# Theoretische Informatik
## Der Algorithmusbegriff

Uwe Menzel, 1998

uwe.menzel@matstat.de

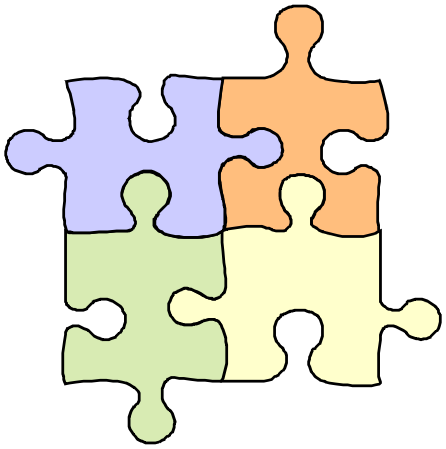
www.matstat.org

# Inhaltsverzeichnis

# Literatur und Links

- David Harel, „Algorithmics“, Addison-Wesley, 1987
- F.S. Beckmann, „Mathematical Foundations of Programming“, Addison-Wesley, 1980.
- F. Kröger, „Einführung in die Informatik“, Springer 1991
- R. Sedgewick, „Algorithmen“, Addison-Wesley, Bonn 1994
- Simon Singh: „Fermats letzter Satz“, dtv
- Online-Numerical Recipes: http://www.ulib.org/webRoot/Books/Numerical_Recipes/
- Quicksort: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/quick/quick.htm
- Game of Life online: http://www.bitstorm.org/gameoflife/
- Claymath: http://www.claymath.org/prizeproblems/statement.htm 
- TM holen: http://www.dbg.rt.bw.schule.de/lehrer/ritters/info/turing/einf.htm
- Eliza-Programm (englisch) http://www.unet.univie.ac.at/~a9725261/eliza.htm
- http://mzs.sowi.uni-goettingen.de/mitarbeitende/arnhold/tesmik/einfuehrung.html
- mit Eliza chatten: http://www.allkuma.de/eliza.htm (auf Deutsch)
- "Kann eine Maschine denken": http://www.uni-ulm.de/~s_mbrusd/aturing.html
- Türme von Hanoi: http://www.cut-the-knot.com/recurrence/hanoi.shtml
- Traveling Salesman Problem, Princeton: http://www.math.princeton.edu/tsp/

# Algorithmen überall …



# 1. Intuitive Algorithmusdefinition

**Definition** # 1:

Ein Algorithmus ist eine endliche und präzise Vorschrift zur schrittweisen Lösung einer bestimmten Klasse gleichartiger Probleme. Jeder Schritt besteht aus einfachen und offensichtlichen Grundaktionen.

**Definition** # 2:

Unter einem Algorithmus versteht man eine präzise, schrittweise und endliche Verarbeitungsvorschrift zur Lösung einer Klasse von Problemen, die so formuliert ist, dass die einzelnen Operationen von einer Maschine ausgeführt werden können.

# Kopf frei für neue Aufgaben

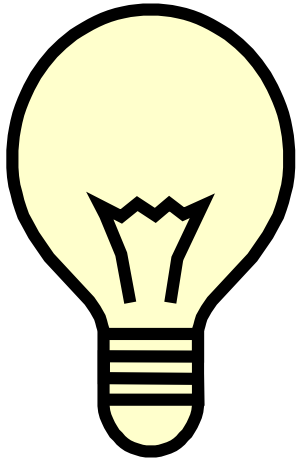
- Algorithmus bekannt ⇨ für die Lösung eines Problems dieser Klasse keine schöpferische Arbeit mehr notwendig.
  - Problem der betreffenden Klasse kann durch schematisches Befolgen der algorithmischen Vorschrift gelöst werden.
  - **Beispiele**: Multiplikation von ganzen Zahlen, Lösung einer quadratischen Gleichung mit Hilfe der Lösungsformal.
- "The ultimate goal of mathematics is to eliminate all need for intelligent thought." [Graham, Knuth, Patashnik: Concrete Mathematics. Addison-Wesley, 1989] ☺

# 2. Kurze Geschichte der Algorithmen

| | |
|---|---|
| 365-300 v.u.Z. | Euklid: Berechnung des g.g.T.<br>erster bekannter nicht-trivialer Algorithmus |
| 230 v.u.Z. | Primzahlen: Sieb des Eratosthenes |
| um 825<br>Bagdad | Muhamad Al Chwarismi schreibt ein Algebra-Lehrbuch ⇨Algorismi ⇨Algorithmus |
| 1050 | Chinesische Gelehrte entwickeln das Dualsystem |
| 1547 | Adam Riess: Rechenbuch zur Rechnung im Dezimalsystem |
| 193x | A. Turing: mathem. Modell für einen Universalrechner. Fortschritte in der Theorie durch Turing, Gödel, Markov, Church |

# 3. Eigenschaften von Algorithmen

(1) Allgemeinheit
(2) Terminierung
(3) Determinismus
(4) Determiniertheit



Algorithmus ≠ Programm

# (1) Allgemeinheit

- Es wird eine ganze Klasse von Problemen gelöst.
- Zur Spezifikation der Aufgabe werden Parameter verwendet:
  - Programm Quadratische Gleichung (Parameter: 2 reelle Zahlen)
  - Bisektion: Nullstelle einer stetigen Funtion (Parameter: Funktion)
  - Lineare Gleichungssysteme (Parameter: Koeffizientenmatrix und Rechte-Seite-Vektor)
  - Compiler (beliebiges, syntaktisch korrektes Programm)

# Allgemeinheit

- Durch die Abstraktion/Parametrisierung werden Algorithmen zu abgeschlossenen Modulen (black boxes: ■ )

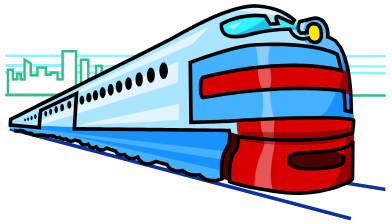
  ⇨ Man steckt sein Problem (Parameter) in den Schwarzen Kasten hinein - heraus kommt die Lösung: 🔓 ⇨ ■ ⇨ 🔓 ☺

- Sehr viele Algorithmen sind inzwischen in allgemein zugängliche in Computer-Programme umgesetzt worden
  - Lineare Algebra FORTRAN Unterprogrammbibliothek: http://www.tu-darmstadt.de/hrz/hhlr/vpp/sw/BLAS.htm
  - Numerical Recipes Online: http://www.ulib.org/webRoot/Books/Numerical_Recipes

# (2) Terminierung

- Satz: Algorithmen nennt man terminierend, wenn er für *jedes* Problem einer Klasse nach endlich vielen Verarbeitungs-Schritten anhält und ein Resultat liefert.
  - Das Terminieren eines Algorithmus darf nicht mit seiner Finitheit verwechselt werden: durch eine endliche Beschreibung kann einen Prozeß definiert werden, der nicht terminiert, z.B. Endlosschleife).
  - Programmierer muss Terminierung managen: z.B. Bisektion

- Gegenbeispiel: Algorithmen, die nicht terminieren sollen:
  - Betriebssystem eines Web-Servers oder einer Workstation
  - Steuerung von Signalanlagen bei der Bahn



# (3) Determinismus


- Einen Algorithmus nennt man deterministisch, wenn zu jedem Zeitpunkt seiner Ausführung der Folgeschritt eindeutig bestimmt ist
  - Beispiel: Lösung einer quadratischen Gleichung: für die gleichen Parameter p und q durchläuft der Algorithmus genau denselben Weg.
  - if-then Konstrukte oder Ähnliches jedoch erlaubt
- nicht-deterministische Algorithmen: stochastische Algorithmen
  - Flächen- oder Volumenberechnungen. Bei erneutem Durchlauf des Programms werden andere Zufallszahlen generiert.
  - Stochastischer Quicksort-Algorithmus

# (4) Determiniertheit

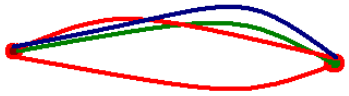
- **Satz**: Ein Algorithmus heißt determiniert, wenn er mit den gleichen Parametern und Startbedingungen stets das gleiche Ergebnis liefert.
  - Der Weg von den Eingabewerten zu den Ausgabewerten kann aber von Mal zu Mal verschieden sein.
- Beispiel: Volumen-/Flächenberechnung
- Beispiel Quicksort: Das Pivot-Element wird durch eine Zufallszahl bestimmt und ist deshalb bei jedem Durchlauf verschieden. Das Resultat ist aber stets dasselbe: das sortierte Feld.
  - http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/quick/quick.htm
- Nicht-determinierte Algorithmen: Zufallsgeneratoren, Spielprogramme

# Determinismus und Determiniertheit

*deterministisch*

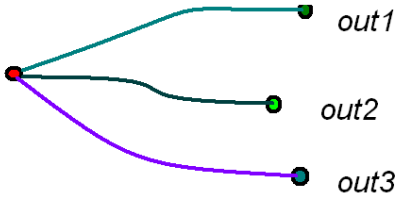
*Input* *Output*

*determiniert*



*Input* *Output*

*nicht determiniert*



# 4. Existenz von Algorithmen

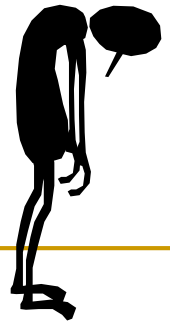
- Kann man für *jedes* Problem einen Algorithmus finden (und die Lösung von einem Computer errechnen lassen) ?
- Leibnitz (1646-1716): Jedes mathematische Problem ist durch einen Algorithmus lösbar.
- David Hilbert (1862-1943): Ein bestimmtes mathematisches Problem muss notwendigerweise einer exakten Lösung zugänglich sein, …
- Kurt Gödel (1906-78): Unvollständigkeitssatz, 1931:

  Es existieren innerhalb eines Axiomensystems Aussagen, die innerhalb dieses Axiomensystems weder bewiesen noch widerlegt werden können.

  (Antinomie: Der Kreter sagt: „Alle Kreter lügen“)

# Ein bisschen Pessimismus

- Es kann gezeigt werden, dass manche Probleme *überhaupt nicht lösbar* sind, weder mit den besten denkbaren Computern, noch in aller zur Verfügung stehenden Zeit.
- Wenn für ein Problem kein Lösungsalgorithmus existiert, dann bedeutet dies insbesondere auch, dass kein Computerprogramm zur Lösung geschrieben werden kann, egal welche Programmiersprache (oder welcher Computer) verwendet wird. Alle Programmiersprachen sind in dieser Hinsicht gleichwertig.



# Entscheidungsprobleme:

- Entscheidungsproblem verlangt dualen Output: 
- Beispiel: Gibt es einen Algorithmus, der für eine gegebene gerade Zahl $n \geq 4$ entscheidet, ob sie die Goldbach-Eigenschaft hat oder nicht ?

  ☞ Es gibt ihn: Beispielprogramm
- Gib eine Zahl aus der erlaubten Input-Menge ein und Du erhältst nach endlicher Zeit **ein JA oder ein NEIN**.

⇨ die Goldbach-Eigenschaft ist **entscheidbar** !!

- Anmerkung: bisher hat noch niemand eine Zahl gefunden, die nicht die Goldbach-Eigenschaft hat, aber im Sinne der obigen Problemformulierung ist das Problem gelöst

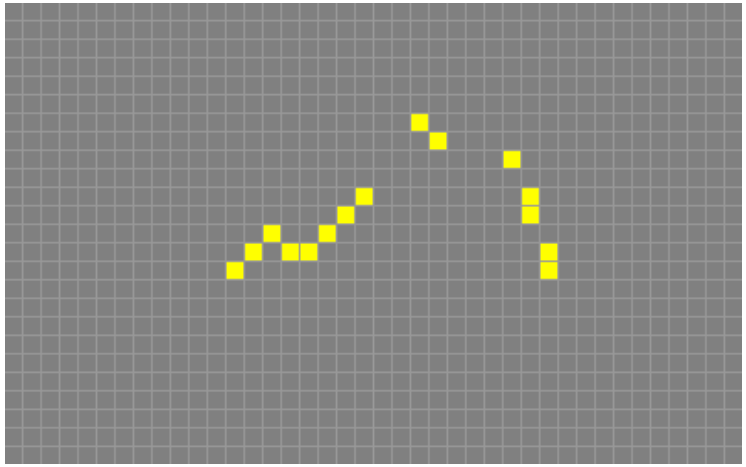
# Wundersame Zahlen

- Gibt es einen Algorithmus, der für eine gegebene natürliche Zahl n entscheidet, ob sie wundersam ist oder nicht ?
- Siehe folgendes Programm: wundersam.pl
  - Ist n wundersam, so liefert dieser Algorithmus nach endlich vielen Schritten die Ausgabe "Wundersam“.
  - Gibt man aber eine "unwundersame" Zahl ein, kommt der Algorithmus zu keinem Ende und damit zu keiner Entscheidung.
- ⇨ Das Problem ist mit Hilfe dieses konkreten Algorithmus nur partiell entscheidbar.
- Es könnte allerdings einen Algorithmus geben, der auch die Unwundersamkeit feststellen kann (noch nicht gefunden).
- Anmerkung: Eine „unwundersame“ Zahl wurde noch nicht gefunden.

# Das Spiel des Lebens

http://www.math.com/students/wonders/life/life.html

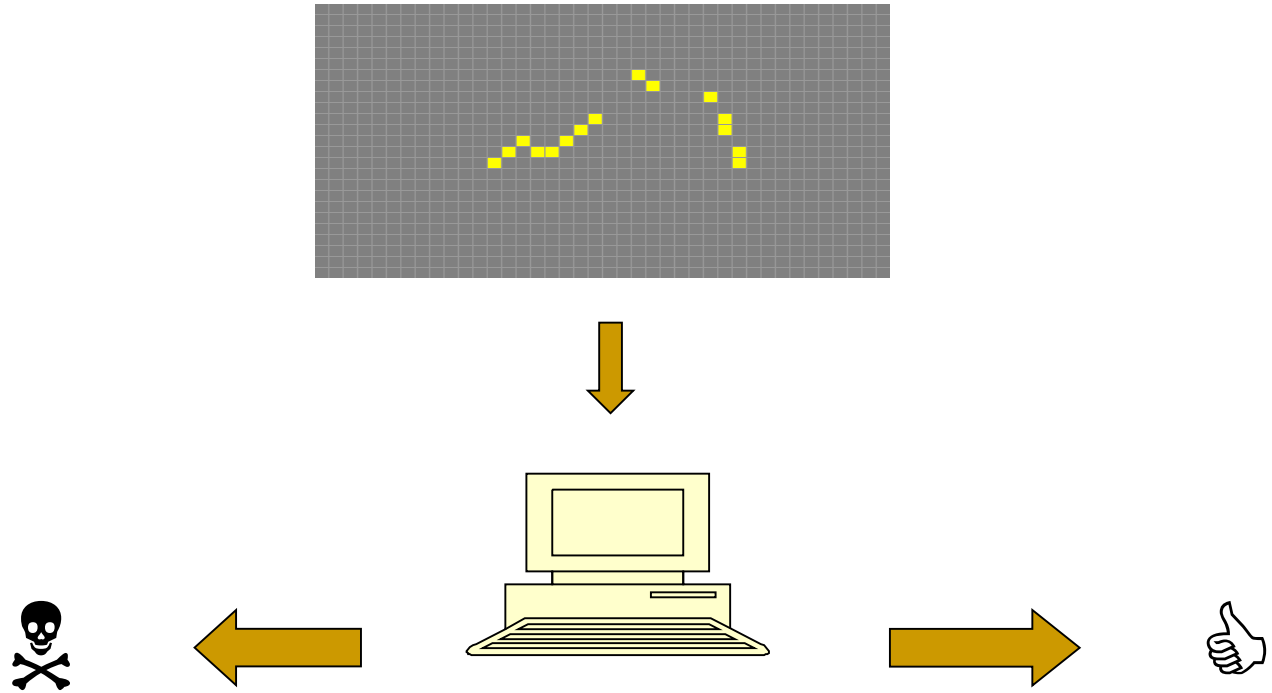
- Eine tote Zelle lebt auf, wenn 3 ihrer 8 Nachbarn leben (Fortpflanzung).
- Eine Zelle stirbt, wenn mindestens 4 Nachbarn leben (Übervölkerung).
- Eine Zelle stirbt, wenn höchstens 1 Nachbar lebt (Vereinsamung).
- http://www.bitstorm.org/gameoflife/ (Online-Spiel)



# Cleveres Programm gesucht

Manche Anfangspopulationen sterben schnell aus, andere scheinen ewig zu leben.

**Gesucht**: Algorithmus, der für eine beliebige Anfangsgeneration des "Game of Life" entscheidet, ob diese sterblich ist oder nicht.



# Geheimnis des Lebens nicht zu lüften …

- Algorithmus könnte einfach das Spiel nachvollziehen:
  - Stirbt Population, so ist die Sterblichkeit festgestellt.
  - Stirbt sie aber nicht, kann man keine Schlussfolgerung ziehen

- Gibt es eine anderen, raffinierteren Algorithmus ?
  - **NEIN**, für das Game of Life ist inzwischen bewiesen, daß kein zweiseitiges Entscheidungsverfahren existiert.

# Das berühmte „Halteproblem“

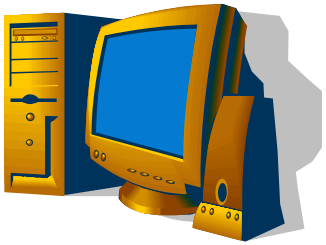
- Programmier – Fehler: Endlosschleife 
- **Gesucht:** ein Programm, das ein anderes Programm auf das Vorhandensein von Endlosschleifen untersuchen könnte
- Sollten nicht Softwarefirmen  weltweit Anstrengungen unternehmen, um das Halteproblem zu lösen ?

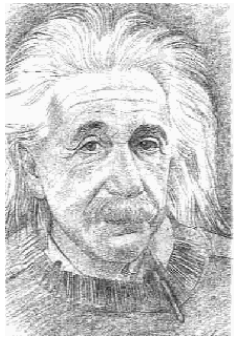
NEIN: ☞ Das Halteproblem ist nicht lösbar ☹

Es existiert kein Algorithmus, der bei Eingabe eines Programmes R und dessen Input X sagen kann, ob R mit der Eingabe X terminiert oder nicht.

# Ein Tröstender Gedanke

- Die Arbeit von Programmierern wird nie überflüssig!
- Es gibt Probleme, die von Menschen durch Nachdenken gelöst werden können, aber nicht von einem Computer.
- Irgendwie muss ein Gehirn ganz anders als ein Computer arbeiten …. ?





# Ein letztes Beispiel für ein unlösbares Problem

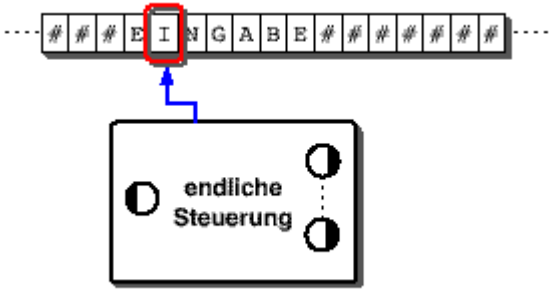
- 10. Hilbertsches Problem (über die Lösbarkeit von diophantischen Gleichungen)
- David Hilbert, 1900 in Paris auf dem 2. Internationalen Mathematikerkongress: 23 Probleme
- Folgemeeting 2000, ebenfalls in Paris
- Sieben neue Probleme: http://www.claymath.org/prizeproblems/statement.htm

# 5. Die Turing-Maschine

- Wie wird nachgeweisen, ob ein Problem algorithmisch lösbar ist oder nicht ?
- ⇨ Notwendigkeit einer präzisen Algorithmusdefinition

- Alan Turing: "On computable numbers, with an application to the Entscheidungsproblem" Proceedings of the London Mathematical Society (2) 42, 1937.
  - Einführung des Konzeptes der Turingmaschine (TM)
  - mathematisches Modell zur Untersuchung prinzipieller Fragen der Berechenbarkeit

# Aufbau der Turingmaschine

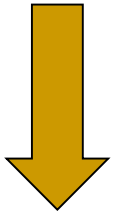



- äußerer linearer Speicher
  - in einzelne Felder eingeteilt
  - Je ein Feld enthält ein Zeichen eines vorgegebenes Bandalphabets
  - Bandalphabet muss Leerzeichen beinhalten
- Lese/Schreibkopf
  - kann ein Zeichen in einem Feld lesen bzw. schreiben
- innerer Speicher (Register)
  - befindet sich zu jedem Zeitpunkt in einem definierten Zustand

# Das „Programm“ der TM

aktuell gelesenes Zeichen

aktueller Registerzustand



Abbildungsvorschrift,
Zustandstabelle

welches Zeichen soll auf das Band geschrieben werden ?

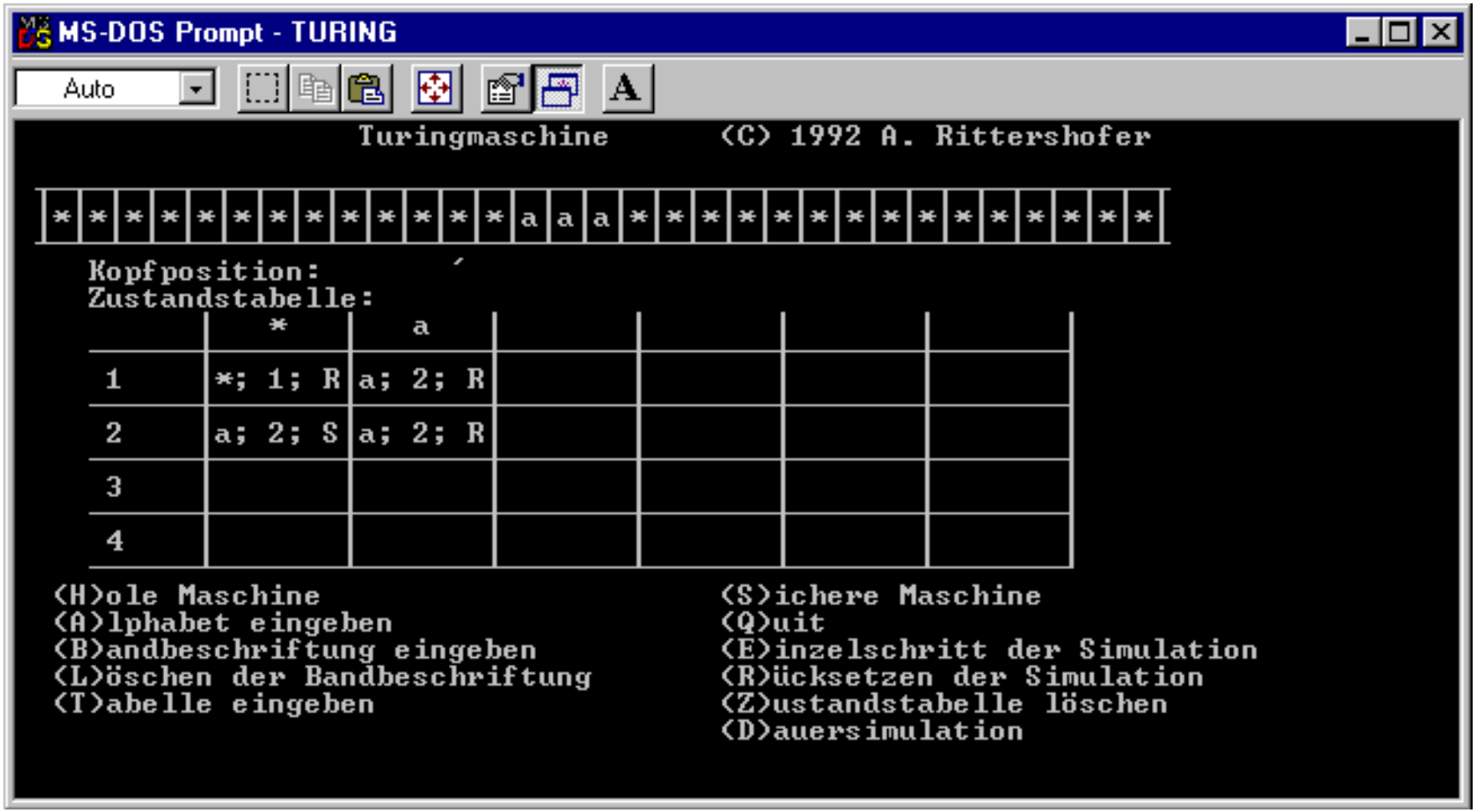
in welchen Zustand soll das Register übergehen ?

wie soll sich das Band bewegen: rechts (R), links (L) oder halten (H)

# Wozu diese primitive Maschine ?

- Die TM ist das einfachste Gedankenmodell für einen Computer
- Elementarer Befehlssatz:
  - jeder allgemeine Rechenvorgang kann in eine Folge dieser einfachsten Befehle zerlegt werden kann.
- Die Zerlegung algorithmischer Prozesse in einfachste Operationen wurde absichtlich auf die Spitze getrieben,
  - damit lassen sich exakte mathematische Beweise über Existenz oder Nichtexistenz von Algorithmen führen.

# Ein Beispiel für die Funktionsweise der TM




Download: http://www.dbg.rt.bw.schule.de/lehrer/ritters/info/turing/einf.htm

# Bedeutung der Turingmaschine

- **Kein** Modell für die Arbeitsweise realer Computer. Für praktische Anwendungen war sie nie gedacht.
- Aber sehr große mathematische Bedeutung:
  - Jedes Problem, das überhaupt irgendwie algorithmisch lösbar ist, lässt sich auch auf einer TM lösen !! (mit geeignetem Alphabet und Zustandstabelle)
  - **Church-Turing-These**: Die Klasse der intuitiv berechenbaren Funktionen ist gleich der Klasse der Turing-berechenbaren.
  - Jeder Algorithmus, den wir finden können, in welcher Programmiersprache auch immer, auf welchem Computer auch immer (einschließlich noch nicht gebauter), kann auch auf einer TM gelöst werden !!

# Bedeutung der TM

- Es gilt auch die Umkehrung:
  - alles, was prinzipiell nicht mit der TM berechenbar ist, lässt sich überhaupt nicht berechnen. („Indikatorfunktion“)
  - Turing konnte mit der TM zeigen, daß das X. Hilbertsche Problem unlösbar ist

- Andere (primäre) Formulierung des Halteproblems:
  - Es gibt keine Turing-Maschine, die für eine beliebige Turing-Maschine T und eine beliebige Bandinschrift x die Frage beantwortet, ob T auf x angewendet nach endlich vielen Schritten terminiert oder nicht, d.h. die **Halte-Eigenschaft ist prinzipiell unentscheidbar** (SATZ von TURING).

# Der fleißige Biber

- Tibor Rado, ungar. Mathematiker, 1962: busy-beaver problem
  - Es ist eine TM gegeben, die nur eine bestimmte Anzahl von Zuständen haben darf. Das Turingband ist anfangs leer. Wie viele Zeichen kann sie maximal schreiben?
  - **Bemerkung**: Es ist kein Problem, eine TM zu schreiben, die unendliche viele Zeichen schreibt. Aber die TM soll ja irgendwann anhalten - das macht das Problem so schwierig!

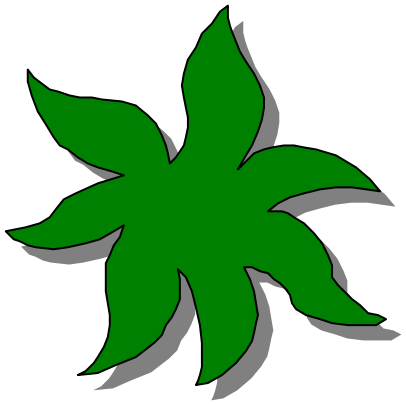
| Anzahl der Zustände der TM | 1 | 2 | 3 | 4 | 5 | 6+ |
|---|---|---|---|---|---|---|
| Max. Anzahl von zu schreibenden Zeichen | 1 | 4 | 6 | 13 | 4098 ? | 🎖 |

# 6. Komplexität von Algorithmen

- *Es gibt Probleme, die zwar theoretisch berechenbar sind, aber wegen des Laufzeitverhaltens der Algorithmen nie auf einer realen Maschine gelöst werden können.*
- Zeitkomplexität eines Algorithmus (RABIN, 1960):
  - Arithm. Funktion, die den Aufwand an Rechenzeit in Abhängigkeit vom Umfang des Problems (Größe Eingabedaten, Ordnung einer Matrix, Grad eines Polynoms) angibt.
  - Aufwand im Mittel (average case) & im schlimmsten Fall (worst case)
- Asymptotische Verhalten: O-Notation (big-O-notation)
  - Sei Eingabedatenmenge durch Zahl n charakterisierbar: Listenlänge, Arraydimension
  - Sei der Aufwand (*Anzahl der auszuführenden Anweisungen*) = A(n)
  - z.B. $A(n) = n^3+5n^2 +10n + 3$ ⇨ Komplexität = **$O(n^3)$** nur höchste Potenz wichtig
  - z.B. $A(n) = 3 + \log(n)$ ⇨ Komplexität = **$O(\log(n))$** Basis nicht wichtig
  - z.B. $A(n) = 2^n -1$ ⇨ Komplexität = $O(2^n)$ exponentiell, Basis nicht wichtig

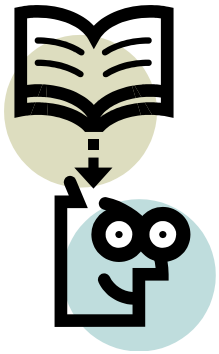
# Komplexität des (Un-)Glücks

- "Sie liebt mich! - Sie liebt mich nicht!" – Algorithmus
- Anzahl der Blütenblätter = N, A(N) = N, Zeitkomplexität O(N), linear
- worst case = average case



# Lineare Suche

- … in einem Telefonbuch, Programm
- Anzahl der notw. Operationen ~ N, der Anzahl der Einträge
- Laufzeit ist eine lineare Funktion zu N:  $L = a \cdot N + b$
- worst case: a=1, average case $a \approx ½$
- nicht wichtig, ob 2N, 3N, oder 100N
- wichtig ist: verdoppelt sich N, so verdoppelt sich die Laufzeit
- **Laufzeitverhalten O(N)**



# Binäre Suche

- … in einem Telefonbuch (alphabetisch geordnet)
- erster Vergleich mit mittlerem Element: ist „Name“ in der 1. oder in der 2. Hälfte ? ⇨ weiter mit dieser Hälfte, usw.
- Wie oft kann man eine Liste mit N Einträgen halbieren, bis nur noch 1 Element (das Gesuchte) übrig bleibt ?

$$\frac{N}{2^k} = 1 \quad \Rightarrow \quad k = \log_2(N)$$

1 Million Einträge: 20 Mal teilen

1 Milliarde Einträge: 30 Mal teilen

$O(\log N)$: logarithmische Komplexität – sehr gut !

# Bubblesort

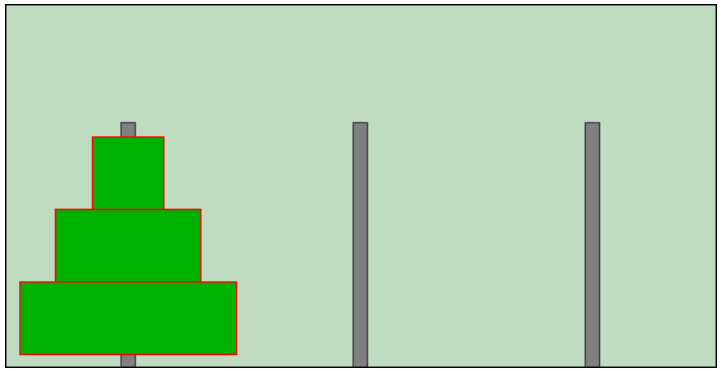
- Indizes laufen über $i = 1, 2, \dots, N$, $j = 1, 2, \dots, i$ ⇨ $\frac{N \cdot (N-1)}{2}$ (halbes Quadrat)
- Anzahl der Durchläufe $\approx N^2/2 - N/2 \sim N^2$
- Nur die höchste Potenz ist relevant (bei großen $N$)
- Zeitkomplexität $O(N^2)$
- Nicht sehr effektiv:
  - Verdoppelt sich $N$, so vervierfacht sich die Zeit
  - Verdreifacht sich $N$, verlängert sich die Laufzeit um das Neunfache
  - …

# Quicksort

- Wählt ein Element aus der Liste aus: das Pivot
- unterteilt ein unsortierte Liste in:
  - Alle Elemente $\leq$ Pivot
  - Alle Elemente > Pivot
- Die Teillisten werden wiederum genauso unterteilt
  - Rekursives Programmieren
  - Prinzip „Teile und Herrsche“
- average case: $O(N \cdot \log N)$ (wie günstigster Fall, wenn immer gleich große Teile enstehen)
- worst case: $O(N^2)$(wenn ein Randelement als Pivot gewählt wird)
- Stochastischer Algorithmus, wenn das Pivot durch ein Monte-Carlo-Verfahren ermittelt wird (siehe Determiniertheit)

# Die Türme von Hanoi

- Ursprung: tibetanisches Mönchskloster
- Aufgabe:
  - bewege den Turm nach B, so dass alle Scheiben wieder in der richtigen Reihenfolge liegen
  - Ringe dürfen nur einzeln bewegt werden
  - keinen großen auf einen kleinen Ring legen



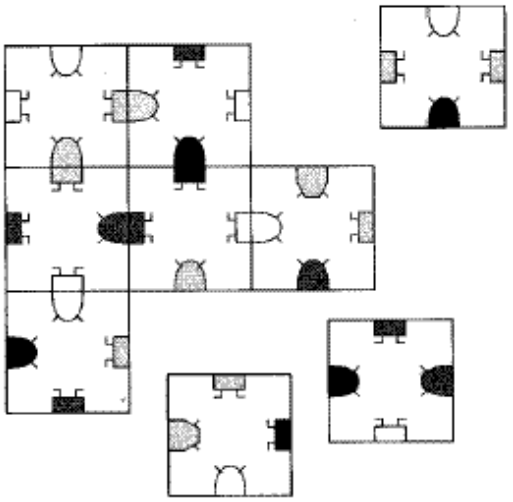
Warum behaupteten die Mönche, dass die Welt enden würde, bevor alle Türme auf B zu liegen kämen ?

# Universum leider zu kurzlebig

- … weil das ursprüngliche (tibetanische) Problem 64 Ringe auf A hatte
- Anzahl der Bewegungen $= 2^N - 1$ ($= 7$ für $N = 3$), d.h. $O(2^N)$
- Online:l http://www.cut-the-knot.com/recurrence/hanoi.shtml

- Exponentielle Zeitkomplexität !
  - Könnten die Mönche alle 10 s einen Ring bewegen, so würden sie 5 Trillionen Jahre brauchen, was tatsächlich nach dem derzeitigen Wissen die Dauer des Universums übersteigen würde
  - Selbst mit einem Computer, der pro Sekunde eine Million Ringe bewegt, würde es mehr als eine halbe Million Jahre dauern, ehe die Ringe richtig zu liegen kommen

# Monkey Puzzle



- Ziel des Spiels: ein 3x3-Quadrat legen, bei dem an jeder Schnittstelle ein vollständiger und einfarbiger Affe entsteht
- Algorithmus könnte nun alle möglichen  Anordnungen durchspielen.
  - Ist eine Anordnung zulässig, hält er mit der Ausgabe „ ja“
  - Sind alle möglichen Anordnungen durchprobiert und keine als zulässig erkannt wurde, hält er mit der Ausgabe nein an (Entscheidungsproblem)

# Erhöhe auf Fünf!

- Wie lange braucht der Computer für eine 5x5-Anordnung mit 25 Karten ??
  - Für die 1. Karte gibt es 25 Möglichkeiten, eine Karte auszuwählen und 4 Möglichkeiten der Lage der Karte
  - Für die 2. Karte auf der nächsten Position gibt es 24 Möglichkeiten und wieder 4 Möglichkeiten der Lage
  - Gesamtzahl: $(25 \cdot 4) \cdot (24 \cdot 4) \cdot (23 \cdot 4) \ldots (1 \cdot 4) = 25! \cdot 4^{25}$ Anordnungen
- Computer, der in einer Sekunde 1 Mio Anordnungen testet:
  - 533.000.000.000.000.000.000.000.000 Jahre (worst case)
  - (1 ns im günstigsten Fall, dies ist aber äußerst unwahrscheinlich)

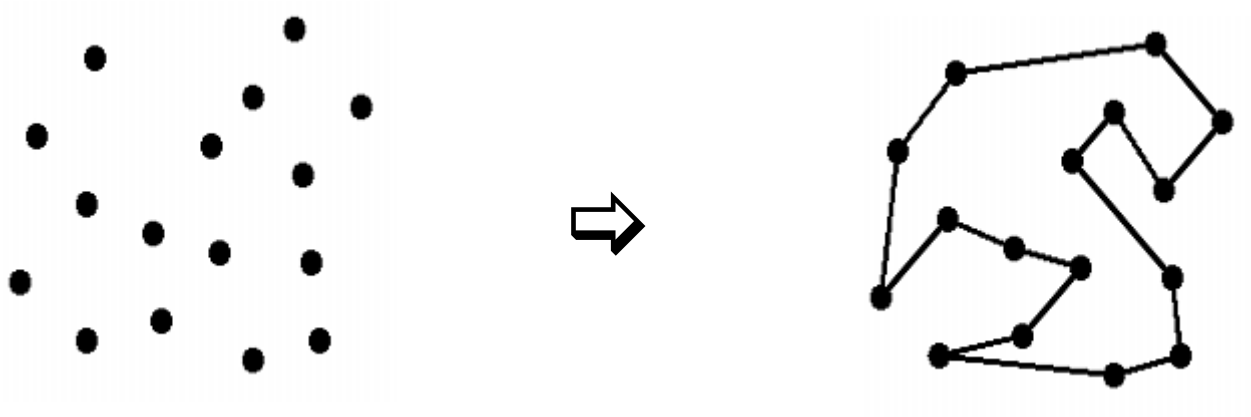
# Size does matter

- eine 5×5 - Fläche auszulegen scheint keine grosse Aufgabe: trotzdem ist sie es im dargelegten Fall
- Tabelle: Rechenzeit für Rechner mit 1 Mio einfache Anweisungen/s
- Selbst mit einem 10 000 mal schnelleren Computer würde man in vielen Fällen nicht viel weiter kommen
  - 102 statt 100 Eingabedaten

| | 10 | 20 | 50 | 100 | 200 |
|---|---|---|---|---|---|
| $N^2$ | $\frac{1}{10000}$ s | $\frac{1}{2500}$ s | $\frac{1}{400}$ s | $\frac{1}{100}$ s | $\frac{1}{25}$ s |
| $N^5$ | $\frac{1}{10}$ s | 3,2 Min. | 5,2 Min. | 2,8 h | 3,7 Tage |
| $2^N$ | $\frac{1}{1000}$ s | 1 s | 35,7 Jahre | über 400 Billionen Jahrhunderte | 45stellige Zahl an Jahrhunderten |
| $N^N$ | 2,8 h | 3,3 Billionen Jahre | 70stellige Zahl an Jahrhunderten | 185stellige Zahl an Jahrhunderten | 445stellige Zahl an Jahrhunderten |

Zum Vergleich: Der Urknall war vor ca. 12-15 Milliarden Jahren.

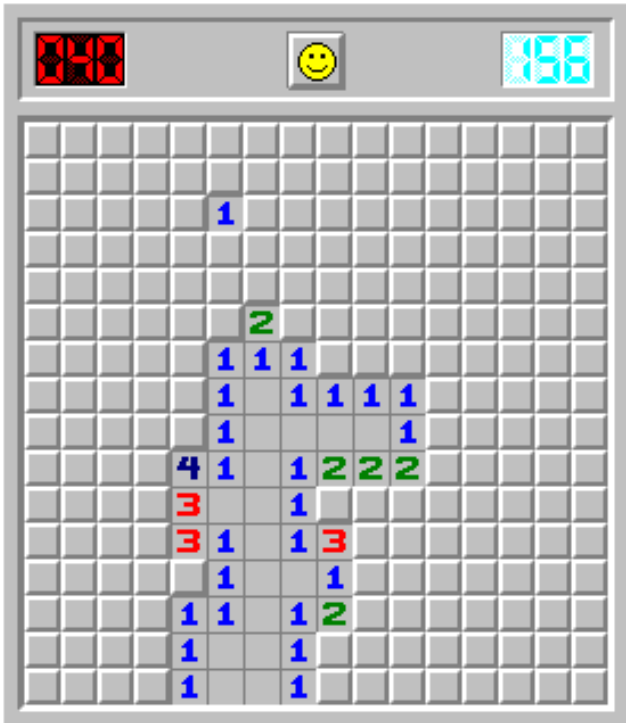
# Traveling Salesman (Handlungsreisender)



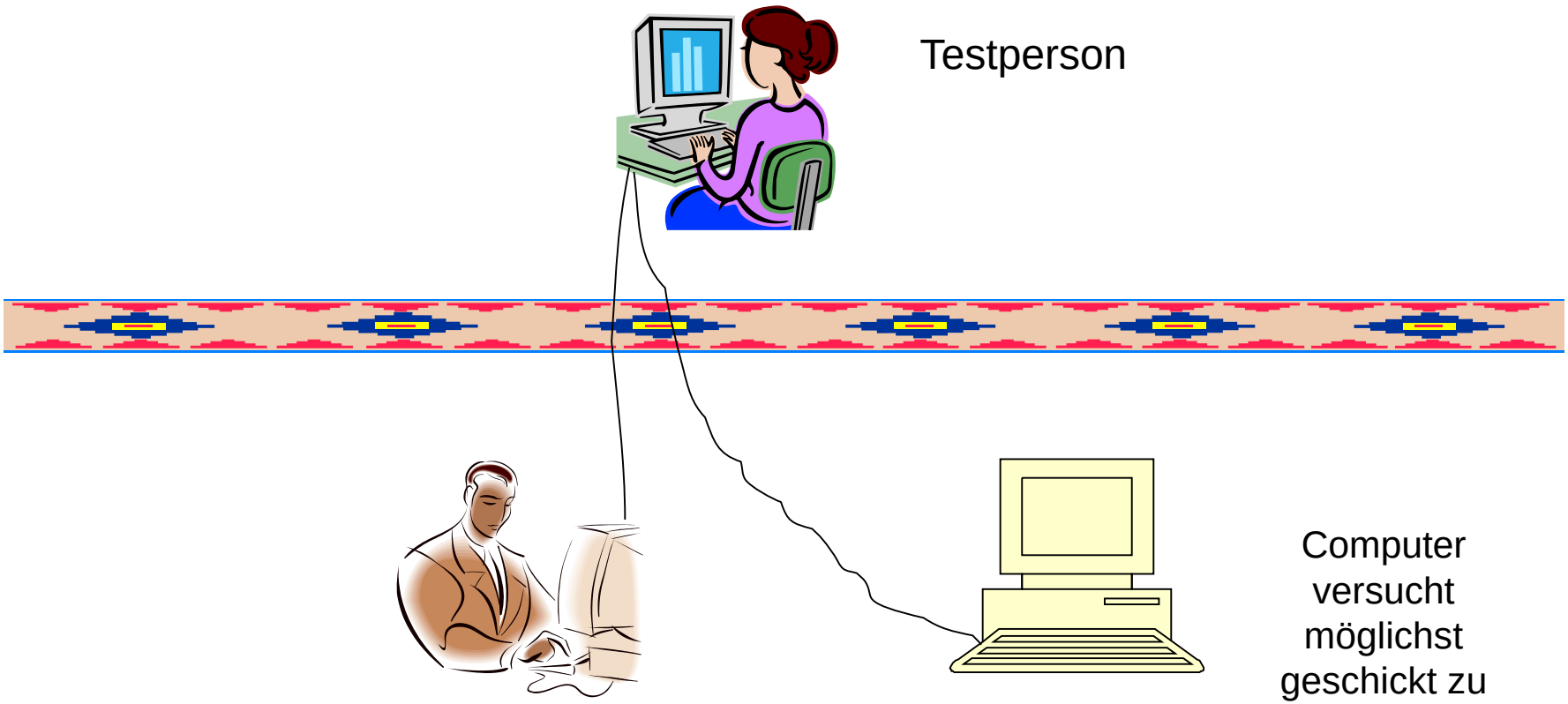
- Praxisbedeutung: Telefon-Netzwerke, Schaltkreise
- Komplexität $O(N!)$, NP-complete
- Zahl 25 z.B. bei Telefon-Netzwerken keinesfalls groß!
- Monte-Carlo-Methoden:
  - wähle zufällig so viele Routen wie möglich aus und nimm die, die nach einem Monat Rechenzeit die beste ist ⇨ starte Fuhrunternehmen
  - http://www.math.princeton.edu/tsp/

# Minesweeper

- Im Jahre 2000 wurde von RICHARD KAYE bewiesen, dass das Problem, für eine vorgelegte Minesweeper-Konstellation zu entscheiden, ob sie logisch konsistent ist, ein NP-Problem ist.



# 7. Computer vs. menschliche Intelligenz:
## Der Turing-Test



# Turing-Test II

- Turing sagte 1950 voraus, dass etwa im Jahre 2000 ein Prüfer nur noch eine Chance von 70% hätte, den Computer zu identifizieren
- Seit 1991: Loebner-Wettbewerb: Preisgeld für ein Programm, das drei von zehn Prüfern 5 Minuten täuschen kann.
- Aber: bisher wurden alle teilnehmenden Programme nach spätestens fünf Minuten entlarvt.
- Unerwartete Erkenntnisse: Mancher menschliche Teilnehmer ging mit der erschütternden Erkenntnis nach Hause, dass er aufgrund seiner langweiligen Antworten für einen Computer gehalten wurde.

# Turing-Test III

- Ist der Test unfair gegenüber dem Computer?:
  - wenn der Mensch vorgeben müsste, ein Computer zu sein, wäre es für die fragende Person ganz einfach, die richtige Zuordnung zu treffen: hinreichend komplizierte arithmetische Aufgabe
- Psychologisches Problem Mensch-Maschine-Kommunikation:
  - Joseph Weizenbaum (60iger Jahre): Programm Eliza simuliert das Interview-Verhalten eines Psychotherapeuten
  - Viele Benutzer nahmen Eliza für voll
  - http://www.unet.univie.ac.at/~a9725261/eliza.htm (englisch)
  - http://mzs.sowi.uni-goettingen.de/mitarbeitende/arnhold/tesmik/einfuehrung.html
  - http://www.allkuma.de/eliza.htm (sofort loschatten)
  - http://www.uni-ulm.de/~s_mbrusd/aturing.html (mehr zum Thema KI)

# Anhang

## Der Algorithmusbegriff

Uwe Menzel, 1998

uwe.menzel@matstat.de

www.matstat.org

quadrat.pl (locked)

File Edit Search Preferences Shell Macro Windows Help

/nfs/1d/menzel/ALGO/quadrat.pl line 18, col 40, 490 bytes

```
#!/usr/local/bin/perl
# uwe.menzel@genpat.uu.se
# Lösung einer quadratischen Gleichung

if ($#ARGV != 1 ) {
  print " \n Usage: quadrat.pl  <p>  <q>  \n";
  print "   Solves x**2 + p*x + q = 0 \n\n"; exit(1);
} else {
  $p = $ARGV[0];  $q = $ARGV[1];
}


$c = -$p/2;
$D = $c * $c - $q;
$w = sqrt(abs($D));

if ( $D < 0 ) {
  print "\n x1 = $c + i*$w     x2 = $c -i*$w  \n\n";
} else {
  $x1 = $c + $w; $x2 = $c - $w;
  print "\n x1 = $x1     x2 = $x2  \n\n";
}

exit 0;
```

# Terminierung bei der Bisektion

Für die Terminierung hat u.U. der Programmierer Sorge zu tragen:

**Residuums-Kriterium**: Programm terminiert, wenn der Funktionswert an der gefundenen Stelle klein genug ist:

$$f(x_n) \leq \varepsilon \quad \Rightarrow \quad n \text{ unbekannt}$$

**Fehler-Kriterium**: Programm terminiert, wenn das Intervall auf der Abszisse hinreichend genau eingegrenzt wurde:

$$\frac{b-a}{2^n} \leq \varepsilon \quad \Rightarrow \quad n \cong \log_2\left(\frac{b-a}{\varepsilon}\right)$$

Terminierung

bisec.for (locked)

File Edit Search Preferences Shell Macro Windows Help

/nfs/1d/menzel/ALGO/bisec.for line 5, col 0, 916 bytes

```
C     uwe.menzel@genpat.uu.se, Numerical Recipes / FORTRAN
C     Compiler: /usr/local/bin/g77
C     Root of func between x1 and x2.

      FUNCTION rtbis(func,x1,x2,xacc)
      INTEGER JMAX
      REAL rtbis,x1,x2,xacc,func
      EXTERNAL func
C     Maximum allowed number of bisections.
      PARAMETER (JMAX=40)
      INTEGER j
      REAL dx,f,fmid,xmid

      fmid=func(x2)
      f=func(x1)

      if(f*fmid.ge.0.) pause 'root must be bracketed in rtbis'

      if(f.lt.0.) then
C     Orient the search so that f>0 lies at x+dx.
        rtbis=x1
        dx=x2-x1
      else
        rtbis=x2
        dx=x1-x2
      endif

      do j=1,JMAX
C       Bisection loop.
        dx=dx*.5
        xmid=rtbis+dx
        fmid=func(xmid)
        if (fmid.le.0.) rtbis=xmid
        if (abs(dx).lt.xacc .or. fmid.eq.0.) return
      enddo
      pause 'too many bisections in rtbis'
      END
```

endlos.c

File Edit Search Preferences Shell Macro Windows Help

:l/JENA/Csource/endlos.c line 18, col 10, 258 bytes

```c
#include <stdio.h>
#include <stdlib.h>
/* compiler:  gcc -lm -o endlos endlos.c */

main(int argc, char *argv[])

{
  int i;
  i = atoi(argv[1]);
  printf ("i is %i\n", i);

  while (i != 0)
  {
    i = i - 2;
    printf ("i is %i\n", i);
  }

return(0);
}
```

goldbach.pl (locked)

File Edit Search Preferences Shell Macro Windows Help

/nfs/1d/menzel/bin/goldbach.pl line 1, col 0, 768 bytes

```
#!/usr/local/bin/perl
# uwe.menzel@genpat.uu.se
# Algorithmus: Test auf Goldbach-Eigenschaft

$n = $ARGV[0];

if (($n < 4) || ($n % 2)) {
  print " \n INPUT ERROR: Input must be even and >= 4 \n\n";exit 2;}

for ( $i=1; $i<=$n/2; $i++ ) {
  if ( &prime($i) && &prime($n-$i) ) {
    print "\n $n meets Goldbach assumption.\n\n";exit 0;}
}

print "\n $n DOES NOT meet the Goldbach assumption.\n";
print "Contact the Royal Society of Mathematics immediately.\n\n";
exit 0;


sub prime {
  my $n = $_[0]; my $i;

  if ($n == 2) {return 1;}  # return with 1 if prime, otherwise with 0

  #  n is a prime if no divider exists between 2 and sqrt of n

  for ($i=2; $i<=sqrt($n); $i++) {
    if ( ($n % $i) == 0 ) {return 0;}
  }

  return 1;
}
```

wundersam.pl (locked)

File Edit Search Preferences Shell Macro Windows Help

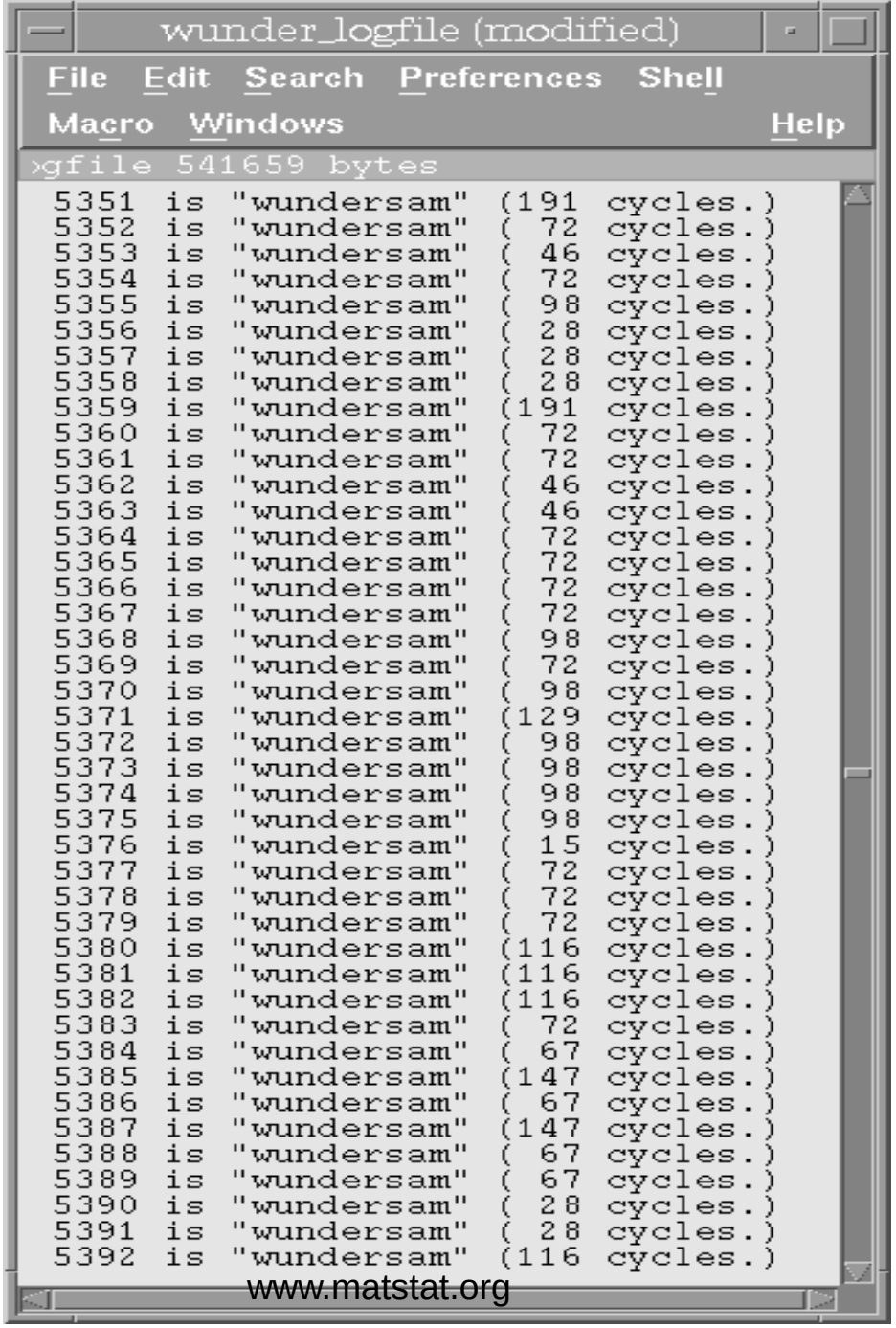
/s/1d/menzel/ALGO/wundersam.pl line 4, col 0, 321 bytes

```
#!/usr/local/bin/perl
# uwe.menzel@genpat.uu.se
# decides if a number is wundersam

$n = $ARGV[0];

$i = 0; # loop counter
while ($n > 1) {
   $i++;
   if ($n % 2) { $n = 3 * $n + 1} else { $n = $n / 2}
   print " $n\n";
   if (!($i / 1000)) { print " $i cycles done.\n";}
}

print " \n wundersam ($i cycles.)\n\n";
exit 0;
```


wunder_logfile (modified)
File Edit Search Preferences Shell
Macro Windows Help
ogfile 541659 bytes
5351 is "wundersam" (191 cycles.)
5352 is "wundersam" ( 72 cycles.)
5353 is "wundersam" ( 46 cycles.)
5354 is "wundersam" ( 72 cycles.)
5355 is "wundersam" ( 98 cycles.)
5356 is "wundersam" ( 28 cycles.)
5357 is "wundersam" ( 28 cycles.)
5358 is "wundersam" ( 28 cycles.)
5359 is "wundersam" (191 cycles.)
5360 is "wundersam" ( 72 cycles.)
5361 is "wundersam" ( 72 cycles.)
5362 is "wundersam" ( 46 cycles.)
5363 is "wundersam" ( 46 cycles.)
5364 is "wundersam" ( 72 cycles.)
5365 is "wundersam" ( 72 cycles.)
5366 is "wundersam" ( 72 cycles.)
5367 is "wundersam" ( 72 cycles.)
5368 is "wundersam" ( 98 cycles.)
5369 is "wundersam" ( 72 cycles.)
5370 is "wundersam" ( 98 cycles.)
5371 is "wundersam" (129 cycles.)
5372 is "wundersam" ( 98 cycles.)
5373 is "wundersam" ( 98 cycles.)
5374 is "wundersam" ( 98 cycles.)
5375 is "wundersam" ( 98 cycles.)
5376 is "wundersam" ( 15 cycles.)
5377 is "wundersam" ( 72 cycles.)
5378 is "wundersam" ( 72 cycles.)
5379 is "wundersam" ( 72 cycles.)
5380 is "wundersam" (116 cycles.)
5381 is "wundersam" (116 cycles.)
5382 is "wundersam" (116 cycles.)
5383 is "wundersam" ( 72 cycles.)
5384 is "wundersam" ( 67 cycles.)
5385 is "wundersam" (147 cycles.)
5386 is "wundersam" ( 67 cycles.)
5387 is "wundersam" (147 cycles.)
5388 is "wundersam" ( 67 cycles.)
5389 is "wundersam" ( 67 cycles.)
5390 is "wundersam" ( 28 cycles.)
5391 is "wundersam" ( 28 cycles.)
5392 is "wundersam" (116 cycles.)

Wundersam

linsearch.pl (locked)

File Edit Search Preferences Shell Macro Windows Help

/nfs/1d/menzel/ALGO/linsearch.pl line 14, col 9, 501 bytes

```
#!/usr/local/bin/perl
# uwe.menzel@genpat.uu.se
# naive algorithm for searching a list

if ($#ARGV != 1 ) {
  print " \n Usage: linsearch.pl  <list>  <name>  \n";
  print "   Searches for <name> in <list>\n\n"; exit(1);
} else {
  $list = $ARGV[0];  $name = $ARGV[1];
}


open(LIST,$list) || die " Cannot open file '$list' \n\n";
@array=<LIST>;


foreach $entry (@array) {
  chomp($entry);
  if ("$name" eq "$entry") {
    print " $list contains $name\n\n";last;}
}


close(LIST);
exit 0;
```

# NP-vollständige Probleme

- Klasse von ca. 2000 verschiedenen algorithmischen Problemen, die sich alle in Hinsicht auf Komplexität ähneln:
  - Sie sind entscheidbar (also prinzipiell lösbar)
  - Zeitkomplexität ist allerdings exponentiell
  - Für kein Problem wurde je Polynomialzeit-Algorithmus gefunden
  - Niemand konnte bisher beweisen, dass sie exponentielle Zeit benötigen müssen
- In gewissem Sinn sind alle diese Probleme verwandt:
  - Sollte jemals für ein einziges Problem ein Algorithmus mit Polynomialzeit gefunden werden, dann ergäben sich sofort Polynomialzeit-Algorithmen für alle anderen Probleme.
  - Sollte jemals für eines der Probleme gezeigt werden, dass es keinen polynomialen Algorithmus geben kann, dann ist damit automatisch dasselbe für alle anderen Probleme bewiesen

# Ein scheinbar „vollkommen“ einfaches Problem

- … die vollkommenen Zahlen: Zahlen, die gleich der Summe ihrer Teiler sind:
  - 6 = 1 + 2 + 3
  - 28 = 1 + 2 + 4 + 7 + 14
- bisher nur 30 vollkommene Zahlen bekannt (die größte hat 840.000 Stellen)
  - Behauptung 1: Alle vollkommenen Zahlen sind gerade
  - Behauptung 2: Es gibt unendlich viele vollkommene Zahlen
- Beide Behauptungen konnten bis heute weder bewiesen noch widerlegt werden.

# Programm vs. Algorithmus

- Ein Programm ist ein in einer bestimmtem Programmiersprache formulierter Algorithmus.
- Der Algorithmus ist somit allgemeiner als das Programm.

| | |
|---|---|
| FORTRAN | Numerische Berechnungen, komplexe Arithmetik, Bibliotheken |
| C/C++ | portable, vielseitig, weit verbreitet |
| Mathcad/Maple | Vielfalt mathematischer Probleme, hübsche Benutzeroberfläche |
| PERL | Textbearbeitung, Parsen, Internet- und Datenbankanbindungen |