Formale Sprachen und Komplexitätstheorie

WS 2019/20

Robert Elsässer

Inhaltsangabe:

- **Einleitung, Motivation**
- **Turing-Maschinen**
- **Arbeitstechniken**
- **Unentscheidbare Probleme**
- **Das Halteproblem**
- **Reduktionen**
- **Zeitkomplexität**
- **Die Klassen P und NP**
- **NP-vollständigkeit**
- **NP-vollständige Probleme**

Einführung

Berechenbakeit

Komplexität

Inhaltsangabe:

- **Einleitung, Motivation**
- **Turing-Maschinen**
- **Arbeitstechniken**
- **Unentscheidbare Probleme**
- **Das Halteproblem**
- **Reduktionen**
- **Zeitkomplexität**
- **Die Klassen P und NP**
- **NP-vollständigkeit**
- **NP-vollständige Probleme**

Einführung

Berechenbakeit

Komplexität

Wiederholung

Gibt es einen Algorithmus Halte, der

- als Eingabe einen beliebigen Algorithmus Alg und eine Eingabe w für Alg erhält und
- entscheidet, ob Alg bei Eingabe w hält?

Satz von Turing: Einen solchen Algorithmus kann es nicht geben.

Wiederholung

Alle sinnvollen Rechenmodelle liefen aus unserer Sicht die gleichen Ergebnisse (Churchsche These).

Es gibt Probleme, die algorithmisch nicht gelöst werden können (z.B. das Halteproblem) -> Berechenbarkeit

Manche Problem können zwar algorithmisch gelöst werden, aber sie können nicht effizient gelöst werden (z.B. das Problem des Handlungsreisenden) -> Komplexität

3. Komplexität

Von Berechenbarkeit zu Komplexität

• Berechenbarkeit in Bezug auf Entscheidbarkeit und Aufzählbarkeit betrachtet nur prinzipielle Lösbarkeit von Problemen mit Hilfe von Computern

• Prinzipiell lösbare Probleme können praktisch nicht lösbar sein, weil jeder Algorithmus zur Lösung des Problems zu viel Zeit (und/oder Platz) benötigt.

• Komplexitätstheorie versucht Probleme gemäß des Zeit- und Platzbedarfs des besten Algorithmus zu ihrer Lösung zu klassifizieren.

• Konzentrieren uns auf Zeitbedarf und die Klassen P und NP

3. Komplexität

Beispielprobleme:

Minimum-Suche:

- Eingabearray *A*[1..*n*].
- Ausgabe: Minimum in *A*[1..*n*].

Sortierproblem:

- Eingabe: Folge von n Zahlen $(a_1, \ldots, a_n)$
- Ausgabe: Umordnung der Zahlen in $(b_1, \ldots, b_n)$ so dass $b_1 \leq \ldots \leq b_n$

1. Einführung

Das Problem des Handlungsreisenden: Wien, Krems, St. Pölten, Wiener Neustadt, Mürzzuschlag, Graz, Wolfsberg, Klagenfurt, Villach, Spital, Lienz, Brenner, Innsbruck, Kitzbühel, Bischofshofen, Hallein, Salzburg, Braunau, Ried, Wels, Linz, Enns, Amstetten, Zwettl, Stockerau

O-Notation

Definition : **Sei $g : \mathbb{N} \to \mathbb{R}^+$ eine Funktion. Dann bezeichnen wir mit $O(g(n))$ die folgende Menge von Funktionen**

$$O(g(n)) := \left\{ f(n) : \begin{array}{l} \text{Es existieren Konstanten } c>0, n_0, \\ \text{so dass für alle } n \geq n_0 \text{ gilt} \\ 0 \leq f(n) \leq c\, g(n). \end{array} \right\}$$

- **$O(g(n))$ formalisiert: Die Funktion $f(n)$ wächst asymptotisch nicht schneller als $g(n)$.**
- **Statt $f(n)$ in $O(g(n))$ in der Regel $f(n) = O(g(n))$**

Ω-Notation

Definition : **Sei $g : \mathbb{N} \to \mathbb{R}^+$ eine Funktion. Dann bezeichnen wir mit $\Omega(g(n))$ die folgende Menge von Funktionen**

$$\Omega(g(n)) := \left\{ f(n) : \begin{array}{l} \text{Es existieren Konstanten } c>0, n_0, \\ \text{so dass für alle } n \geq n_0 \text{ gilt} \\ 0 \leq c\, g(n) \leq f(n). \end{array} \right\}$$

- **$\Omega(g(n))$ formalisiert: Die Funktion $f(n)$ wächst asymptotisch mindestens so schnell wie $g(n)$.**
- **Statt $f(n)$ in $\Omega(g(n))$ in der Regel $f(n) = \Omega(g(n))$**

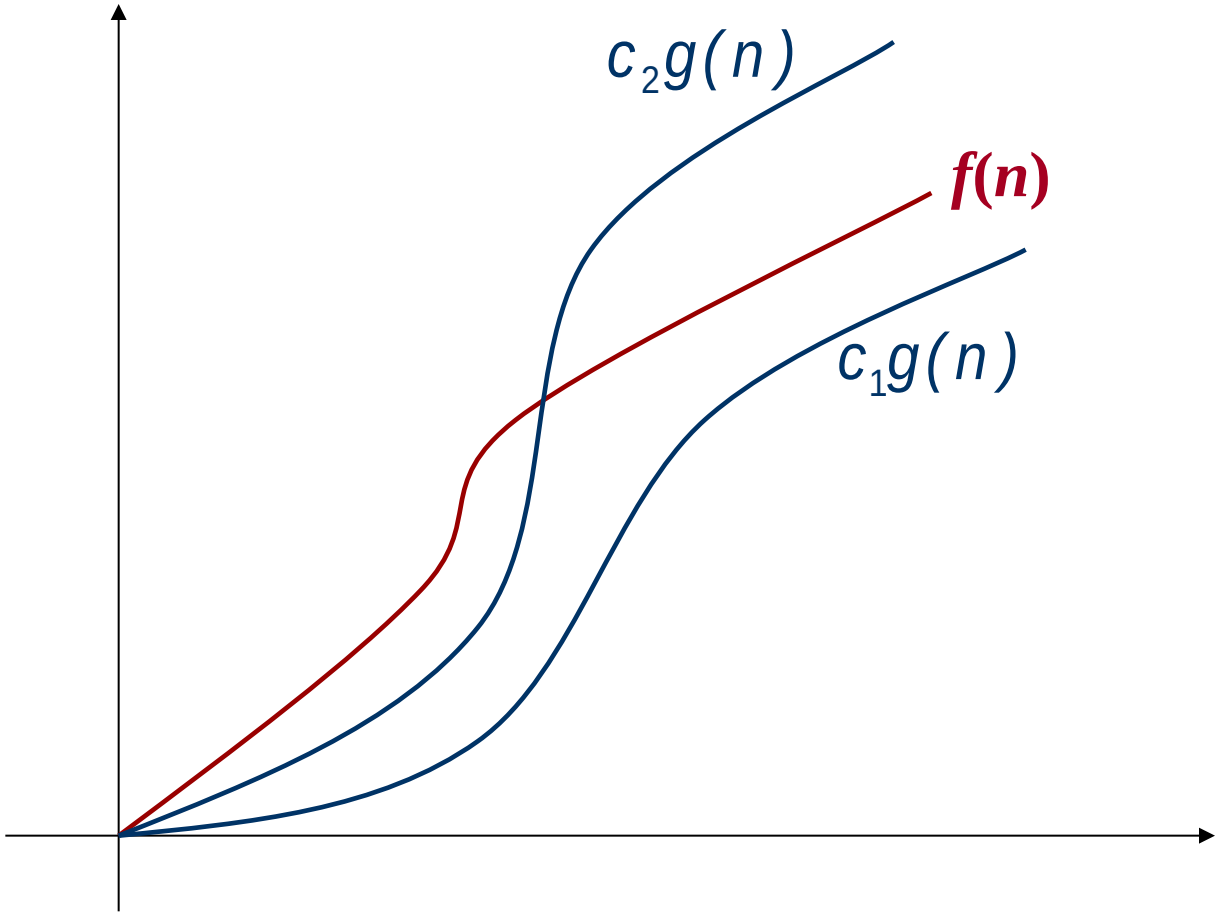
Θ-Notation

Definition : $g : \mathbb{N} \to \mathbb{R}^+$ **eine Funktion. Dann bezeichnen wir mit** $\Theta(g(n))$ **die folgende Menge von Funktionen**

$$\Theta(g(n)) := \left\{ f(n) : \begin{array}{c} \text{Es existieren Konstanten } c_1 > 0, c_2, n_0, \\ \text{so dass für alle } n \geq n_0 \text{ gilt} \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n). \end{array} \right\}$$

- **$\Theta(g(n))$ formalisiert: Die Funktion $f(n)$ wächst asymptotisch genau so schnell $g(n)$.**
- **Statt $f(n)$ in $\Theta(g(n))$ in der Regel $f(n) = \Theta(g(n))$**

Illustration von $\Theta(g(n))$



Effekt guter Algorithmen - Sortieren

Insertion - Sort :

Sortiert n Zahlen mit $c_1 n^2$ Vergleichen.

Computer A :
10^9 Vergleiche/Sek.
$c_1 = 2, n = 10^6.$

Dann benötigt A
$\frac{2 \cdot (10^6)^2}{10^9} = 2000$ Sek.

Merge - Sort :

Sortiert n Zahlen mit
$c_2 n \log(n)$ Vergleichen.

Computer B :
10^7 Vergleiche/Sek.
$c_2 = 50, n = 10^6.$

Dann benötigt B
$\frac{50 \cdot (10^6) \log(10^6)}{10^7} \approx 100$ Sek.

Regeln für Kalküle - Transitivität

O-, Ω- und Θ-Kalkül sind transitiv, d.h.:

- **Aus $f(n) = O(g(n))$ und $g(n) = O(h(n))$ folgt $f(n) = O(h(n))$.**
- **Aus $f(n) = \Omega(g(n))$ und $g(n) = \Omega(h(n))$ folgt $f(n) = \Omega(h(n))$.**
- **Aus $f(n) = \Theta(g(n))$ und $g(n) = \Theta(h(n))$ folgt $f(n) = \Theta(h(n))$.**

Regeln für Kalküle - Reflexivität

- **O-, Ω- und Θ-Kalkül sind reflexiv, d.h.:**

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

$$f(n) = \Theta(f(n))$$

- **Θ-Kalkül ist symmetrisch, d.h.**

$f(n) = \Theta(g(n))$ genau dann, wenn $g(n) = \Theta(f(n))$.

Regeln für Kalküle

Satz: Sei $f : \mathrm{N} \to \mathrm{R}^+$ mit $f(n) \geq 1$ für alle n. Weiter sei $k, l \geq 0$ mit $k \geq l$. Dann gilt

1. $f(n)^l = \mathrm{O}(f(n)^k)$.
2. $f(n)^k = \Omega(f(n)^l)$.

Satz: Seien $\varepsilon, k > 0$ beliebig. Dann gilt

1. $\log(n)^k = \mathrm{O}(n^\varepsilon)$.
2. $\mathrm{n}^\varepsilon = \Omega(\log(\mathrm{n})^\mathrm{k})$.

3. Komplexität

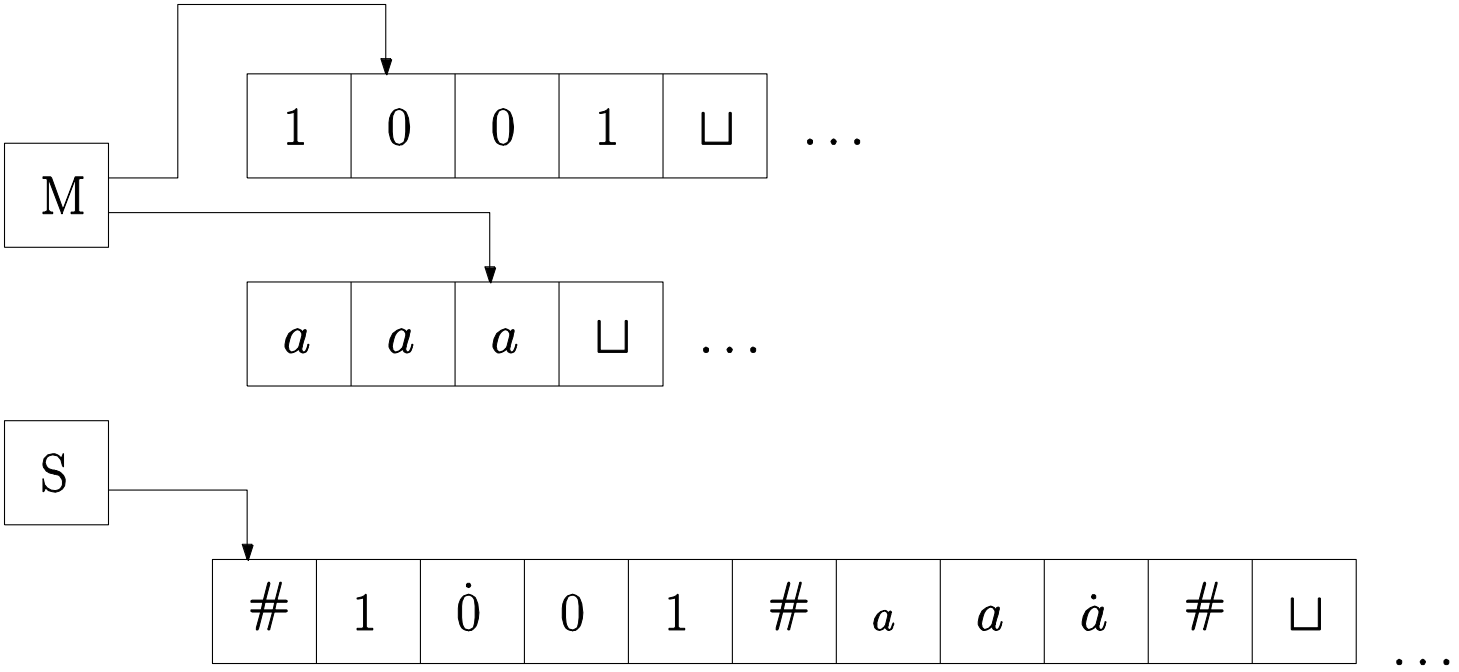
Laufzeit einer DTM:

Definition: DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{n-1}, q_n)$ halte bei jeder Eingabe.

- Für w aus Σ^* ist $T_M(w)$ die Anzahl der Rechenschritte von M bei Eingabe w.
- Für eine natürliche Zahl n ist $T_M(n) := \max \{T_M(w) \mid w \text{ aus } \Sigma^{\leq n}\}$.
- Die Funktion T_M heißt Zeitkomplexität oder Laufzeit der DTM M.
- M hat Laufzeit O(f(n)), wenn $T_M(n) = O(f(n))$.

Satz: Sei t eine monoton wachsende Funktion mit $t(n) \geq n$. Jede Mehrband DTM mit Laufzeit t(n) kann durch eine 1-Band DTM mit Laufzeit $O(t(n)^2)$ simuliert werden.

3. Komplexität



3. Komplexität

Die Klasse P:

Definition: Die Klasse P ist definiert als

$$\bigcup_k \mathrm{DTIME}(n^k).$$

P ist die Klasse der Sprachen, die durch die DTM mit polynomieller Laufzeit entschieden werden können.

Beispiel: $\{0^n1^n \mid n \geq 1\}$ ist in P.

3. Komplexität

Warum P?:

1. P ist eine mathematisch robuste Klasse.

2. Probleme in P lassen sich in der Praxis verhältnismäßig gut lösen, für Probleme außerhalb von P trifft dieses in der Regel nicht zu.

3. P liefert eine interessante Theorie.

3. Komplexität

Warum P?:

1. P ist eine mathematisch robuste Klasse.

2. Probleme in P lassen sich in der Praxis verhältnismäßig gut lösen, für Probleme außerhalb von P trifft dieses in der Regel nicht zu.

3. P liefert eine interessante Theorie.

3. Komplexität

Darstellung von Zahlen und Graphen:

1. Repräsentieren Zahlen in der Regel in Binärdarstellung.

    1. Dastellung bezüglich anderer Basen ist zulässig, da die Eingabegröße sich dadurch nur um einen konstanten Faktor ändert. Ist die Laufzeit für die Binärdarstellung in P, so trifft dies auch für jede andere Basis zu.

2. Repräsentiere Graphen durch

    1. Adjazenzmatrizen oder

    2. Adjazenzlisten

    Diese Darstellungen unterscheiden sich nur um einen quadratischen Faktor, so dass eine polynomielle Laufzeit bzgl. der einen Eingabeform auch polynomiell bzgl. der anderen ist.

3. Komplexität

Pfade in Graphen

Definition: Das Problem Pfad ist definiert wie folgt:

Pfad := {<G,s,t> | G=(V,E) ist ein gerichteter Graph mit s,t in V und einem gerichteten Pfad von s nach t.}

Satz: Pfad liegt in P.

3. Komplexität

Pfade in Graphen

M bei Eingabe <G,s,t>:

1. Markiere den Knoten s.

2. Wiederhole den folgenden Schritt bis keine zusätzlichen Knoten markiert werden.

3. Durchlaufe alle Kanten (a,b) von G. Ist a markiert und b nicht markiert, so markiere b.

4. Ist t markiert, akzeptiere, sonst lehne ab.

3. Komplexität

Teilerfremde Zahlen

Definition: Zwei natürliche Zahlen a,b heißen teilerfremd, wenn ihr größter gemeinsame Teiler (ggT) 1 ist.

RelPrim := {<x,y> | x und y sind teilerfremd.}

Satz: RelPrim liegt in P.

3. Komplexität

Teilerfremde Zahlen

E bei Eingabe <x,y>:

1. Wiederhole die folgenden Schritte bis y=0
    1. x ← x mod y
    2. Vertausche x und y.
2. Ausgabe x

R bei Eingabe <x,y>:

1. Simuliere E bei Eingabe <x,y>
2. Ist die Ausgabe von E 1, so akzeptiere, ansonsten lehne ab.