

## VL-00: Einführung

(Berechenbarkeit und Komplexität, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

# Organisatorisches

Planung der nächsten paar Wochen:

Vorlesung: Freitag, Oktober 12, 16:30–18:00, Audimax

Vorlesung: Donnerstag, Oktober 25, 12:30–14:00, Aula

Vorlesung: Freitag, Oktober 26, 16:30–18:00, Audimax

Vorlesung: Freitag, November 2, 16:30–18:00, Audimax

Vorlesung: Donnerstag, November 15, 12:30–14:00, Aula

Achtung:

Keine Vorlesung am Donnerstag, Oktober 18

Keine Vorlesung am Freitag, Oktober 19

Keine Vorlesung am Donnerstag, November 1 (Allerheiligen)

Keine Vorlesung am Donnerstag, November 8

Keine Vorlesung am Freitag, November 9

Webseite:

<http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php>

# **Organisatorisches**

- Personen
- Termine
- Übungen und Tutorium
- Klausur
- Andere Studiengänge
- Materialien

- Vorlesung: Gerhard Woeginger (Zimmer 4024 im E1)  
Sprechstunde: Freitag 11:15–12:00
- Übungen: Jan Böker, Tim Hartmann  
Email: [buk@lists.rwth-aachen.de](mailto:buk@lists.rwth-aachen.de)



Jan Böker



Tim Hartmann

- Tutorium: Viele Tutoren

# Termine

Die Vorlesung ist dreistündig (3V+2Ü), wird aber in zwei 90-minütigen Blöcken abgehalten, die nicht jede Woche stattfinden.

- Vorlesungszeiten (ab heute):  
Donnerstag, 12:30–14:00, Aula  
Freitag, 16:30–18:00, Audimax
- Globalübung (ab Oktober 31):  
Mittwoch, 12:30–14:00, AH IV
- Tutorium (ab der Woche Oktober 22)  
Zu vielen verschiedenen Zeiten an vielen verschiedenen Orten

# Übungen (1): Anmeldung

Sie müssen Sich zu den Übungen anmelden

- falls Sie an den Übungen teilnehmen wollen
- falls Sie einem Tutorium zugewiesen werden wollen

Zur Anmeldung:

- <https://lufgi2.informatik.rwth-aachen.de/buk18/>  
(Dieser Link ist auch auf unserer Webseite verfügbar)
- Alle Details zur Anmeldung: Übungsblatt 0  
(Dieses Übungsblatt ist auf unserer Webseite verfügbar)
- **Achtung: Nur bis Dienstag, Oktober 16, 23:55 möglich (!!!)**

Webseite:

<http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php>

## Übungen (2): Details

- Die Übungsbätter erscheinen wöchentlich im L2P System  
(das erste Blatt erscheint in der nächsten Woche)
- Die Übungen werden in Gruppen zu 2 Studierenden bearbeitet
- Abgabeschluss ist jeweils am Mittwoch, um 12:15
- Die Ausarbeitungen sind mit Übungsgruppe, Namen und Matrikelnummern zu beschriften
- Blätter zusammenheften!
- Webseite:  
<http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php>

# Übungen (3): Mehr Details

Abgabe:

jeweils am Mittwoch, um 12:15, in den Kasten vor dem Lehrstuhl i1



## Das Tutorium

- wird in 26 Kleingruppen abgehalten
- bearbeitet Extra-Aufgaben, die auf die Hausaufgaben vorbereiten

## Verteilung auf Kleingruppen:

- Die Verteilung auf Kleingruppen erfolgt übers Übungssystem
- Präferenzen können nur im Übungssystem formuliert werden

## Webseite:

<http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php>

# Klausur (1)

Zulassungsvoraussetzung für die Klausur:

- Um an der Klausur teilzunehmen, müssen Sie 50% der Punkte in den Hausaufgaben erreichen
- Zulassungen aus früheren Studienjahren: sind verfallen

Bonusregelung:

- Die Bonusregel betrifft nur Studierende, die mindestens 70% der Punkte in den Hausaufgaben erreichen
- Für diese Studierenden verbessert sich die Note einer **bestandenen** Klausur um einen Grad (= 0.3)

# Klausur (2)

## Klausur:

- Die Bearbeitungszeit beträgt 120 Minuten
- Erster Termin: Februar 19, 2019
- Wiederholung: März 6, 2019

## Anmeldung

- Für die Klausur müssen Sie Sich beim ZPA anmelden
- **Anmeldefrist** beachten!

# Andere Studiengänge ( $\neq$ Bachelor-Informatik)

- Um einen benoteten Schein zu erhalten (und das ist der Normalfall!) nehmen Sie genau wie die Bachelor-Informatik-Studierenden an der Klausur teil. Dazu müssen Sie auch (wie beschrieben) die Zulassung erlangen.
- Für einen unbenoteten Teilnahmeschein müssen Sie an den Übungen teilnehmen und die Kriterien für die Klausurzulassung erfüllen. In diesem Fall brauchen Sie aber nicht an der Klausur teilzunehmen.
- Studierende, die die Vorlesung als Master-Auflage absolvieren müssen, nehmen ganz normal an den Übungen und an der Klausur teil. Diese Studierenden müssen sich aber **direkt bei uns anmelden**, und zwar per E-Mail an

`buk@lists.rwth-aachen.de`

mit Name und Matrikelnummer, und den Worten “Master-Auflage” im Subject.

# Materialien (1)

- In der Vorlesung wird der Stoff per Beamerpräsentation vermittelt, manchmal auch an der Tafel oder auf dem Overheadprojektor
- Vor der Vorlesung wird ein Foliensatz auf der BuK-Webseite zur Verfügung gestellt. (Die Folien basieren auf Material, das im Laufe der Jahre von Berthold Vöcking, Wolfgang Thomas, Martin Grohe und Pascal Schweitzer entwickelt wurde.)
- Ausserdem gibt es auf der BuK-Webseite das Vorlesungsskript von Berthold Vöcking aus dem Wintersemester 2014 (mit 139 Seiten)
- Keine Video Lectures

Webseite:

<http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php>

## Materialien (2): Buchempfehlungen

Die folgenden Bücher zum Thema sind in der Informatikbibliothek zu finden:

- Uwe Schöning. [Theoretische Informatik - kurzgefasst.](#)  
Spektrum Akademischer Verlag, 2001.
- Michael Sipser. [Introduction to the Theory of Computation.](#)  
Cengage Learning, 2012.

Ein weiterführendes Buch ist

- Sanjeev Arora, Boaz Barak. [Computational Complexity.](#)  
Cambridge University Press, 2009.

# Was tun bei Fragen?

- Erster Ansprechpartner ist immer der Tutor / die Tutorin der Kleingruppe.
- Sollte das Tutorium die Frage nicht klären können, so können Sie mich nach der Vorlesung ansprechen, oder Jan Böker oder Tim Hartmann in der Globalübung fragen.
- Dann erst, oder in dringenden(!) Fällen E-Mail an:  
`buk@lists.rwth-aachen.de`

# Wie können Sie mich erreichen?

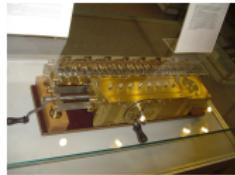
- Immer nach der Vorlesung.
- In meiner Sprechstunde: Freitag 11:15–12:00
- Zur Not einen anderen Termin mit Frau Schlebusch (Sekretariat i1) vereinbaren.

## Nun zum Vorlesungsstoff

- Berechenbarkeit
- Komplexität

# **Berechenbarkeit**

# Rechenmaschinen



1672/1700

User:Kolossos/Wikimedia Commons/CC-BY-SA-3.0



1923

Greg Goebel/Wikimedia Commons/Public Domain



?



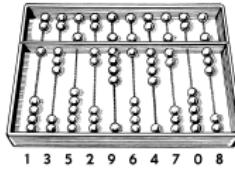
1983



1980



2013



-3000

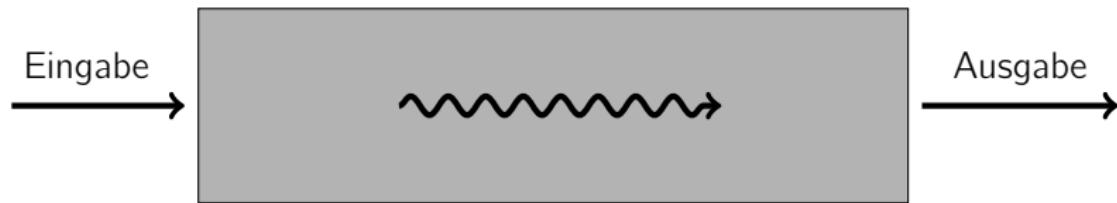
Pearson Scott Foresman/Wikimedia Commons/Public Domain



~2014

# Die absoluten Grenzen des Computers (1)

In der Vorlesung interessieren wir uns allgemein für Berechnungen und Berechnungsprobleme.



# Die absoluten Grenzen des Computers (2)

## Zentrale Frage

Gibt es überhaupt Probleme,  
die wir nicht mit dem Computer lösen können?

## Bessere Formulierung

Gibt es “algorithmische Probleme” (oder “Berechnungsprobleme”),  
die ein Computer nicht lösen kann?

# David Hilbert (1862–1943)

Wikipedia:

David Hilbert war ein deutscher Mathematiker. Er gilt als einer der bedeutendsten Mathematiker der Neuzeit. Viele seiner Arbeiten auf dem Gebiet der Mathematik und mathematischen Physik begründeten eigenständige Forschungsgebiete.

Hilbert begründete die moderne formalistische Auffassung von den Grundlagen der Mathematik und veranlasste eine kritische Analyse der mathematischen Begriffsdefinitionen und des mathematischen Beweises.



# Die absoluten Grenzen des Computers (3)

Zurück zu unserer zentralen Frage

Gibt es “algorithmische Probleme” (oder “Berechnungsprobleme”), die ein Computer nicht lösen kann?

Was dachte David Hilbert zu dieser Frage?

YouTube Video:

[https://www.youtube.com/watch?v=EBgAu\\_X2mm4](https://www.youtube.com/watch?v=EBgAu_X2mm4)

# Die absoluten Grenzen des Computers (4)

David Hilbert: Radioansprache 1930

In der Tat: Wir beherrschen nicht eher eine naturwissenschaftliche Theorie, als bis wir ihren mathematischen Kern herausgeschält und völlig enthüllt haben. Ohne Mathematik ist die heutige Astronomie und Physik unmöglich.

[...]

Wir dürfen nicht denen glauben, die heute mit philosophischer Miene und überlegenem Tone den Kulturuntergang prophezeien und sich in dem Ignorabimus gefallen. Für uns gibt es kein Ignorabimus, und meiner Meinung nach auch für die Naturwissenschaft überhaupt nicht.  
Statt des törichten Ignorabimus heisse im Gegenteil unsere Lösung:

Wir *müssen* wissen.

Wir *werden* wissen.

# Die absoluten Grenzen des Computers (5)

Wir werden aber sehen:

- Es gibt keinen Algorithmus, der entscheidet, ob ein gegebenes Programm in einen bestimmten Zustand läuft.  
(Error: 0E : 016F : BFF9B3D4.)

**Windows**

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this,  
you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

Press any key to continue \_

# Die absoluten Grenzen des Computers (6)

Wir werden sehen:

- Es gibt keinen Algorithmus, der entscheidet, ob ein gegebenes Programm in einen bestimmten Zustand läuft.  
**(Error: 0E : 016F : BFF9B3D4.)**
- Allgemein lässt sich die Funktionsweise von Programmen nur schwer algorithmisch überprüfen.
- Zum Beispiel gibt es keinen Algorithmus, der entscheidet, ob ein gegebenes Programm immer die Summe zweier eingegebenen Zahlen berechnet.

## Allgemeines Halteproblem

- **Eingabe:** Ein Programm in einer wohldefinierten, universellen Programmiersprache (z.B. Java, C++, Python, Haskell).
- **Frage:** Terminiert dieses Programm?

Wir werden beweisen, dass es keinen Algorithmus gibt, der dieses Problem entscheiden kann.

# **Komplexität**

# Die Grenzen der effizienten Berechenbarkeit

Zentrale Frage:

Lässt sich ein gegebenes algorithmisches Problem **effizient** lösen?

Mit “**effizient**” meinen wir:

- in vernünftiger Laufzeit,
- mit vernünftigem Speicherbedarf
- und unter vernünftiger Verwendung anderer Ressourcen?

# Beispiel 1: Zauber-Dodekaeder



Aufgabe

Löse den Dodekaeder

Lösung: Probiere alle Möglichkeiten durch

Ist das eine effiziente Lösung?

# Beispiel 2: Rush Hour



User:Welt-der-Form/Wikimedia Commons/CC-BY-SA-3.0

## Aufgabe

Befreie das rote Auto aus dem Verkehrsstau

Lösung lautet wieder: Probiere alle Möglichkeiten durch

Nachteil lautet wieder: Extrem langsamer Algorithmus

## Beispiel 3: Passwort



### Aufgabe

Bestimme das alpha-numerische  
15-stellige Passwort

Lösung lautet wieder: Probiere  
alle Möglichkeiten durch

Nachteil lautet wieder: Viel zu  
langsam!

# Beispiel 4: Traveling Salesman (1)



## Aufgabe

Finde eine kurze Rundreise durch die 14 grössten deutschen Städte und zurück zum Ausgangsort.

Die angegebene Route ist die kürzeste unter 43.589.145.600 möglichen.

Auch dieses Problem kann dadurch gelöst werden, dass man einfach alle Möglichkeiten durchprobiert.

Und auch dieser Algorithmus ist extrem ineffizient. (Wie lange braucht dieser Algorithmus wohl für 40 Städte?)

# Beispiel 4: Traveling Salesman (2)

## Traveling Salesman Problem (TSP)

- Eingabe: vollständiger Graph  $G$  mit allen Kantenlängen
- Ausgabe: eine Rundreise, die alle Knoten in  $G$  besucht und dabei so kurz wie möglich ist

In der Vorlesung werden wir sehen,  
dass es unter einer gewissen Hypothese (" $P \neq NP$ ")  
**keinen effizienten** Algorithmus für das TSP gibt.

## Aus der Wikipedia

Prolog is a general-purpose logic programming language associated with artificial intelligence and computational linguistics.

In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations. Relations and queries are constructed using Prolog's single data type, the term. Relations are defined by clauses.

Given a query, the Prolog engine attempts to find a resolution refutation of the negated query. If the negated query can be refuted, i.e., an instantiation for all free variables is found that makes the union of clauses and the singleton set consisting of the negated query false, it follows that the original query, with the found instantiation applied, is a logical consequence of the program. This makes Prolog (and other logic programming languages) particularly useful for database, symbolic mathematics, and language parsing applications.

# **Übersicht (Inhalt)**

# Übersicht (1)

## Teil 1: Grundlagen

- Modellierung von Problemen
- Einführung der Turingmaschine (TM)
- Einführung der Registermaschine (RAM)
- Vergleich TM versus RAM
- Church-Turing-These

## Teil 2: Berechenbarkeit

- Existenz unentscheidbarer Probleme
- Unentscheidbarkeit des Halteproblems
- Diagonalisierung / Unterprogrammtechnik / Reduktion
- Das Post'sche Korrespondenzproblem
- Hilberts zehntes Problem
- WHILE- und LOOP-Programme
- Primitiv rekursive Funktionen

## Teil 3: Komplexität

- Die Komplexitätsklassen P und NP
- NP-Vollständigkeit und der Satz von Cook und Levin
- Kochrezept für NP-Vollständigkeitsbeweise  
(Polynomielle Reduktion)
- NP-Vollständigkeit zahlreicher Probleme
- Weitere Komplexitätsklassen: coNP, PSPACE, EXPTIME