Grenzen der Berechenbarkeit

M. Jakob

Gymnasium Pegnitz

24. Februar 2019

Inhaltsverzeichnis

Laufzeitanalyse
- Alltagsbeispiele
- Experimentelle Laufzeitanalyse
- Laufzeitabschätzung ausgewählter Algorithmen

Ausblick in die Kryptographie
- Historische Kryptographie
- Moderne Kryptographie

Das Halteproblem

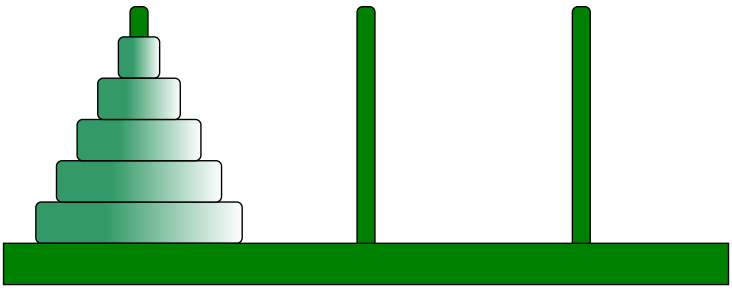
In diesem Abschnitt

Laufzeitanalyse
Alltagsbeispiele
Experimentelle Laufzeitanalyse
Laufzeitabschätzung ausgewählter Algorithmen

Beispiel: Turm von Hanoi

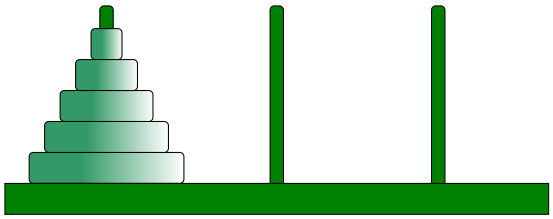
Auf einem Brett mit drei Stäbe befindet sich auf Stab A ein Turm mit n unterschiedlich große Scheiben. Der Turm soll nach folgenden Regeln auf Stab C umgesetzt werden.

- Die Scheiben dürfen nur auf den drei Stäben abgelegt werden.
- Es darf stets nur eine Scheibe, und zwar die oberste eines Turmes bewegt werden.
- Zu keiner Zeit darf eine größere Scheibe auf einer



Turm von Hanoi — Laufzeitanalyse

Um einen Turm mit n Zügen umzusetzen, benötigt man mindestens $2^n - 1$ Züge.

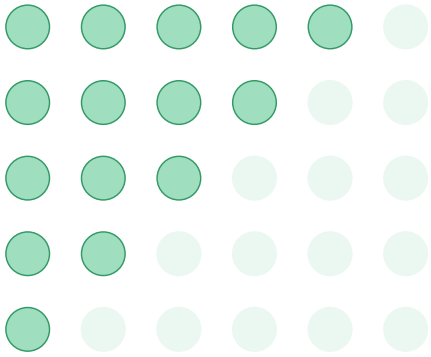


Begründung mittels vollständiger Induktion. . .

Beispiel: Anstoßproblem

Wenn n Menschen zusammenstehen und jeder mit jedem anderen mit seinem Glas anstößt, wie oft klingelt es dann?

Es klingelt $\frac{n(n-1)}{2}$ mal.



Beispiel: Rasen mähen

Wie lange braucht man um n Quadratmeter Rasen zu mähen, wenn man für einen Quadratmeter t_0 Sekunden benötigt?

Es dauert $t_0 \cdot n$ Sekunden.

Beispiel: Zahlenraten linear

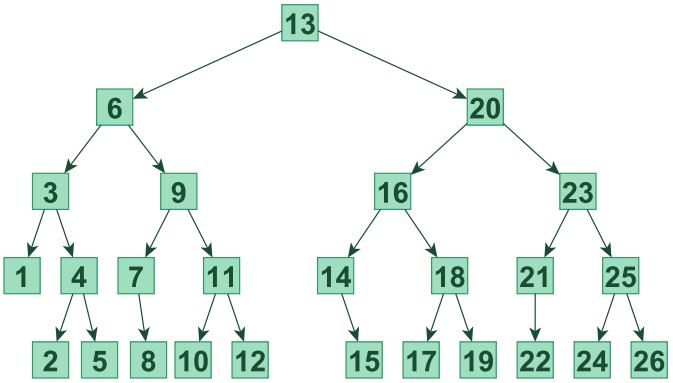
Wie viele Versuche braucht im best case und im worst case um aus n natürlichen Zahlen die richtige Zahl herauszufinden?

best case:	1
worst case:	n
average case:	$\frac{1}{2}n$

Beispiel: Zahlenraten schnell

Wie viele Versuche braucht im best case und im worst case um aus n natürlichen Zahlen die richtige herauszufinden? Man darf nach jedem Tipp testen fragen, ob die gesuchte Zahl größer oder kleiner ist.

best case:	1
worst case:	$\lceil \log_2(n+1) \rceil$



Komplexitätsterme

Komplexitätsterme

Bei der Laufzeit von Programmen will man wissen, wie sich der Aufwand für Zeit, Kosten, Einzelschritte o.ä. mit der Anzahl der zu verarbeitenden Elemente n entwickelt. Diese Entwicklung wird durch einen Komplexitätsterm beschrieben.

Vorgang	Komplexitätsterm
Rasenmähen	$t \cdot n$
Anstoßen	$\frac{1}{2}n^2 - \frac{1}{2}n$
Umschichten	$2^n - 1$
lin. Z-Raten (best case)	1
lin. Z-Raten (worst case)	n
schnelle Z-Raten (worst case)	$\lceil \log_2 n \rceil$

Übung

- Ü 1.1: Laufzeitanalyse

Für die obigen Beispiele sollen die Zeiten für verschiedene Werte von n verglichen werden. Beim linearen uns schnellen Suchen betrachten wir nur den „worst case“.

(a) Fertige eine Tabelle und ein Diagramm an, aus der hervorgeht, wie viel Zeit vergeht, um den Vorgang mit 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100, 200 Objekten durchzuführen. Geh davon aus, dass für $n = 1$ jeweils eine Sekunde benötigt wird.

(b) Wir gehen davon aus, wir hätten für jeden Vorgang 1, 10, 100, 1000 Jahre Zeit. Mit wie vielen Objekten können wir diesen Vorgang durchführen (Wieder eine Sekunde pro Einzelschritt).

Komplexitätsklassen

Komplexitätsklassen

Bei der Laufzeit von Programmen interessiert man sich oft nicht für den exakten Komplexitätsterm, sondern nur dafür, welcher Typ von Grundfunktion das Laufzeitverhalten bestimmt, man sagt, zur welcher Komplexitätsklasse das Problem gehört.

Vorgang	Term	Typ	Komplexitätsklasse
Rasenmähen	$t \cdot n$	linear	$\mathcal{O}(n)$
Anstoßen	$\frac{1}{2}n^2 - \frac{1}{2}n$	quadratisch	$\mathcal{O}(n^2)$
Umschichten	$2^n - 1$	exponentiell	$\mathcal{O}(e^n)$
lin. Z-Raten (bc)	1	konstant	$\mathcal{O}(0)$
lin. Z-Raten (wc)	n	linear	$\mathcal{O}(n)$
schn. Z-Raten (wc)	$\lceil \log_2 n \rceil$	logarithmisch	$\mathcal{O}(\ln n)$

Weitere Vergröberung der Komplexitätsklassen

Grundlegend lassen sich drei Komplexitätsklassen unterscheiden:

Komplexitätsklasse $\mathcal{O}(e^n)$: Exponentielles Laufzeitverhalten. Diese Probleme sind nur für kleine n effizient lösbar.

Komplexitätsklasse P: Probleme sind in polynomialer Zeit lösbar.

Komplexitätsklasse NP: Bei Probleme dieser Klasse kann in polynomialer Zeit überprüft werden, ob eine vorgegebene Lösung richtig ist.

- Typisches NP-Problem: Handlungsreisender
`http://www.geosimulation.de/umsetzungen/2_0_2_Modelle/Problem_des_Handlungsreisenden.html`

Die 1-Million-Dollar–Frage

Die 1-Million-Dollar–Frage lautet:

$$P = NP?$$

D.h. wenn es einen Algorithmus gibt, der die Richtigkeit einer Lösung in polynomialer Zeit überprüft, gibt es dann auch einen Algorithmus, der die Lösung in polynomialer Zeit findet?

- Milleniumsprobleme
`http://de.wikipedia.org/wiki/Millennium-Probleme`

In diesem Abschnitt

Laufzeitanalyse
Alltagsbeispiele
Experimentelle Laufzeitanalyse
Laufzeitabschätzung ausgewählter Algorithmen

Berühmte Algorithmen

- Dijkstra-Algorithmus1
`http://www.unf.edu/~wkloster/foundations/DijkstraApplet/DijkstraApplet.htm`
- Dijkstra-Algorithmus2
`run:Simulationen/DijkstraVis/DijkstraVis/DijkstraVis.bat`
- Sortieralgorithmen
`http://www.sorting-algorithms.com/`
- Mergesort
`http://en.wikipedia.org/wiki/Merge_sort`
- Selectionsort
`http://en.wikipedia.org/wiki/Selection_sort`

Übung

- Ü 1.2: Experimentelle Laufzeitanalyse
  `Arbeitsmaterial/ExperimentelleLaufzeitanalyse/ExperimentelleLaufzeitanalyse.pdf`
- Ü 1.3: S.140/7

In diesem Abschnitt

Laufzeitanalyse
- Alltagsbeispiele
- Experimentelle Laufzeitanalyse
- Laufzeitabschätzung ausgewählter Algorithmen

Typische Ausführungszeiten elementarer Operationen in einer Hochsprache

Anweisung	10^{-9} s
Wertzuweisung	0,5
Summe ganzer Zahlen	0,5
Produkt ganzer Zahlen	1
Attributdeklaration	2
Summe Dezimalzahlen	2
Produkt Dezimalzahlen	3
Vergleich	3
Quotient Dezimalzahlen	15
Objekt erzeugen	15

Zeitkritische Operationen

Zeitkritische Operationen

in einer Hochsprache wie Java sind

- das Erzeugen von Objekten,
- das Rechnen mit Dezimalzahlen (v.a. die Division)
- das Vergleichen von Objekten und
- als zusammengesetzte Operationen alle Methodenaufrufe.

Diese Operationen sind in Schleifen und rekursiven Aufrufen möglichst zu vermeiden.

Beispiel Fakultät

```
1  public static int fact(int n
     ) {
2    if(n<=1) {
3      return 1;
4    }
5    else {
6      return n * fact(n-1);
7    }
8  }
```

n	1	2	3	4	5	n
Anzahl der Aufrufe	1	2	3	4	5	n

K-Klasse: $\mathcal{O}(n)$

Beispiel Fibonacci-Folge

```
1  public int fibonacci(int n)
     {
2    if(n == 0 || n == 1) {
       return 1; }
3    else {
4      return (fibonacci(n-1) +
           fibonacci(n-2));
5    }
6  }
```

n	1	2	3	4	5	6	7	n
Anzahl der Aufrufe	1	3	5	9	15	25	41	$\approx fib(n)$

K-Klasse: $\mathcal{O}(fib(n)) \approx \mathcal{O}(2^n)$

Beispiel Selectionsort Pseudocode

```
1   Selectionsort (DATENELEMENT [] d)
2     wiederhole von i = 1 bis (d.length
        - 1)
3       pos = i;
4       wiederhole von j = i + 1 bis d.
          length
5         wenn d[j] < d[pos]  0) //
            zeitkritisch
6           pos = j
7         endewenn
8       endewiederhole
9       tausche Inhalte der Zellen d[i]
          und d[pos];
10    endewiederhole
```


Übung

- Ü 1.4: Laufzeitanalyse Selectionsort (siehe vorangegangenes Beispiel)

(a) Erstelle für den Aufruf `Selectionsort({5, 7, 3, 2, 6})` eine Variablenbelegungstabelle aller auftretenden Variablen.

(b) Implementiere Selectionsort in Java.

(c) Bau in die Implementierung einen Zähler ein, der zählt, wie oft die zeitkritische Programmzeile 5 durchlaufen wird, und gib für Datenelemente der Länge $n = 1, 2, \ldots 10$ die Anzahl der Durchläufe an.

(d) Verändere den Zähler so, dass ersichtlich ist, wie oft Programmzeile 5 in Abhängigkeit von i und n durchlaufen wird.

(e) Gib mit Begründung den Komplexitätsterm und die Komplexitätsklasse des Algorithmus an.

Übung

- Ü 1.5: Laufzeitanalyse Insertionsort
`run:Arbeitsmaterial/Insertionsort/Insertionsort.pdf`
- Ü 1.6: Ackermannfunktion (Klett, 132/7)
`run:Arbeitsmaterial/AckermannFunktion/AckermannFunktion.pdf`

In diesem Abschnitt

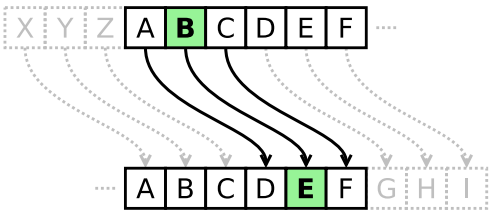
Ausblick in die Kryptographie
Historische Kryptographie
Moderne Kryptographie

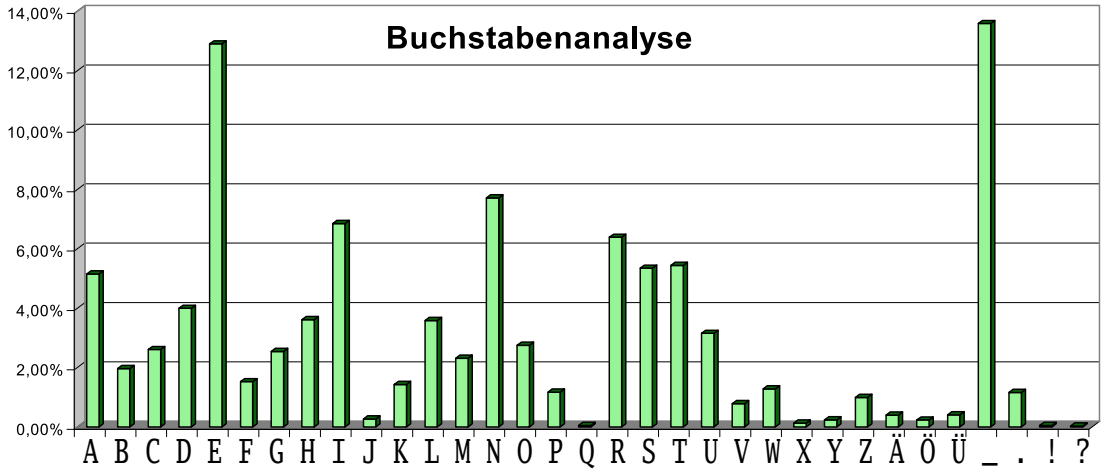
Begriffsbestimmung

Kryptographie

befasst sich mit der Konzeption, Definition und Konstruktion von Informationssystemen, die widerstandsfähig gegen unbefugtes Lesen und Verändern sind.

Caesar-Chiffre





Caesar-Chiffre

- Caesar-Verschlüsselung
  `http://www.cryptool-online.org/index.php?option=com_cto&view=tool&Itemid=96&lang=de`
- Caesar-Verschlüsselung (Matheprisma LP)
  `http://www.matheprisma.uni-wuppertal.de/Module/Caesar/index.htm`

Vigenère-Verschlüsselung

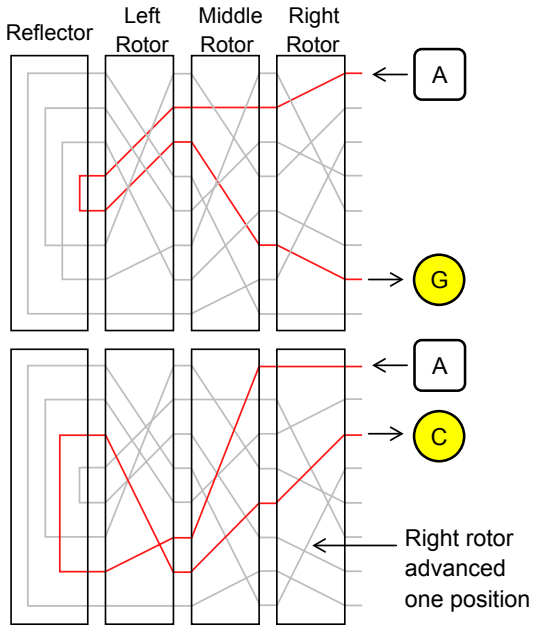
G E H E I M N I S
a k e y a k e y a
g o l c i w r g s

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenère-Verschlüsselung

- Vigenère-Verschlüsselung
  `http://www.cryptool-online.org/index.php?option=com_cto&view=tool&Itemid=128&lang=de`
- Vigenère-Verschlüsselung
  `http://einklich.net/etc/vigenere.htm`

Enigma



Enigma

- Enigma (Wikipedia)
  http://de.wikipedia.org/wiki/Enigma_(Maschine)
- Enigma (Krytotools)
  http://www.cryptool-online.org/index.php?option=com_cto&view=tool&Itemid=105&lang=de
- Enigma (Matheprisma)
  http://www.matheprisma.uni-wuppertal.de/Module/Enigma/index.htm

Kerckhoff'sches Prinzip

Kerckhoff'sches Prinzip

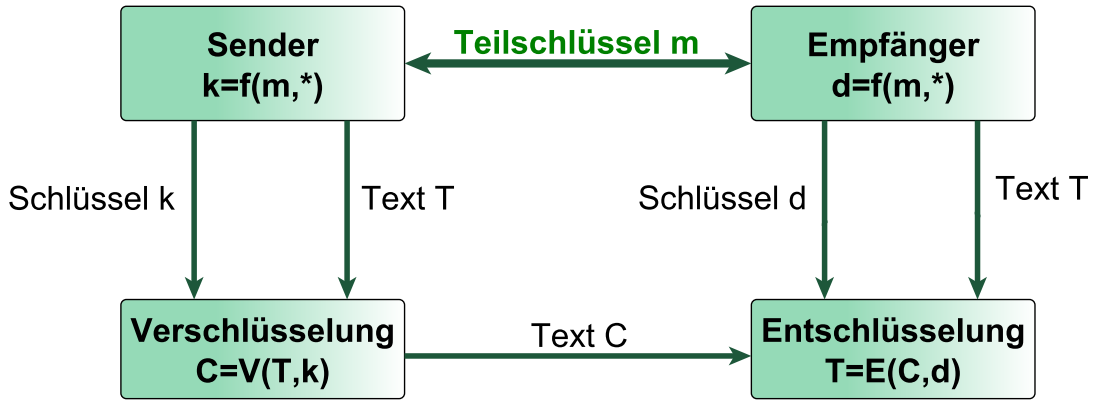
Die Sicherheit eines Kryptographiesystems darf nicht von der Geheimhaltung des Algorithmus abhängen, sondern nur auf der Geheimhaltung des Schlüssels.

Oberstes Ziel der Kryptographie muss ein sicheres Schlüsselmanagement sein.

In diesem Abschnitt

Ausblick in die Kryptographie
Historische Kryptographie
Moderne Kryptographie

Ver- und Entschlüsselungsvorgang

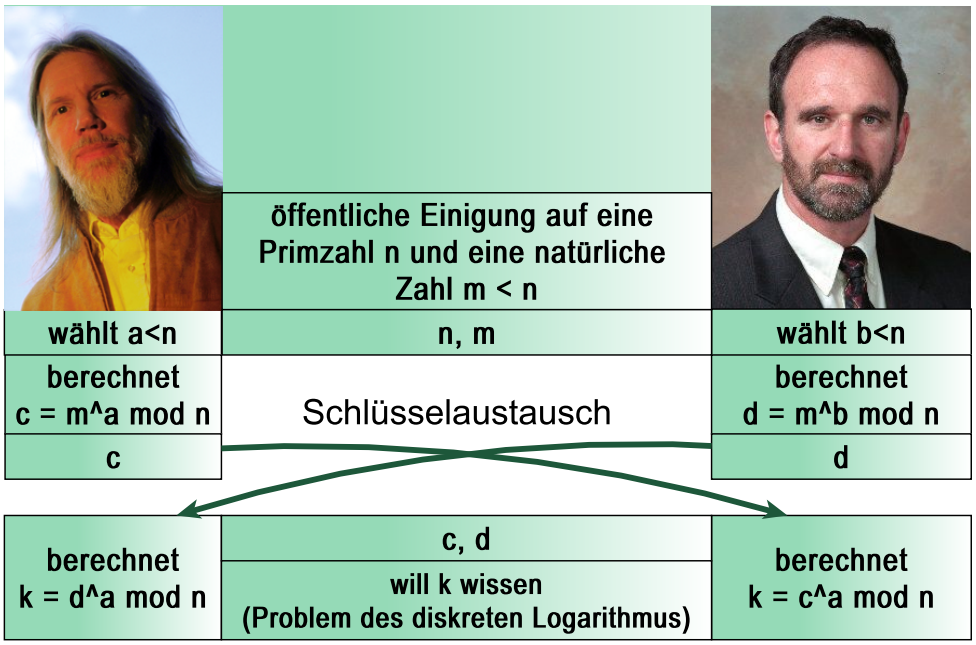


Unglaubliche Möglichkeiten der modernen Kryptographie

Public-Key-Algorithmus Zwei Personen, die bislang noch nie Kontakt hatten, können durch Austausch eines öffentlichen Teilschlüssels einen gemeinsamen Gesamtschlüssel erhalten, ohne dass die mithörende Umgebung diesen Schlüssel erhält (Diffie und Hellman, 1976).

Zero-Knowledge-Protokolle Man kann geheime Nachrichten auch übertragen, ohne einen gemeinsamen Schlüssel zu besitzen (Fiat und Shamir, 1985)

Diffie-Hellman-Schlüsselaustausch (vereinfacht)

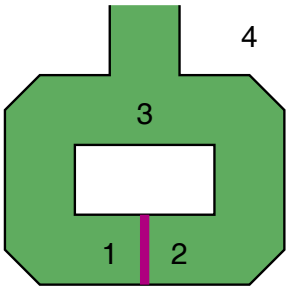


- RSA (Matheprisma)
  `http://www.cryptool-online.org/index.php?option=com_cto&view=tool&Itemid=128&lang=de`

Zero-Knowledge-Protokoll (schematisch)

Geheimnis von Alice: Schlüssel für die rote Tür.

- Bob steht bei 4 und sieht Alice in die Höhle gehen aber nicht ob sie nach 1 oder 2 geht.
- Bob geht zu Position 3 und verlangt von Alice, dass sie auf einer von ihm festgelegten Seite aus der Höhle kommt.
- Alice und Bob gehen zu Position 4.
- Dieser Vorgang wird mehrfach wiederholt.



Anwendungen der modernen Kryptographie

- Digitale Signaturen
- Elektronische Zahlungssystem
- Chipkartenauthentifikation
- E-Mail-Verkehr (OpenPGP, S/MIME)
- SSH, SSL, https

Typische Schlüssel moderner Kryptographieprotokolle

```
1071508607186267320948425049060001810561404811 70

553360744375038837035105112493612249319837881569

585812759467291755314682518714528569231404359845

775746985748039345677748242309854210746050623711

418779541821530464749835819412673987675591655439

460770629145711964776865421676604298316526243868

37205668069673
```

Typische Schlüssel sind

Primzahlen und daraus abgeleitete Potenzen und Produkte mit

Das Knacken der Schlüssel

. . . beruht oft darauf, dass

- eine Zahl faktorisiert oder
- ein diskreter Logarithmus berechnet werden muss, d.h. die Lösung x der Gleichung $y = a^x \mod p$ zu bestimmen ist.

Das Knacken der Schlüssel

ist im Allgemeinen nur durch einen brute-force-Angriff, d.h. das Ausprobieren aller Kombinationen möglich. Dieser brute-force-Angriff gehört aber zu der Komplexitätsklasse NP und weil erwartungsgemäß $P \neq NP$ angenommen wird, gelten die Schlüssel bei angemessener Schlüssellänge als sicher.

Kryptografische Angriffe

Weil brute-force-Angriffe nicht zum Erfolg führen, wird versucht aus folgenden Informationen den Schlüssel zu ermitteln:

Known ciphertext attack: beliebig viel Geheimtext,

Known plaintext attack: einen kleines Stück von zusammengehörigem Klar- und Geheimtext,

Chosen plaintext attack: selbst gewählter Text wird mit dem aktuellen Schüssel codiert,

Man-in-the-middle attack: abhören und manipulieren des Informationskanals ohne dass es die Kommunikationspartner merken.

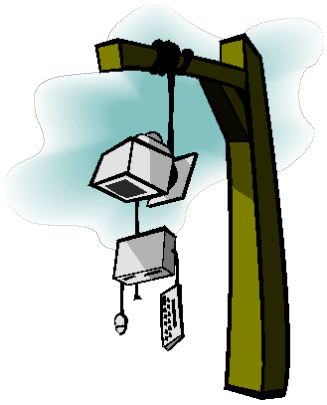
Wiederholung Algorithmus

Definition

Ein Algorithmus ist eine Verarbeitungsvorschrift, die aus endlich vielen, eindeutig ausführbaren Anweisungen besteht.

Zentrale Frage

Muss ein Algorithmus zwangläufig terminieren, d.h. nach endlicher Zeit enden?



Forumlierung des Halteproblems

Forumlierung des Halteproblems

Kann es einen Algorithmus geben, der für einen beliebigen anderen Algorithmus testen kann, ob dieser bei vorgegebenen Eingabeparametern terminiert.

Beachte

- Die Aussage ist unabhängig vom technologischen Stand formuliert und bezieht sich auf abstrakte Maschinen (z.B. Turingmaschinen), die nicht gebaut werden müssen.

  - Turing-Maschine
  `http://www.matheprisma.uni-wuppertal.de/Module/Turing/`
- Datentypen o.ä. gibt es nicht, nur Bitfolgen, die nach einer Vorschrift abgearbeitet werden.

Satz

Satz

Das Halteproblem ist nicht in allen Fällen lösbar, d.h. es gibt Algorithmen bei denen man grundsätzlich nicht feststellen kann, ob diese terminieren oder nicht.

Beweis durch Widerspruch

1. Schritt Annahme: Es gibt einen Algorithmus `HALT`, der berechnet, ob ein Programm `prog` für bestimmte Eingaben `eing` terminiert.

```
HALT( prog , eing )
        wenn prog( eing )
            terminiert
            Ausgabe wahr
        sonst
            Ausgabe falsch
        endewenn
```


Beweis durch Widerspruch

2. Schritt Wir formulieren folgendes Programm `TEST`.

```
TEST( prog )
        wiederhole solange
            HALT ( prog ,
            prog ) = wahr
        endewiederhole
```

Das Programm `TEST` terminiert nicht, wenn `HALT` terminiert.

Beweis durch Widerspruch

3. Schritt TEST wird mit sich selbst als Parameter aufgerufen

```
TEST( TEST )
            wiederhole solange
                HALT ( TEST ,
                TEST ) = wahr
            endewiederhole
```

Das Programm TEST mit dem Parameter TEST terminiert also, wenn es nicht terminiert. ↯
Somit kann es kein Programm HALT geben, das feststellt, ob ein beliebiger Algorithmus terminiert.

Zusammenfassung

Zusammenfassung

Eine Funktion heißt in der Informatik berechenbar, wenn es eine Turing-Maschine gibt, die sie löst (also ihren Ergebnis berechnet).
Dabei lassen sich folgende Aussagen treffen:

1. Probleme, die nicht durch eine Turing-Maschine formalisiert werden können, können auch nicht berechnet werden.
2. Es gibt formal beschreibbare Probleme, die prinzipiell berechnet werden können, deren Berechnung aber viel zu lange dauert.
3. Es gibt formal beschreibbare Probleme / Aussagen, die prinzipiell nicht berechnet / bewiesen werden können.