Grundlagen der Informatik

- Einführung in Berechenbarkeit und Komplexität -

Prof. Dr. Klaus Volbert

Hochschule für angewandte Wissenschaften
Fakultät Informatik und Mathematik

Wintersemester 2010/11
Regensburg, 18. Januar 2011

Entwicklung von Algorithmen

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Algorithmen stehen im Mittelpunkt der Informatik
- Hauptziel beim Entwurf von Algorithmen
  - Korrekte Problemlösung (totale Korrektheit)
    - Terminiertheit: Der Algorithmus endet für jede spezifizierte Eingabe
    - Partielle Korrektheit: Der Algorithmus liefert für jede spezifizierte Eingabe das geforderte Ergebnis
- Nebenziel beim Entwurf von Algorithmen
  - Effiziente Problemlösung hinsichtlich Zeit und Platz
- Interessant sind meist nur effiziente Algorithmen
- Wesentliche Effizienzmaße
  - Rechenzeitbedarf
    - Zeitkomplexität: Zählen der atomaren Schritte
  - Speicherplatzbedarf
    - Platzkomplexität: Zählen der verwendeten Speicherzellen

Übersicht Komplexitätsklassen

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

Menge	Laufzeit	Beispiele
$O(1)$	Konstant	Abfrage eines Wertes an einer bestimmten Stelle in einem Feld
$O(\log n)$	Logarithmisch	Binäre Suche in einem Feld (Halbierungsprinzip)
$O(n)$	Linear	Suche eines Wertes in einem unsortierten Feld, Fibonacci iterativ
$O(n \log n)$	Super-Linear	Sortieren mit guten Algorithmen (z.B. Mergesort)
$O(n^k)$	Quadratisch, Kubisch, … Allgemein: Polynomiell	Sortieren mit einfachen Algorithmen (z.B. Bubblesort)
$O(2^n)$	Exponentiell	Rekursive Variante zur Berechnung der Fibonacci-Folge, SAT
$O(n!)$	Faktoriell	Problem des Handlungsreisenden (Traveling Salesman Problem, TSP)

Weitere, elementare Komplexitätsklassen

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Klasse **P**
  - Bezeichnet die Klasse aller in höchstens polynomieller Zeit deterministisch lösbaren Probleme:

$$\mathrm{DTIME}\big(t(n)\big) = \{L \mid \exists \text{ determ. TM mit Laufzeit } O\big(t(n)\big), \text{die } L \text{ entscheidet}\}$$

$$P = \bigcup_{k \in \mathrm{IN}} \mathrm{DTIME}(n^k)$$

- Klasse **NP**
  - Bezeichnet die Klasse aller in höchstens polynomieller Zeit nicht-deterministisch lösbaren Probleme:

$$\mathrm{NTIME}\big(t(n)\big) = \{L \mid \exists \text{ nicht–determ. TM mit Laufzeit } O\big(t(n)\big), \text{die } L \text{ entscheidet}\}$$

$$NP = \bigcup_{k \in \mathrm{IN}} \mathrm{NTIME}(n^k)$$

- Nach Definition: P $\subseteq$ NP, aber NP $\subseteq$ P? Offen! (Millenium-Problem)

Anmerkungen Klasse P

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- P gilt als Klasse der
  - praktisch lösbaren Probleme
  - wirklich/in der Realität lösbaren Probleme
  - in akzeptabler Zeit lösbaren Probleme

  Zeit- und Speicheraufwand wächst akzeptabel mit der Problemgröße (und zwar maximal wie ein Polynom)
- Typische Beispiele
  - Sequentielle Suche $O(n)$
  - Sortieralgorithmen $O(n \log\ n)$
  - Primzahlensieb nach Eratosthenes $O(n^2)$
  - Klassische Matrizenmultiplikation $O(n^3)$
  - Schnelle Fouriertransformation $O(n \log\ n)$
- Praktisch unlösbare Probleme:
  - Zeit-/Speicheraufwand wächst echt stärker als polynomiell (z.B. $O(2^n)$)

Anmerkungen Klasse NP

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Als praktisch unlösbar gelten die Probleme, die zur Lösung mindestens exponentiellen Aufwand (Zeit, Platz) erfordern
- Vorsicht: NP steht nicht für nicht polynomiell
- NP gilt als eine spezielle Teilmenge der praktisch unlösbaren Probleme
- Es ist unbeantwortet, ob manche Probleme in NP zur Lösung mindestens exponentiellen Aufwand erfordern
  - Beispiele: Ganzzahliges Rucksackproblem, SAT
- Probleme in NP können zweistufig gelöst werden:

1. Rate eine Lösung (meist $O(n)$)	nicht-deterministisch
2. Verifiziere, ob die Lösung stimmt $O(n^k)$	deterministisch

- Lösungen deterministisch zu finden ist „sehr schwer“

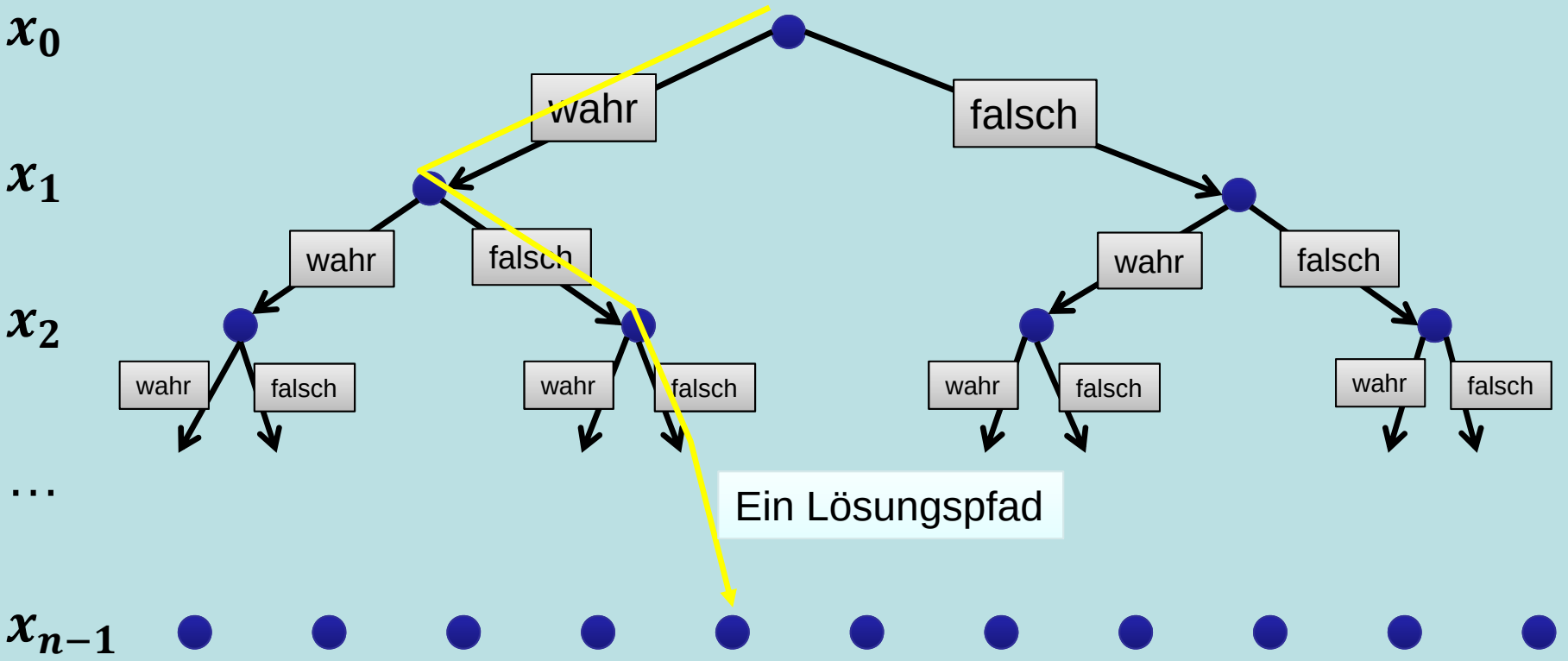
Erfüllbarkeitsproblem (engl. satisfiability)

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

$$\text{SAT} = \{K \mid K \text{ ist erfüllbarer boolescher Ausdruck in KNF}\}$$

- Eingabe: Boolescher Ausdruck K in KNF über $\boldsymbol{x_0, \ldots, x_{n-1}}$
- Ausgabe: K erfüllbar (K ∈ SAT) oder K nicht erfüllbar (K ∉ SAT)
- Naiver deterministischer Algorithmus:
  - Überprüfe alle möglichen Belegungen für $\boldsymbol{x_0, \ldots, x_{n-1}}$
- Zeitkomplexität? Gilt SAT ∈ P?
  - Wie viele mögliche Belegungen gibt es? 2^n viele
  - Zeitkomplexität ist $O(2^n)$ (exponentiell in der Eingabelänge)
  - Schnellere Algorithmen sind nicht bekannt, d.h. „SAT ∈ P?“ ist offen
- Nichtdeterministischer Algorithmus, SAT ∈ NP:
  - Rate eine Belegung $O(n)$
  - Überprüfe, ob die Belegung wahr ist $O(n^2)$

Berechnungsbaum für SAT



- Wie tief/hoch ist der Baum? $O(n)$
- Wie lang ist der Lösungspfad? $O(n)$
- Wie viele Pfade/Blätter gibt es? $O(2^n)$

Satz von Cook-Levin

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Satz: Jedes Problem in NP lässt sich in Polynomialzeit auf das SAT-Problem zurückführen (*SAT ist NP-vollständig*)
- Eine Sprache L heißt NP-schwer, wenn sich jede Sprache L' aus NP in deterministisch polynomieller Zeit auf L zurückführen lässt. Schreibweise: $L' \leq_p L$
- Eine Sprache L heißt NP-vollständig, wenn L NP-schwer ist und L in NP liegt.
- Interpretationen
  - Ein NP-schweres Problem ist mindestens so schwer wie das schwerste Problem in NP
  - Ein NP-vollständiges Problem ist NP-schwer und liegt selbst in NP, womit es zu den schwersten Problemen in NP gehört
- Folgerung: SAT gehört zu den schwersten Problemen in NP

Anmerkungen zur NP-Vollständigkeit

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Sei L NP-vollständig, dann gilt:

$$L \in \mathrm{P} \Leftrightarrow \mathrm{NP} \subseteq \mathrm{P}$$

- Beweis:

"$\Rightarrow$"

Sei $L' \in \mathrm{NP}$ beliebig. Da L NP-vollständig ist, ist L NP-schwer, d.h. jedes Problem in NP lässt sich in deterministisch polynomieller Zeit auf L zurückführen, insbesondere L'. Da L in P liegt, gibt es eine deterministische TM, die L in polynomieller Zeit entscheidet. Mittels dieser TM und der Zurückführung folgt, dass $L' \in \mathrm{P}$

"$\Leftarrow$"

Da L NP-vollständig ist, gilt $L \in \mathrm{NP}$, und wegen $\mathrm{NP} \subseteq \mathrm{P}$ auch $L \in \mathrm{P}$

Wesentliche Folgerungen

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Alle NP-vollständigen Probleme sind gleich schwierig
- Für den Nachweis, dass NP=P ist, genügt es, für ein NP-vollständiges Problem einen deterministisch polynomiellen Algorithmus anzugeben
- Um zu zeigen, dass ein Problem p ∈ NP NP-vollständig ist, genügt es, ein anderes NP-vollständiges Problem in polynomieller Zeit auf p zu reduzieren
- NP-vollständige Probleme sind nicht in P, wenn NP≠P
- Da bisher kein deterministisch polynomieller Algorithmus für ein NP-vollständiges Problem gefunden wurde, geht man davon aus, dass folgendes gilt:

$$NP \neq P$$

(Nachweis offen!)

Beispiel: Dreifärbbarkeit

$$\text{3-COLOR} = \{G = (V, E) \mid G \text{ ist ungerichteter Graph und } \exists\, c: V \to \{1,2,3\} \text{ mit } \forall\, (u, v) \in E \text{ gilt } c(u) \neq c(v)\}$$

- Eingabe: Ungerichteter Graph G mit $n \geq 4$ Knoten
- Ausgabe: G ist 3-färbbar oder G ist nicht 3-färbbar
- Naiver deterministischer Algorithmus:
  - Überprüfe alle möglichen Farben für die Knoten (exponentiell)
- Nichtdeterministischer Algorithmus, 3-COLOR ∈ NP:
  - Rate eine Farbe für jeden Knoten $O(n)$
  - Überprüfe, ob die Färbungsbedingung wahr ist $O(n^2)$
- Gleiches Problem wie bei SAT:
  - Algorithmus mit polynomieller Laufzeit ist unbekannt
- Wir zeigen:
  - 3-COLOR lässt sich auf SAT polyn. zurückführen (3-COLOR $\leq_p$ SAT)

3-COLOR $\leq_p$ SAT

- Wir drücken die Bedingungen für eine Dreifärbbarkeit in Form eines booleschen Ausdrucks in KNF aus:
  - Boolesche Variable:
    - $r_i = 1 \Leftrightarrow$ Knoten i ist rot
    - $g_i = 1 \Leftrightarrow$ Knoten i ist gelb
    - $b_i = 1 \Leftrightarrow$ Knoten i ist blau
  - Jeder Knoten hat mindestens eine Farbe:
    - $\alpha_1 = \Lambda_{i=1}^{n}(r_i + g_i + b_i)$ $O(n)$
  - Kein Knoten hat zwei Farben:
    - $\alpha_2 = \Lambda_{i=1}^{n}\left((\overline{r_i} + \overline{g_i})(\overline{g_i} + \overline{b_i})(\overline{r_i} + \overline{b_i})\right)$ $O(n)$
  - Knoten an einer Kante haben verschiedene Farben
    - $\alpha_3 = \Lambda_{(v_i,v_j)\in E}\left((\overline{r_i} + \overline{r_j})(\overline{g_i} + \overline{g_j})(\overline{b_i} + \overline{b_j})\right)$ $O(n^2)$
  - Der zu überprüfende boolesche Ausdruck ist nun: $K = \alpha_1\alpha_2\alpha_3$
- Folgerung: 3-COLOR ist ein Spezialfall von SAT

Weitere Anmerkungen

- Hierarchie der NP-vollständigen Probleme

  SAT $\leq_p$ 3-SAT $\leq_p$ 3-COLOR

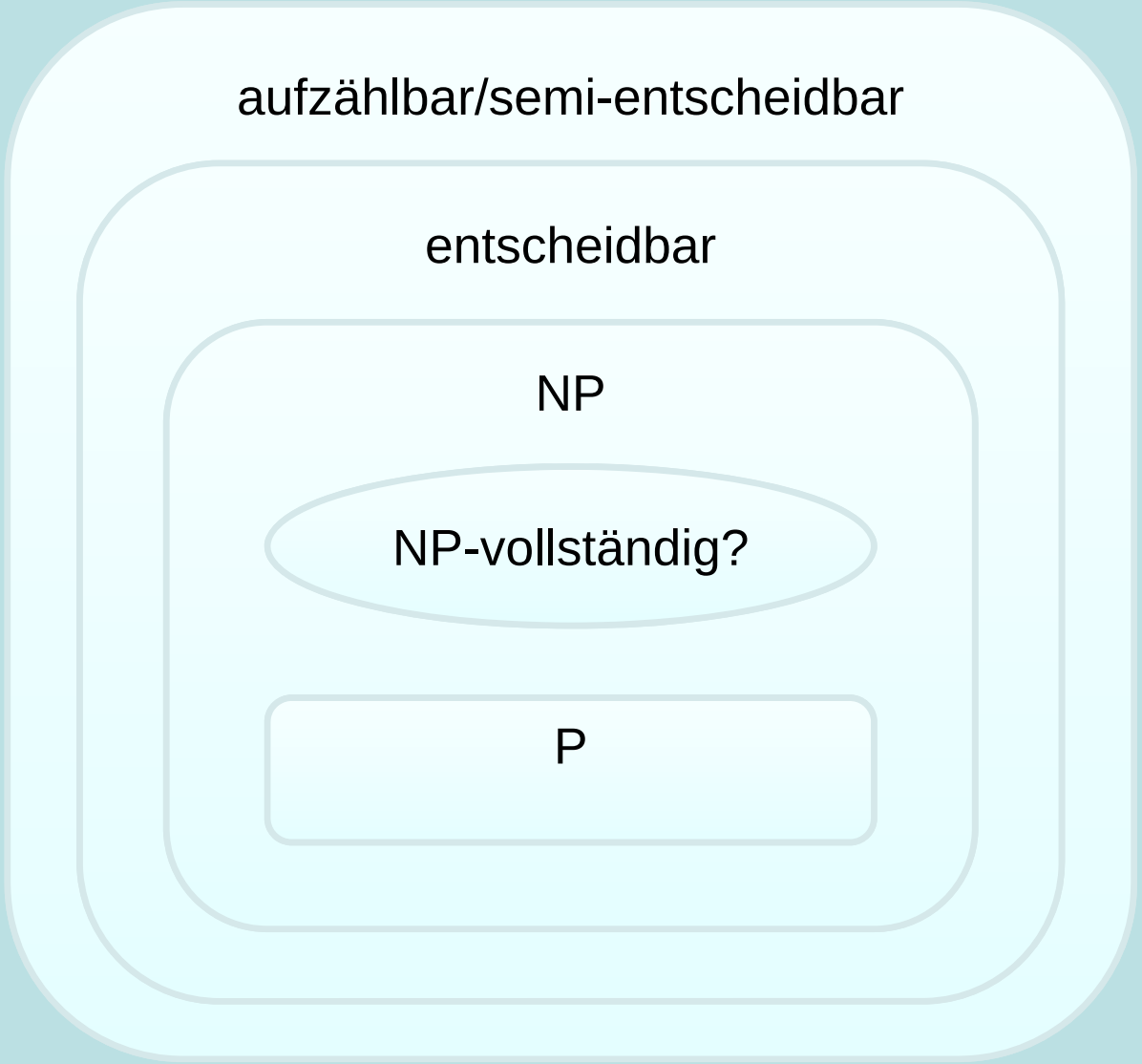
  $\leq_p$ CLIQUE

  $\leq_p$ RUCKSACK $\leq_p$ SUBSUM (Ganzz. Rucksack)

  $\leq_p$ HAMILTON $\leq_p$ TSP

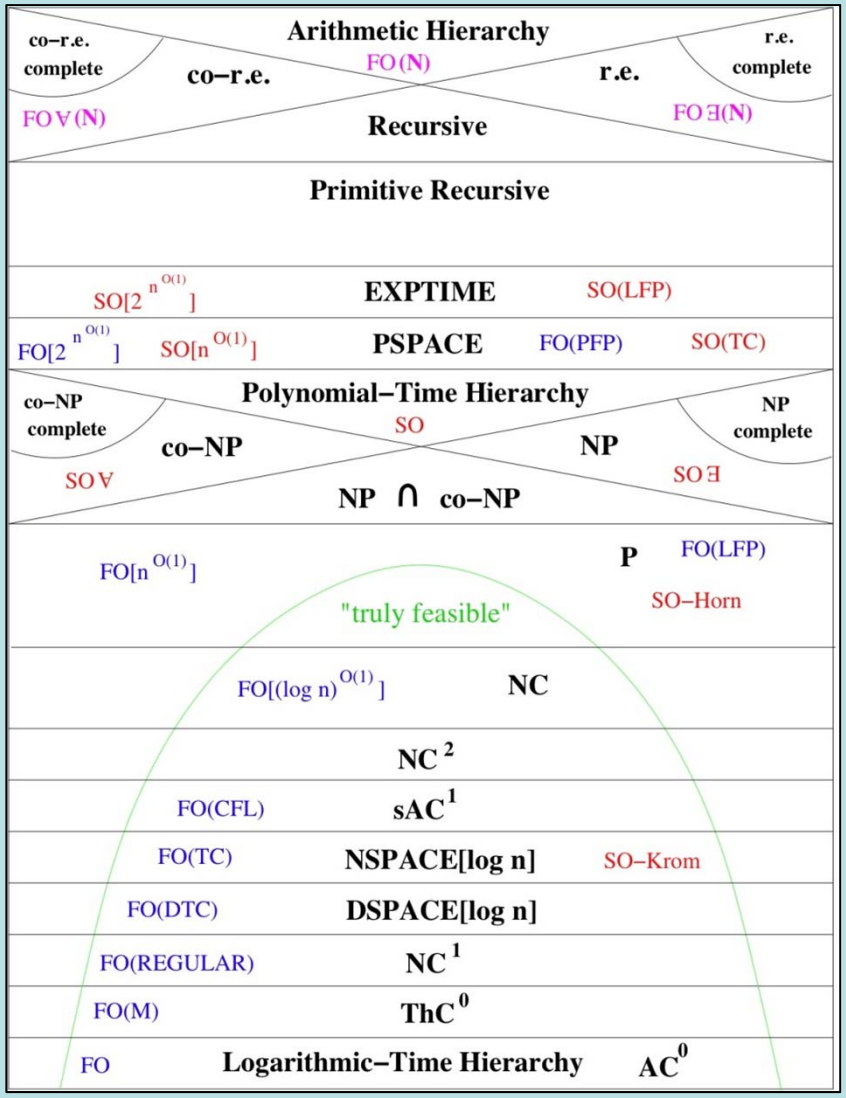
- Heute sind $\geq$ 1.000 NP-vollständige Probleme bekannt
- NP-vollständige Rätselspiele: Sudoku, Tetris, Minesweeper
- NP-vollständig = hoffnungslos schwierig? Nicht ganz:
  - Effiziente Ermittlung von akzeptablen Lösungen für NP-vollständige Probleme mittels Approximationsalgorithmen oder Heuristiken
- Es gibt sogar Probleme, die nachweisbar nicht effizient und nicht qualitativ hochwertig approximiert werden können

Übersicht der Komplexitätstheorie



Ausblick der Komplexitätstheorie

- …es gibt noch viele weitere Komplexitätsklassen:



Inhaltlicher Rückblick

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

- Grundlagen der Informatik
  - Allgemeine Einführung (Was ist Informatik?, Geschichte, Begriffe)
  - Teilgebiet 1:Einführung in die Technische/Praktische Informatik
    - Hardware (Schaltungen, Komponenten, Von-Neumann-Rechner, RAM)
    - Software (Vom Programm zum Maschinenprogramm, Programmieren im Kleinen, Programmieren im Großen)
  - Teilgebiet 2: Einführung in die Theoretische Informatik
    - Berechenbarkeitstheorie (Berechenbarkeit, Turingmaschinen, Halteproblem, Church‘sche These)
    - Komplexitätstheorie (Polynomielle Algorithmen, Nichtdeterminismus, Klassen P und NP, NP-Vollständigkeit)

Viel Erfolg bei den anstehenden Prüfungen!