \{0,1\}^*,\ |y| \le p(|x|) :\ V \text{ akzeptiert } y\#x$$

- Es sei M eine NTM, die L in polynomieller Zeit erkennt
- Die Laufzeit von M sei beschränkt durch ein Polynom q
- Der maximale Verzweigungsgrad eines Berechnungsbaumes sei Δ

Konstruktion von Zertifikat und Verifizierer:

- Für die Eingabe $x \in L$ betrachten wir den kürzesten akzeptierenden Rechenweg im Berechnungsbaum
- Das Zertifikat y kodiert den akzeptierenden Rechenweg Schritt für Schritt, mit $d := \log_2 \Delta$ Bits pro Verzweigung
- Das Zertifikat hat polynomielle Länge $|y| \le d \cdot q(|x|)$
- Der Verifizierer V erhält $y\#x$ und simuliert den Rechenweg der NTM M für die Eingabe x

Beweis: NTM $\Leftarrow$ Zertifikat

$$x \in L \iff \exists y \in \{0,1\}^*,\ |y| \le p(|x|):\ V \text{ akzeptiert } y\#x$$

- Es sei V ein Verifizierer mit polynomieller Laufzeitschranke
- Das Polynom p beschränkt die Länge des Zertifikats

Konstruktion von NTM:

- Für eine Eingabe x rät M zunächst non-deterministisch das Zertifikat $y \in \{0,1\}^*$ mit $|y| \le p(|x|)$
- Dann simuliert M den Verifizierer V auf dem Wort $y\#x$, und akzeptiert wenn der Verifizierer akzeptiert
- Das Zertifikat wird in polynomieller Zeit $p(|x|)$ geraten.
  Die Zeit für die Simulation ist polynomiell beschränkt in der polynomiellen Laufzeit des Verifizierers.

Katalog von Problemen in NP

Probleme in NP (1): Satisfiability

Problem: Satisfiability (SAT)

Eingabe: Eine Boole'sche Formel φ in CNF über einer Boole'schen Variablenmenge $X = \{x_1, \ldots, x_n\}$

Frage: Existiert eine Wahrheitsbelegung von X, die φ erfüllt?

- Literal: positive oder negierte Variable
- Klausel: ODER-Verknüpfung von einigen Literalen

Beispiele

$$\begin{aligned}
\varphi_1 &= (x \lor y \lor z) \land (\neg x \lor \neg y \lor \neg z) \\
\varphi_1 &= (x + y + z)\,(\overline{x} + \overline{y} + \overline{z})
\end{aligned}$$

$$\begin{aligned}
\varphi_2 &= (x \lor y) \land (\neg x \lor y) \land (x \lor \neg y) \land (\neg x \lor \neg y) \\
\varphi_2 &= (x + y)\,(\overline{x} + y)\,(x + \overline{y})\,(\overline{x} + \overline{y})
\end{aligned}$$

Frage: Wie sieht ein NP-Zertifikat für SAT aus?

Probleme in NP (2): Clique / Independent Set / VC

Problem: CLIQUE

Eingabe: Ein ungerichteter Graph $G = (V, E)$; eine Zahl k

Frage: Enthält G eine Clique mit $\geq k$ Knoten?

Problem: Independent Set (INDEP-SET)

Frage: Enthält G eine unabhängige Menge mit $\geq k$ Knoten?

Problem: Vertex Cover (VC)

Frage: Enthält G ein Vertex Cover mit $\leq k$ Knoten?

- Unabhängige Menge (independent set) $S \subseteq V$: spannt keine Kanten
- Vertex Cover $S \subseteq V$: berührt alle Kanten

Frage: Wie sieht NP-Zertifikat für CLIQUE / INDEP-SET / VC aus?

Probleme in NP (3): Hamiltonkreis / TSP

Problem: Hamiltonkreis (Ham-Cycle)

Eingabe: Ein ungerichteter Graph $G = (V, E)$

Frage: Besitzt G einen Hamiltonkreis?

Problem: Travelling Salesman Problem (TSP)

Eingabe: Städte $1, \ldots, n$; Distanzen $d(i, j)$; eine Zahl γ

Frage: Gibt es eine Rundreise (TSP-Tour) mit Länge höchstens γ?

Fragen:
Wie sieht ein NP-Zertifikat für Ham-Cycle aus?
Wie sieht ein NP-Zertifikat für TSP aus?

Probleme in NP (4): Exact Cover

Problem: Exact Cover (EX-COVER)

Eingabe: Eine endliche Menge X; Teilmengen $S_1, \ldots, S_m$ von X

Frage: Existiert eine Indexmenge $I \subseteq \{1, \ldots, m\}$,
sodass die Mengen S_i mit $i \in I$ eine Partition von X bilden?

Frage: Wie sieht ein NP-Zertifikat für Exact Cover aus?

Probleme in NP (5): SUBSET-SUM / PARTITION

Problem: SUBSET-SUM

Eingabe: Positive ganze Zahlen $a_1, \ldots, a_n$; eine ganze Zahl b

Frage: Existiert eine Indexmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = b$?

Problem: PARTITION

Eingabe: Positive ganze Zahlen $a_1, \ldots, a_n$; mit $\sum_{i=1}^{n} a_i = 2A$

Frage: Existiert eine Indexmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = A$?

Fragen:
Wie sieht ein NP-Zertifikat für SUBSET-SUM aus?
Wie sieht ein NP-Zertifikat für PARTITION aus?

P versus NP

P versus NP

Übung

Beweisen Sie, dass $P \subseteq NP$ gilt

Argument 1: Über TM und NTM
Argument 2: Wie sieht das NP-Zertifikat für ein Problem in P aus?

Was haben wir bis jetzt gesehen?

Die Komplexitätsklasse P

Die Klasse *P* enthält alle Entscheidungsprobleme,
die effizient auf dem Computer gelöst werden können

Intuitiv: *P* enthält die Probleme, die wir gut verstehen können und die wir in vernünftiger Rechenzeit erledigen können

Die Komplexitätsklasse NP

Die Klasse *NP* enthält alle Entscheidungsprobleme,
für die eine kurze Lösung existiert,
und deren kurze Lösung wir effizient überprüfen können
(wenn wir diese Lösung erst einmal gezeigt bekommen)

Intuitiv: *NP* enthält so ziemlich alle Probleme, die nach einem konkreten Lösungsobjekt fragen

Die grosse offene Frage der Informatik

P=NP ?

Falls die Lösung eines Problems einfach zu überprüfen ist,
ist es dann auch immer einfach, die Lösung zu entdecken?

Mögliche Konsequenzen

Falls P=NP ist:

- Viele schwierige Probleme aus Wirtschaft und Industrie können schnell gelöst werden
- Perfekte Fahrpläne, Produktionspläne, Transportpläne, etc
- Die Mathematik erreicht eine neue Stufe: Wenn es für ein Theorem einen kurzen Beweis gibt, so können wir diesen Beweis auch finden
- Die moderne Kryptographie bricht zusammen

Falls P≠NP ist:

- Schwierige Probleme aus Wirtschaft und Industrie können nur durch viel Rechenzeit und Expertenwissen attackiert werden
- Perfekte Lösungen für schwierige Probleme mit grossen Datenmengen sind nicht zu erwarten
- Mathematik und Kryptographie arbeiten genauso weiter wie bisher

Eine Million Dollar Preisgeld

Im Jahr 2000 hat das Clay Mathematics Institute (CMI) je eine Million Dollar Preisgeld für die Lösung der folgenden sieben Probleme angeboten:

- P versus NP Problem
- Hodge Vermutung
- Poincaré Vermutung (erledigt 2002–2006; Grigori Perelman)
- Riemann'sche Vermutung
- Quantenversion der Yang-Mills Gleichungen
- Navier-Stokes Gleichungen
- Birch und Swinnerton-Dyer Vermutung