

# Berechenbarkeit und Komplexität

## Polynomielle Komplexitätsklassen

Prof. Berthold Vöcking  
präsentiert von Prof. Joost-Pieter Katoen

15. Dezember 2008

# Definition von Polynomialzeitalgorithmus

## Definition (worst case Laufzeit eines Algorithmus)

*Die worst case Laufzeit  $t_A(n)$ ,  $n \in \mathbb{N}$ , eines Algorithmus  $A$  entspricht den maximalen Laufzeitkosten auf Eingaben der Länge  $n$  bezüglich des logarithmischen Kostenmaßes der RAM.*

## Definition (Polynomialzeitalgorithmus)

*Wir sagen, die worst case Laufzeit  $t_A(n)$  eines Algorithmus  $A$  ist polynomiell beschränkt, falls gilt*

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

*Einen Algorithmus mit polynomiell beschränkter worst case Laufzeit bezeichnen wir als Polynomialzeitalgorithmus.*

# Definition der Klasse P

## Definition (Komplexitätsklasse P)

*P ist die Klasse der Probleme, für die es einen Polynomialzeitalgorithmus gibt.*

### Anmerkungen:

- Alternativ kann man sich auch auf die Laufzeit einer TM beziehen, da sich RAM und TM gegenseitig mit polynomiellen Zeitverlust simulieren können.
- Polynomialzeitalgorithmen werden häufig auch als „effiziente Algorithmen“ bezeichnet.
- P ist in diesem Sinne die Klasse derjenigen Probleme, die effizient gelöst werden können.

# Beispiele für Probleme in P

- Sortieren
- Kürzeste Wege
- Minimaler Spannbaum
- Graphzusammenhang
- Maximaler Fluss
- Maximum Matching
- Lineare Programmierung
- Größter Gemeinsamer Teiler
- Primzahltest

# Warum ist Sortieren in P?

## Problem (Sortieren)

**Eingabe:**  $N$  Zahlen  $a_1, \dots, a_N \in \mathbb{N}$

**Ausgabe:** *aufsteigend sortierte Folge der Eingabezahlen*

**Anmerkung:** Soweit wir nichts anderes sagen, nehmen wir an, dass Zahlen binär kodiert sind.

# Warum ist Sortieren in P?

## Satz

*Sortieren*  $\in$  P.

### Beweis:

- Wir lösen das Problem beispielsweise mit Mergesort.
- Laufzeit im uniformen Kostenmaß:  $O(N \log N)$ .
- Laufzeit im logarithmischen Kostenmaß:  $O(\ell N \log N)$ , wobei  $\ell = \max_{1 \leq i \leq N} \log(a_i)$ .
- Sei  $n$  die Eingabelänge. Es gilt  $\ell \leq n$  und  $\log N \leq N \leq n$ .
- Somit ist die Laufzeit beschränkt durch  $\ell N \log N \leq n^3$ .



# Warum ist Graphzusammenhang in P?

## Problem (Graphzusammenhang)

**Eingabe:** Graph  $G = (V, E)$

**Frage:** Ist  $G$  zusammenhängend?

**Anmerkung:** Bei Graphproblemen gehen wir grundsätzlich davon aus, dass der Graph in Form einer Adjazenzmatrix eingegeben wird.

# Warum ist Graphzusammenhang in P?

## Satz

*Graphzusammenhang*  $\in P$ .

## Beweis

- Wir lösen das Problem mit einer Tiefensuche.
- Laufzeit im uniformen Kostenmaß:  $O(|V| + |E|)$
- Laufzeit im logarithmischen Kostenmaß:  
 $O((|V| + |E|) \cdot \log |V|)$
- Die Eingabelänge ist  $n = |V|^2 \geq |E|$ .
- Die Gesamtlaufzeit ist somit

$$O((|V| + |E|) \log |V|) = O(n \log n) = O(n^2) .$$





# Definition von NTM

## Definition (Nichtdeterministische Turingmaschine – NTM)

*Eine nichtdeterministische Turingmaschine (NTM) ist definiert wie eine deterministische Turingmaschine (TM), nur die Zustandsübergangsfunktion wird zu einer **Relation***

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

# Erläuterung der Rechnung einer NTM

- Eine Konfiguration  $K'$  ist direkter Nachfolger einer Konfiguration  $K$ , falls  $K'$  durch einen der in  $\delta$  beschriebenen Übergänge aus  $K$  hervorgeht.
- Rechenweg = Konfigurationsfolge, die mit Startkonfiguration beginnt und mit Nachfolgekonfigurationen fortgesetzt wird bis sie eine Endkonfiguration im Zustand  $\bar{q}$  erreicht.
- Beachte: Zu einer Konfiguration kann es mehrere direkte Nachfolgekonfigurationen geben. Der Verlauf der Rechnung ist also nicht eindeutig bestimmt (nicht deterministisch).

# Definition des Akzeptanzverhaltens

## Definition (Akzeptanzverhalten der NTM)

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

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

# Definition der Laufzeit

## Definition (Laufzeit der NTM)

*Sei  $M$  eine NTM. Die Laufzeit von  $M$  auf einer Eingabe  $x \in L(M)$  ist definiert als*

$$T_M(x) := \text{Länge des } \textcolor{red}{\text{kürzesten}} \text{ akzeptierenden Rechenweges von } M \text{ auf } x \text{ .}$$

*für  $x \notin L(M)$  definieren wir  $T_M(x) = 0$ .*

*Die worst case Laufzeit  $t_M(n)$  für  $M$  auf Eingaben der Länge  $n \in \mathbb{N}$  ist definiert durch*

$$t_M(n) := \max\{T_M(x) \mid x \in \Sigma^n\} \text{ .}$$

# Definition der Klasse NP

## Definition (Komplexitätsklasse NP)

NP ist die Klasse der Entscheidungsprobleme, die durch eine NTM  $M$  erkannt werden, deren worst case Laufzeit  $t_M(n)$  polynomiell beschränkt ist.

NP steht dabei für **nichtdeterministisch polynomiell**.

# Beispiel für ein Problem aus NP

## Problem (Cliquenproblem – CLIQUE)

**Eingabe:** Graph  $G = (V, E)$ ,  $k \in \{1, \dots, |V|\}$

**Frage:** Gibt es eine  $k$ -Clique?

- für das Cliquenproblem kennen wir keinen Polynomialzeitalgorithmus.
- Die besten bekannten Algorithmen haben eine exponentielle Laufzeit.

# Beispiel für ein Problem aus NP

## Satz

$CLIQUE \in NP$ .

**Beweis:** Wir beschreiben eine NTM  $M$  mit  $L(M) = CLIQUE$ :

- 1 Syntaktisch inkorrekte Eingaben werden verworfen.
- 2  $M$  „rt“ einen 0-1-String  $y$  der Länge  $|V|$ .
- 3 Sei  $C = \{i \in V \mid y_i = 1\} \subseteq V$ .
- 4  $M$  akzeptiert, falls  $C$  aus  $k$  Knoten besteht und jedes Knotenpaar aus  $C$  durch eine Kante verbunden ist.

*Korrektheit:* Es gibt genau dann einen akzeptierenden Rechenweg, wenn  $G$  eine  $k$ -Clique enthält.

*Laufzeit:* Alle Schritte haben polynomielle Laufzeit.



# Alternative Charakterisierung der Klasse NP

## Satz

*Eine Sprache  $L \subseteq \Sigma^*$  ist genau dann in NP, wenn es einen Polynomialzeitalgorithmus  $V$  (einen sogenannten Verifizierer) und ein Polynom  $p$  mit der folgenden Eigenschaft gibt:*

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



# Beweis: Von der NTM zu Zertifikat & Verifizierer

## Gegeben:

- Sei  $M$  eine NTM, die  $L \in NP$  in polynomieller Zeit erkennt.
- $M$ 's Laufzeit sei beschränkt durch ein Polynom  $q$ .
- O.B.d.A. sehe die Überführungsrelation von  $\delta$  immer genau zwei Übergänge vor, die wir mit 0 und 1 bezeichnen.

## Konstruktion von Zertifikat und Verifizierer:

- für die Eingabe  $x \in L$  beschreibe  $y \in \{0, 1\}^{q(n)}$  den Pfad von  $M$  auf einem akzeptierenden Rechenweg.
- Wir verwenden  $y$  als Zertifikat.
- Der Verifizierer  $V$  erhält als Eingabe  $y \# x$  und simuliert einen Rechenweg der NTM  $M$  für die Eingabe  $x$ .

# Beweis: Von der NTM zu Zertifikat & Verifizierer

## Korrektheit der Konstruktion:

- Gemäß Konstruktion gilt

$$\begin{aligned}x \in L &\Leftrightarrow M \text{ akzeptiert } x \\&\Leftrightarrow \exists y \in \{0,1\}^{q(n)} : V \text{ akzeptiert } y\#x.\end{aligned}$$

- Der Verifizierer kann die durch das Zertifikat  $y$  beschriebene Rechnung mit polynomielltem Zeitverlust simulieren.
- Somit erfüllen  $y$  und  $V$  die im Satz geforderten Eigenschaften.

# Beweis: Von Zertifikat & Verifizierer zur NTM

## Gegeben:

Verifizierer  $V$  mit polynomieller Laufzeitschranke und Polynom  $p$  mit der Eigenschaft:

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

## Konstruktion der NTM:

- 1  $M$  rät das Zertifikat  $y \in \{0,1\}^*, |y| \leq p(n)$ .
- 2  $M$  führt  $V$  auf  $y\#x$  aus und akzeptiert, falls  $V$  akzeptiert.

# Beweis: Von Zertifikat & Verifizierer zur NTM

## Korrektheit der Konstruktion:

- $M$  erkennt die Sprache  $L$ , weil gilt

$$\begin{aligned}x \in L &\Leftrightarrow \exists y \in \{0,1\}^*, |y| \leq p(n) : V \text{ akzeptiert } y\#x \\&\Leftrightarrow \text{Es gibt einen akzeptierenden Rechenweg für } M \\&\Leftrightarrow M \text{ akzeptiert } x.\end{aligned}$$

- Die Laufzeit von  $M$  ist polynomiell beschränkt, denn
  - die Laufzeit von Schritt 1 entspricht der Länge des Zertifikats, und
  - die Laufzeit von Schritt 2 entspricht der Laufzeit des Verifizierers.



# Optimierungsprobleme und ihre Entscheidungsvariante

Beim Rucksackproblem (KP) suchen wir eine Teilmenge  $K$  von  $N$  gegebenen Objekten mit Gewichten  $w_1, \dots, w_N$  und Nutzenwerten  $p_1, \dots, p_N$ , so dass die Objekte aus  $K$  in einen Rucksack mit Gewichtsschranke  $b$  passen und dabei der Nutzen maximiert wird.

## Problem (Rucksackproblem, Knapsack Problem – KP)

**Eingabe:**  $b \in \mathbb{N}$ ,  $w_1, \dots, w_N \in \{1, \dots, b\}$ ,  $p_1, \dots, p_N \in \mathbb{N}$

**zulässige Lösungen:**  $K \subseteq \{1, \dots, N\}$ , so dass  $\sum_{i \in K} w_i \leq b$

**Zielfunktion:** Maximiere  $\sum_{i \in K} p_i$

**Entscheidungsvariante:**  $p \in \mathbb{N}$  sei gegeben. Gibt es eine zulässige Lösung mit Nutzen mindestens  $p$ ?

# Optimierungsprobleme und ihre Entscheidungsvariante

Beim Bin Packing Problem suchen wir eine Verteilung von  $N$  Objekten mit Gewichten  $w_1, \dots, w_N$  auf eine möglichst kleine Anzahl von Behältern mit Gewichtskapazität jeweils  $b$ .

## Problem (Bin Packing Problem – BPP)

**Eingabe:**  $b \in \mathbb{N}$ ,  $w_1, \dots, w_N \in \{1, \dots, b\}$

**zulässige Lösungen:**  $k \in \mathbb{N}$  und Fkt  $f : \{1, \dots, N\} \rightarrow \{1, \dots, k\}$ ,

$$\text{so dass } \forall i \in \{1, \dots, k\} : \sum_{j \in f^{-1}(i)} w_j \leq b$$

**Zielfunktion:** Minimiere  $k$  (= Anzahl Behälter)

**Entscheidungsvariante:**  $k \in \mathbb{N}$  ist gegeben. Passen die Objekte in  $k$  Behälter?

# Optimierungsprobleme und ihre Entscheidungsvariante

Beim TSP ist ein vollständiger Graph aus  $N$  Knoten (Orten) mit Kantengewichten (Kosten) gegeben. Gesucht ist eine Rundreise (ein *Hamiltonkreis*, eine *Tour*) mit kleinstmöglichen Kosten.

## Problem (Traveling Salesperson Problem – TSP)

**Eingabe:**  $c(i, j) \in \mathbb{N}$  für  $i, j \in \{1, \dots, N\}$  mit  $c(j, i) = c(i, j)$

**zulässige Lösungen:** Permutationen  $\pi$  auf  $\{1, \dots, N\}$

**Zielfunktion:** Minimiere  $\sum_{i=1}^{N-1} c(\pi(i), \pi(i+1)) + c(\pi(N), \pi(1))$

**Entscheidungsvariante:**  $b \in \mathbb{N}$  ist gegeben. Gibt es eine Tour der Länge höchstens  $b$ ?

# Zertifikat & Verifizierer für Optimierungsprobleme

## Satz

*Die Entscheidungsvarianten von KP, BPP und TSP sind in NP.*

## Beweis:

Entscheidungsvarianten von Opt.problemen haben einen natürlichen Kandidaten für ein Zertifikat, nämlich **zulässige Lösungen**.

Es muss allerdings gezeigt werden, dass

- diese Lösungen eine polynomiell in der Eingabelänge beschränkte Kodierungslänge haben, und
- ihre Zulässigkeit durch einen Polynomialzeitalgorithmus überprüft werden kann.



# Zertifikat & Verifizierer für Optimierungsprobleme

- KP: Die Teilmenge  $K \subseteq \{1, \dots, N\}$  kann mit  $N$  Bits kodiert werden. Gegeben  $K$  kann die Einhaltung von Gewichts- und Nutzenwertschranke in polynomieller Zeit überprüft werden.
- BPP: Die Abbildung  $f : \{1, \dots, N\} \rightarrow \{1, \dots, k\}$  kann mit  $O(N \log k)$  Bits kodiert werden. Gegeben  $f$  kann die Einhaltung der Gewichtsschranken in polynomieller Zeit überprüft werden.
- TSP: für die Kodierung einer Permutation  $\pi$  werden  $O(N \log N)$  Bits benötigt. Es kann in polynomieller Zeit überprüft werden, ob die durch  $\pi$  beschriebene Rundreise die vorgegebene Kostenschranke  $b$  einhält.



# Optimierungsproblem versus Entscheidungsproblem

- Mit Hilfe eines Algorithmus, der ein Optimierungsproblem löst, kann man offensichtlich auch die Entscheidungsvariante lösen. (Wie?)
- Häufig funktioniert auch der umgekehrte Weg. Wir illustrieren dies am Beispiel von KP.
- In den Übungen zeigen wir dasselbe auch für TSP und BPP.

## Satz

*Wenn die Entscheidungsvariante von KP in polynomieller Zeit lösbar ist, dann auch die Optimierungsvariante.*

# Beweis: Entscheidungsvariante $\rightarrow$ Zwischenvariante

**Zwischenvariante:** Gesucht ist nicht eine optimale Lösung sondern nur der **optimale Zielfunktionswert**.

## Polynomialzeitalgorithmus B für die Zwischenvariante

Wir verwenden eine Binärsuche mit folgenden Parametern:

- Der minimale Profit ist 0. Der maximale Profit ist  $P := \sum_{i=1}^N p_i$ .
- Wir finden den optimalen Zielfunktionswert durch Binärsuche auf dem Wertebereich  $\{0, P\}$ .
- Sei  $A$  ein Polynomialzeitalgorithmus für die Entscheidungsvariante von  $KP$ .
- In jeder Iteration verwenden wir Algorithmus  $A$ , der uns sagt in welche Richtung wir weitersuchen müssen.

# Beweis: Entscheidungsvariante $\rightarrow$ Zwischenvariante

Die Anzahl der Iterationen der Binärsuche ist  $\lceil \log(P + 1) \rceil$ .

Diese Anzahl müssen wir in Beziehung zur Eingabelänge  $n$  setzen.

## Untere Schranke für die Eingabelänge:

- Die Kodierungslänge von  $a \in \mathbb{N}$  ist  $\kappa(a) := \lceil \log(a + 1) \rceil$ .
- Die Funktion  $\kappa$  ist subadditiv, d.h. für alle  $a, b \in \mathbb{N}$  gilt  $\kappa(a + b) \leq \kappa(a) + \kappa(b)$ .
- Die Eingabelänge  $n$  ist somit mindestens

$$\sum_{i=1}^N \kappa(p_i) \geq \kappa\left(\sum_{i=1}^N p_i\right) = \kappa(P) = \lceil \log(P + 1) \rceil .$$

Also reichen  $n$  Aufrufe von  $A$  um den optimalen Zielfunktionswert zu bestimmen.

# Beweis: Zwischenvariante $\rightarrow$ Optimierungsvariante

Aus einem Algorithmus  $B$  für die Zwischenvariante konstruieren wir jetzt einen Algorithmus  $C$  für die Optimierungsvariante.

## Algorithmus $C$

- ❶  $K := \{1, \dots, N\};$
- ❷  $p := B(K);$
- ❸ **for**  $i := 1$  **to**  $N$  **do**  
    **if**  $B(K \setminus \{i\}) = p$  **then**  $K := K - \{i\};$
- ❹ **Ausgabe**  $K.$

*Laufzeit:*  $N + 1$  Aufrufe von Algorithmus  $B$ , also polynomiell beschränkt, falls die Laufzeit von  $B$  polynomiell beschränkt ist.

# Die große offene Frage der Informatik lautet

$$P = NP?$$

Hierbei ist  $P$  natürlich auf Entscheidungsprobleme eingeschränkt. Das machen wir implizit immer dann, wenn  $P$  zu  $NP$  in Bezug gesetzt wird.

## Offensichtlich gilt

$$P \subseteq NP.$$

Klar: Denn eine (deterministische) TM ist eine spezielle NTM.

# Exponentielle Laufzeitschranke für Probleme aus NP

## Satz

*für jedes Entscheidungsproblem  $L \in \text{NP}$  gibt es einen Algorithmus  $A$ , der  $L$  entscheidet, und dessen worst case Laufzeit durch  $2^{q(n)}$  nach oben beschränkt ist, wobei  $q$  ein geeignetes Polynom ist.*



# Exponentielle Laufzeitschranke für Probleme aus NP

## Beweis:

Um die Eingabe  $x \in \{0, 1\}^n$  zu entscheiden,

- starte Verifiz.  $V$  mit  $y \# x$  für jedes Zertifikat  $y \in \{0, 1\}^{p(n)}$ ;
- akzeptiere, falls  $V$  eines der generierten Zertifikate akzeptiert.

## Laufzeitanalyse:

- Sei  $p'$  eine polynomielle Laufzeitschranke für  $V$ .
- Die Laufzeit unseres Algorithmus ist dann höchstens

$$\begin{aligned} 2^{p(n)} \cdot p'(p(n) + 1 + n) &\leq 2^{p(n)} \cdot 2^{p'(p(n)+1+n)} \\ &\leq 2^{p(n)+p'(p(n)+1+n)} = 2^{q(n)} . \end{aligned}$$

für das Polynom  $q(n) = p(n) + p'(p(n) + 1 + n)$ . □