Informatik-Propädeutikum

Dozent: Prof. Rolf Niedermeier

Betreuer: Sepp Hartung, André Nichterlein, Jiehua Chen
Sekretariat: Christlinde Thielcke (TEL 509b)

TU Berlin
Institut für Softwaretechnik und Theoretische Informatik
Fachgruppe Algorithmik und Komplexitätstheorie
http://www.akt.tu-berlin.de

Wintersemester 2012/2013

Organisation

Für aktuelle Informationen, Übungsmaterial etc. bitte über ISIS anmelden!

- Vorlesung: Dienstag 16-18 Uhr, HE 101
- (freiwilliges!) Großtutorium: Mittwoch 16-18 Uhr, H 3010

Prüfungsleistung: Schriftliche Prüfung mit der Vorleistung des Bestehens (50% der Punkte) von regelmäßigen Online-Testaten bezogen auf den Lehrstoff (inkl. Aufgabenblätter).

Literatur: Verschiedene Quellen, z.B.:

- Hal Abelson, Ken Leeden, Harry Lewis: Blown to Bits, 2008.
- Tim Bell, Ian H. Witen, Mike Fellows: Computer Science Unplugged, 1998.
- Jens Gallenbacher: Abenteuer Informatik, 3. Auflage, 2012.
- Christopher Moore, Stephan Mertens: The Nature of Computation, 2011.
- Uwe Schöning: Ideen der Informatik, 3. Auflage, 2008.

Motivation – Worum geht's?

Ziel: Ideengetriebene, leicht verständliche Darstellung von elementaren Grundlagen, Anwendungen und Perspektiven der Informatik. „Rundumschlag".

Einige zentral behandelte Ideen (= Konzepte, Modelle, Algorithmen, Beschreibungsmethoden, Protokolle, Vorgehensweisen, ...):

- Abstraktion.
- Rekursion.
- Effizienz.
- Information.
- Kodierung und Verschlüsselung.
- Verteiltheit und Nebenläufigkeit.
- Zufall.
- Informatik in Anwendungen.
- ...

Gliederung

Einführung: Was ist ein Propädeutikum?

Siehe Wikipedia:
Das Propädeutikum (von Griechisch propaideuein „im Voraus unterrichten“, von pro „vor“ und paideuein „lehren, unterrichten“) ist meist eine Vorbereitungsveranstaltung auf ein wissenschaftliches Gebiet. Der Begriff wird oft bedeutungsgleich mit Einführungsveranstaltung verwendet; ...

Aus der Modulbeschreibung zur Vorlesung:
... **Einblick in grundlegende wissenschaftliche Fragestellungen und Phänomene der Informatik**[1] ... historische Entwicklung der Informatik ... Kompetenzprofil ... Informatikanwendungsfelder ... Abgrenzung von anderen Wissenschaftsgebieten ... interdisziplinäre Anforderungen an Informatikarbeit ... gesellschaftliche Relevanz... kritische Hinterfragung von Informatikanwendungen und -methoden....

[1] Unser Schwerpunkt hier!

Was ist Informatik?[2]

Wikipedia: Informatik ist die „Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Digitalrechnern“.

Oder: Informatik ist die *Faszination*, sich die Welt der Information und des symbolisierten Wissens zu erschließen und dienstbar zu machen.

Ein Zitat, das oft Edsger W. Dijkstra, dem Wegbereiter der strukturierten Programmierung, zugeschrieben wird, wohl aber Folklore ist:

„In der Informatik geht es genau so wenig um Computer, wie in der Astronomie um Teleskope.“

Edsger W. Dijkstra, 1930–2002

[2] Vergleiche auch das gleichnamige Positionspapier der Gesellschaft für Informatik e.V.

Einordnung in die Wissenschaftslandschaft

Informatik = Information und Automatik (bzw. Information und Mathematik). Im Englischen „Computer Science“; dort wird „Informatics“ eher für bestimmte Teile der angewandten Informatik verwendet (z.B. „Bioinformatics“).

Historisch hat sich die Informatik einerseits als Formalwissenschaft aus der **Mathematik** entwickelt, andererseits als **Ingenieursdisziplin** aus dem praktischen Bedarf nach einer schnellen und insbesondere automatischen Ausführung von Berechnungen.

Informatik ist
- Grundlagenwissenschaft und
- Ingenieurwissenschaft und auch
- Experimentalwissenschaft ($\rightsquigarrow$ Simulationen).

Informatik ist wie die Mathematik eine auf alle anderen Wissensgebiete ausstrahlende Grundlagenwissenschaft; steht bei der Mathematik mehr das „formal Denkbare“ im Vordergrund, so in der Informatik das „Realisierbare“ ($\rightsquigarrow$ konstruktiver Aspekt).

Säulen der Informatik

Klassischer Weise werden die folgenden vier Teilbereiche der Informatik unterschieden, wobei die Grenzen fließend sind:

- Theoretische Informatik;
- Technische Informatik;
- Praktische Informatik;
- Angewandte Informatik.

Die vielfältigen Anwendungsgebiete der Informatik

Informatik besitzt eine ungewöhnliche Breite. Wie die Mathematik ist sie eine Querschnittswissenschaft.

Zusammenwachsende Wissenschaften:

- Bioinformatik
- Geoinformatik
- Medieninformatik
- Medizininformatik
- Wirtschaftsinformatik
- ...

Charakteristisch ist für Informatiker das Arbeiten im **Team** mit Anwendern und Fachleuten **anderer Disziplinen**.

(Kurz-)Geschichtliches

Wichtige Ereignisse für die Informatikgeschichte:

- **Kerbhölzer** als Zähl- und Rechenhilfe vor 30000 Jahren.
- **Abakus** der Chinesen und Römer (seit über 3000 Jahren).
- **Euklid'scher Algorithmus** (ca. 300 vor Chr.): Bestimmung des größten gemeinsamen Teilers zweier ganzer Zahlen. Euklid Autor des einflussreichsten Mathematikbuchs aller Zeiten: „Elemente" (Geometrie, Zahlentheorie); spezieller Wesenszug des Buchs: konstruktive (!) Existenzbeweise.
- **Sieb des Eratosthenes** (284–202 vor Chr.; geboren in Kyrene (heute Libyen), gestorben in Alexandria): Algorithmus zur Bestimmung aller Primzahlen bis zu einer vorgegebenen Maximalzahl.
- **Heron** von Alexandria (100 nach Christus): Verfahren zum Wurzelziehen (lange zuvor schon bei den Babyloniern...)
- **Chinesischer Restsatz** nach Sun Zi (ca. 200 nach Chr.): $\rightsquigarrow$ Algorithmus zum Lösen modularer Gleichungssysteme.

(Kurz-)Geschichtliches II

- **Abu Ja'fer Mohammed Ibn Musa Al-Khowarizmi** (ca. 780–850; Persien-Arabien): Verfasser eines sehr einflussreichen **Rechenbuchs**[3] (Dezimalsystem mit Null); abgeleitete lateinische Sprachfloskel „Dixit Algorizmi" („Also Sprach Al-Khowarizmi") als Gütezeichen für die Richtigkeit einer Rechnung $\rightsquigarrow$ Begriff **Algorithmus**.

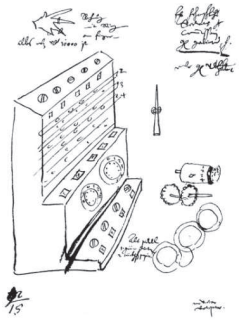
Al-Khowarizmi auf einer sowjetischen Erinnerungsbriefmarke (1983)

[3] „Rechnen auf der Linie" (1518) von Adam Ries in weiten Teilen eine „Übersetzung" von Al-Khowarizmis Buch.

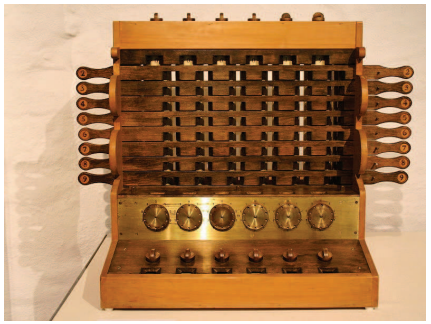
(Kurz-)Geschichtliches III

Wilhelm Schickard (1592–1635): Rechenmaschine mit Zahnradgetriebe zur Erleichterung astronomischer Rechnungen; Addieren und Subtrahieren von sechsstelligen Zahlen.

- 



Originalzeichnung von Schickard



Nachbau der Rechenmaschine

(Kurz-)Geschichtliches IV

- **Blaise Pascal** (1623–1662): „Pascaline“-Rechenmaschine (für seinen Vater, einen hohen Steuerbeamten) zum Addieren und Subtrahieren. Ca. 50 Exemplare wurden wirklich gebaut.

Eine Pascaline aus dem Jahr 1652

Blaise Pascal

Niklaus Wirth (ETH Zürich) benannte nach ihm die Programmiersprache Pascal, die zur Lehre des strukturierten Programmierens dient(e) (gut prüfbarer und wartbarer Code).

(Kurz-)Geschichtliches V

- Weitere Rechenmaschinen von Gottfried Wilhelm Leibniz (1646–1716), Giovanni Poleni (1683–1761), Antonius Braun (1686–1728), Jacob Leupold (1674–1727), ...

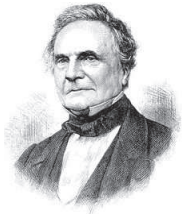
Wesentlich Einschränkung aller obigen Rechenmaschinen:
Von Erbauung an klar festgelegt, was sie tun sollten, sprich festgelegte Funktionalität; keine Trennung von „Hardware“ und „Software“.

Dies änderte sich im 19. Jahrhundert durch die Vision von Charles Babbage: Eine **programmierbare** und damit „universelle“ Rechenmaschine.

Zitat: „As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise — by what course of calculation can these results be arrived at by the machine in the shortest time?“
(Babbage (1864) Passages from the Life of a Philosopher)

(Kurz-)Geschichtliches VI

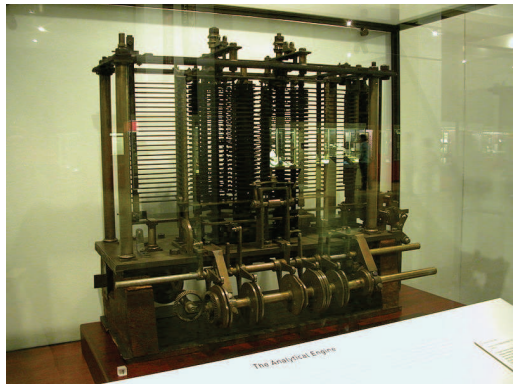
- **Analytical Engine** von **Charles Babbage** (1791–1871) unter Mitarbeit von **Ada Lovelace** (1815–1852) ($\rightsquigarrow$ Namensgebung der Programmiersprache Ada ihr zu Ehren): Als mechanische Rechenmaschine für allgemeine Anwendungen (universell programmierbar!) Vorläufer des modernen Computers; Ada Lovelace beschrieb die Programmierung der Maschine (u.a. Konzept der Schleifen) in der Theorie und gilt daher als erste Programmiererin.



Charles Babbage

Ada Lovelace

(Kurz-)Geschichtliches VII



Versuchsmodell einer Analytical Engine

Geplant: Dampfmaschinenantrieb, Lochkarteneingabe (ähnlich wie bei schon bekannten mechanischen Webstühlen), Speicher für 1000 Wörter mit bis 50 Dezimalstellen (ca. 20,7 kB); vier Grundrechenarten. Programmiersprache ähnlich Assemblersprachen.

(Kurz-)Geschichtliches VIII

- **Konrad Zuse** (1910–1995), Bauingenieur-Alumnus der TU Berlin: Z1 (1938; mechanische Schalter); Z3 aus dem Jahre 1941 erster vollautomatischer, programmgesteuerter und frei programmierbarer Digitalrechner der Welt (basierend auf elektromagnetischer Relaistechnik mit ca. 2200 Relais).

Konrad Zuse

(Kurz-)Geschichtliches IX

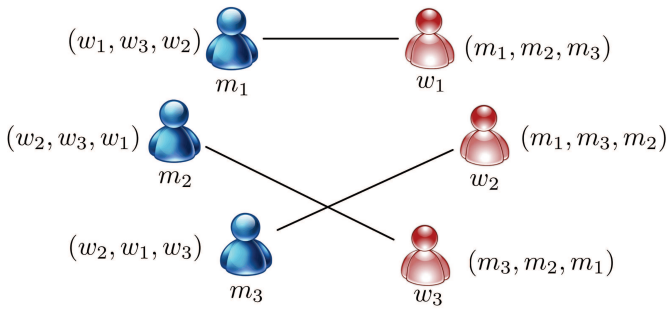
- ENIAC (Electronic Numerical Integrator and Computer): 1946 an der Universität Pennsylvania: Computer auf Basis von Elektronenröhren.
- UNIVAC (Universal Automatic Computer): wirtschaftlich überlebensfähige Weiterentwicklung der ENIAC. Einsatz bei Volkszählung. Aufgrund des Erfolgs stieg nun IBM in das Geschäft mit universellen Rechenmaschinen ein...
- 1956 erster Computer auf Transistorbasis (statt Elektronenröhren)
- ...

Abschließende Bemerkung: Die ersten Programmiersprachen waren sehr maschinennah und wenige Menschen beherrschten die Programmierung.
$\rightsquigarrow$ Programmiererknappheit
$\rightsquigarrow$ Entwicklung von Hochsprachen zur Programmierung: FORTRAN, COBOL, ALGOL (58, 60, 68, W), ...;
$\rightsquigarrow$ Beginn der Dominanz der Software (Algorithmen) gegenüber der Hardware in der Informatik.

Algorithmen aktuell: Wirtschafts-Nobelpreis 2012

Lloyd Shapley (1923–) gewann 2012 zusammen mit Alvin E. Roth den Wirtschafts-Nobelpreis 2012 für die Theorie stabiler Zuweisungen und den Entwurf praktischer Marktmechanismen.

Ein wesentlicher Aspekt hierbei ist der Gale-Shapley-Algorithmus für „Stable Matching“ (auch bekannt als „Stable Marriage“)...



Stable Matching (Gale und Shapley)

Input: $M = \{m_1, \ldots, m_n\}$ („Männer“)
$W = \{w_1, \ldots, w_n\}$ („Frauen“)
jedes $m \in M$ ordnet alle Elemente aus W nach Präferenz
jedes $w \in W$ ordnet alle Elemente aus M nach Präferenz

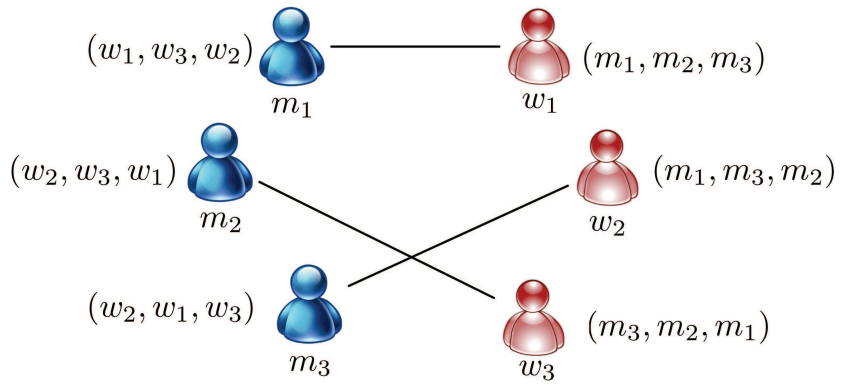
Aufgabe: Finde paarweise Zuordnung zwischen den Elementen aus M und W, so dass für jeden $m \in M$ und jede $w \in W$ die nicht m zugeordnet ist gilt:

1. m zieht ihm zugeordnete w' gegenüber w vor, oder
2. w zieht ihr zugeordneten m' gegenüber m vor.

$\rightsquigarrow$ „**stabile Zuordnung**“

Stable Matching: Beispiel 1

Eine stabile Zuordnung:



Stable Matching: Beispiel 2

Bsp.: $M = \{X, Y, Z\}$, $W = \{A, B, C\}$

$X : A < B < C$	$A : Y < X < Z$
$Y : B < A < C$	$B : X < Y < Z$
$Z : A < B < C$	$C : X < Y < Z$

Test 1: Zuordnung $(X, C), (Y, B), (Z, A)$ stabil?

Test 2: Zuordnung $(X, A), (Y, B), (Z, C)$ stabil?

Effizienter Algorithmus für Stable Matching

Propose&Reject [Gale, Shapley 1962]

1 Initialisiere alle $m \in M$ und alle $w \in W$ als „frei“
2 **while** $\exists m \in M$: m ist frei und $\exists w \in W$ der m noch keinen Antrag gemacht hat
3 $w \leftarrow$ erste noch „unbeantragte“ Frau in m's Präferenzfolge
4 **if** w ist frei **then:**
5 (m, w) wird Paar, $m \leftarrow$„verlobt“, $w \leftarrow$„verlobt“
6 **else if** w zieht m ihrem aktuellen „Verlobten“ m' vor **then:**
7 (m, w) wird Paar, $m \leftarrow$„verlobt“, $w \leftarrow$„verlobt“, $m' \leftarrow$„frei“
8 **else:** w lehnt m ab

Stable Matching: Eigenschaften von Propose&Reject

Mitteilung: Propose&Reject findet immer ein „**perfektes Matching**“ (d.h. bei n Männern und n Frauen werden genau n Paare gebildet), das **stabil** ist, und benötigt dazu $\leq n^2$ **Durchläufe** der while-Schleife.

Mitteilung: Propose&Reject liefert immer eine **„männeroptimale“** Zuordnung, die zudem für Frauen schlechtestmöglich ist.

Fazit: Es lohnt sich, kluge Algorithmen zu entwerfen und sie zu analysieren!

Bemerkungen:

- Viele Problemvarianten: Mehrfachzuordnungen; Unisex; Präferenzlisten ohne totale Ordnung, ...
- (Stable) Matching Probleme haben sehr viele Anwendungen und füllen (zusammen mit ihren Lösungsmethoden) Bücher.

Abstraktion

Franz Marc (1880–1916), Kämpfende Formen

Abstraktion

Henri Matisse (1869–1954), Nackte Blaue

Abstraktion

Wikipedia: Das Wort Abstraktion (lat. abstractus – „abgezogen“, Partizip Perfekt Passiv von abs-trahere – „abziehen, entfernen, trennen“) bezeichnet meist den induktiven Denkprozess des **Weglassens von Einzelheiten** und des **Überführens auf etwas Allgemeineres oder Einfacheres**.

Beispiel:

Karikaturen abstrahieren!

Autos und Abstraktion

Ein klassisches Beispiel für Abstraktion: **Nachdenken über Autos**.

Für die Schrauber: Bei der Reparatur eines Autos denkt man es sich (zunächst) nicht bestehend aus Schrauben, Muttern, Kabel, Gummi, Plastik, ..., sondern bestehend aus **Komponenten** wie Motor, Kraftstoffeinspritzung, Bremsen, Filter, Batterie, etc.
Das ist Abstraktion!

Für die Fahrer: In den Frühtagen des Automobils mussten die Fahrer zugleich nahezu Mechaniker sein. Später bedurften Autos zumindest auch des gekonnten Umgangs mit Kupplung und Gangschaltung; Autos mit Automatikgetriebe erfordern i.w. nur noch die Kenntnis von Gas und Bremse...
Das ist Abstraktion! (Nun kann jede(r) fahren(?)!)

Wichtig: **Schnittstellen** wie Gas- und Bremspedal, Lenkrad, ...

Abstraktion: London Underground

London Underground Übersicht 1928:

London Underground Übersicht 1933 (dank Harry Beck):

Zitate zur Abstraktion

„Eigentlich sollte man nämlich sagen: 'von etwas abstrahieren', nicht 'etwas abstrahieren'". – Immanuel **Kant**, Von der Form der Sinnes- und Verstandeswelt und ihren Gründen.

„Geld ist Abstraktion in Aktion. Wert hin, Wert her, Geschäft bleibt Geschäft. Dem Geld ist alles egal. Es ist das Medium, in dem die Gleichsetzung des Verschiedenen sich praktisch verwirklicht. Wie nichts anderes besitzt es die Kraft, Verschiedenes auf den gleichen Nenner zu bringen." – Peter **Sloterdijk**, Kritik der zynischen Vernunft Bd. 2

„Viele Menschen sind unglücklich, weil sie nicht abstrahieren können." – Immanuel **Kant**, Anthropologie in pragmatischer Hinsicht.

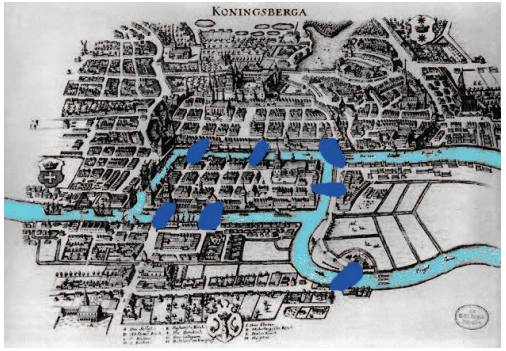
„This idea that there is generality in the specific is of far-reaching importance." – Douglas R. **Hofstadter**, Gödel, Escher, Bach: An Eternal Golden Braid.

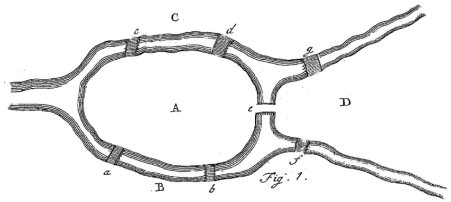
Wozu Abstraktion?

Vorteile des Abstrahierens:

- **Modellierung** und damit Abstraktion ist eine der zentralsten Informatiktechniken und unabdingbar für die Abbildung der realen Welt auf den Computer / die Software.
- Rechnen (Computing) ist letztlich nichts anderes als das Konstruieren, Manipulieren (Verarbeiten) und Folgern auf Basis von Abstraktionen.
- Mit Abstraktion lässt sich **Komplexität (besser) beherrschen**, z.B. komplexe Softwaresysteme.
- Der Umgang mit **mehreren Abstraktionsstufen** ist unerlässlich insbesondere beim Erstellen großer Softwaresysteme.
- Abstrahieren hilft beim Verallgemeinern von Lösungsansätzen und der Mehrfachverwendung von Softwarelösungen (vgl. Programmbibliotheken).
- Abstrahieren schärft den Blick aufs Wesentliche und macht damit erst den Weg für manche Lösungsstrategie frei.

Ein Abstraktionsklassiker: Königsberger Brückenproblem I

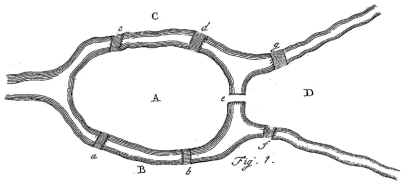


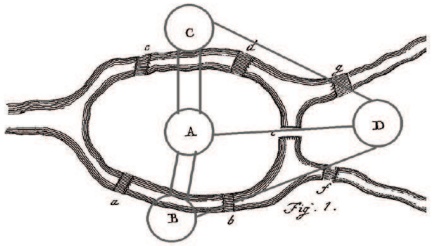


Im preußischen Königsberg gibt es eine Insel A, genannt der Kneiphof, und der Fluss, der sie umfließt, teilt sich in zwei Arme. Über die Arme dieses Flusses führen sieben Brücken a, b, c, d, e, f und g. Nun galt es die Frage zu lösen, ob jemand seinen Spazierweg so einrichten könne, dass er jede dieser Brücken einmal und nicht mehr als einmal überschreite.

– Leonhard Euler, 1707–1783

Eulers Königsberger Brückenproblem II





Beobachtungen:

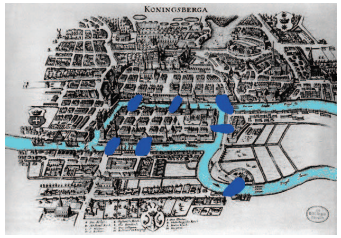
- Größen und Rechnen mit Größen irrelevant!
- Eigenschaften bzgl. Lagen und Beziehungen wichtig!

Modellierung:

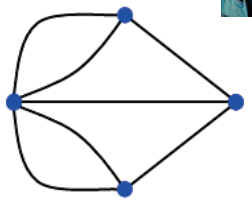
- Orte A, B, C, und D $\hat{=}$ Knoten;
- Brücken $\hat{=}$ Kanten.

Gesucht: Ein Weg durch alle Knoten so dass jede Kante genau einmal besucht wird!

Eulers Königsberger Brückenproblem III



$\Rightarrow$



Für einen zusammenhängenden (Multi-)Graphen gilt:

Satz: Ein zusammenhängender Graph enthält einen **Euler-Kreis** genau dann wenn jeder Knoten geraden Grad (Anzahl anliegender Kanten) hat; falls genau zwei Knoten ungeraden Grad haben, so hat er einen **Euler-Pfad** aber keinen Euler-Kreis.

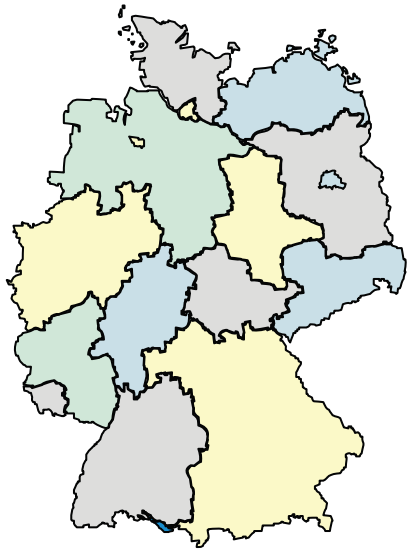
Eulers Königsberger Brückenproblem IV

Rückblick:

- Mit schneller Rechentechnik hätte man ohne Nachzudenken einfach alle möglichen Permutationen der Brücken (Kanten) überprüfen können?! **Ja, aber** bei m Kanten sind das $m!$ Möglichkeiten. Bei wachsendem m wächst $m!$ exponentiell schnell und kann schon für $m = 100$ nicht mehr berechnet werden.
- Eulers Lösung hingegen führt zu einem einfachen und effizienten Algorithmus, dessen Laufzeit nur linear in der Anzahl der Kanten wächst: Gehe alle Knoten durch und bestimme ihren Grad. Dies läuft in Sekundenschnelle selbst auf Graphen mit Milliarden von Kanten.

Bemerkung: In aktueller Forschung beschäftigen wir uns damit, Graphen durch Hinzufügen möglichst weniger Kanten „eulersch“ zu machen. Das könnte einmal der Berliner Stadtreinigung helfen...

Noch ein Abstraktionsklassiker: Landkartenfärben I

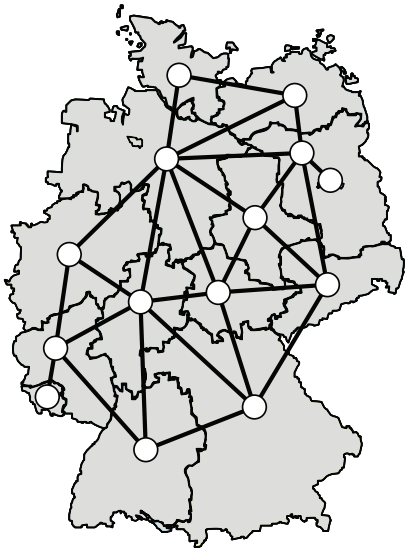


Aufgabe: Färbe Karte sodass zwei benachbarte Bundesländer unterschiedliche Farbe haben.

Triviale Lösung: Eigene Farbe für jedes Bundesland.

Ziel: Verwende möglichst wenige Farben!

Landkartenfärben II

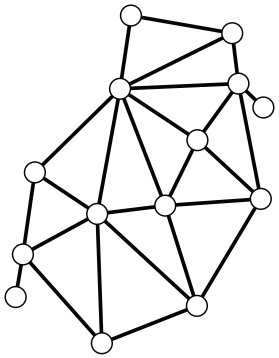


Beobachtung: Genaue Form der Bundesländer nicht wichtig, nur Nachbarschaftsrelation entscheidend.

Modellierung als Graphproblem:
Bundesländer $\hat{=}$ Knoten
Nachbarschaft $\hat{=}$ Kanten

Landkartenfärben III

Landkartenfärben ist ein Knotenfärbungsproblem in Graphen:



Problem: Färbe Knoten des Graphen sodass benachbarte („adjazente“) Knoten unterschiedliche Farben haben.

Beobachtung: Form der Bundesländer ist zusammenhängend $\rightsquigarrow$ Kanten kreuzen sich nicht, dies ergibt **planare Graphen**.

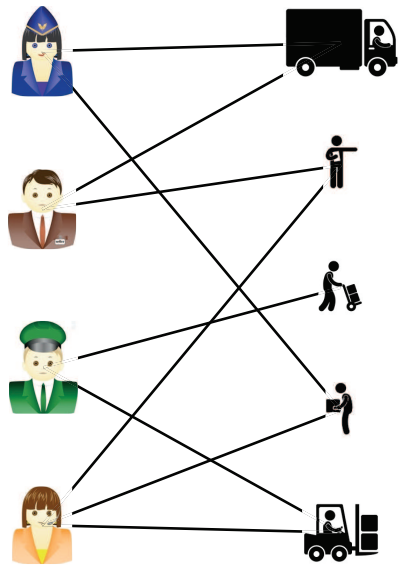
Vier-Farben-Satz: Jeder planare Graph kann mit vier Farben gefärbt werden.
Aufwändiger Beweis: Computergestützt! [Appel & Haken, 1976]

Zuordnungsproblem I

Problem: Unternehmer hat Mitarbeiter $m_1, m_2, \ldots, m_t$ und Aufträge $a_1, a_2, \ldots, a_s$ und weiß ob Mitarbeiter m_i die Qualifikation hat Aufgabe a_j zu erfüllen.

Ziel: Finde Zuweisung von Aufgaben an Mitarbeiter, sodass möglichst viele Aufträge erfüllt werden.

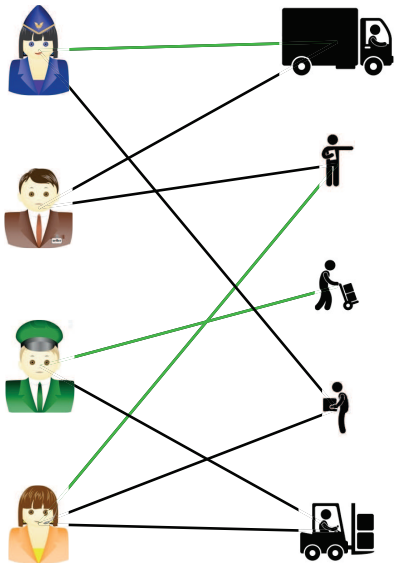
Annahme: Jeder Mitarbeiter kann nur einen Auftrag erfüllen und es braucht auch nur einen Mitarbeiter pro Auftrag.



Zuordnungsproblem II

Bipartites Graphproblem: Finde eine größtmögliche Kantenteilmenge M sodass kein Knoten Endpunkt von zwei oder mehr Kanten in M ist.

$\rightsquigarrow$ „**Bipartites Matching**“



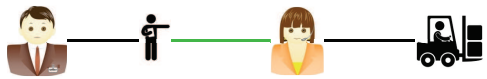
Zuordnungsproblem III

Lösungsansatz (Algorithmus) vermöge folgender Beobachtung:

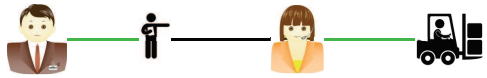
Satz [Berge 1957]: Eine Zuordnung (**Matching**) M ist größtmöglich genau dann wenn es keinen **augmentierenden Pfad** gibt.

Definition: Ein Pfad zwischen zwei Knoten heißt *augmentierend*, wenn beide Knoten nicht Endpunkt einer Kante in M sind und die Kanten des Pfades abwechselnd aus M und $\overline{M}$ (der Komplementmenge von M) sind.

Augmentierender Pfad (aus vorherigem Bsp.):



kann verbessert werden zu:



(Passives) Virales Marketing I

Wikipedia: Beim passiven viralen Marketing verbreitet der Nutzer die Nachricht **allein durch die Nutzung** des Produkts.

Beispiel: Der kostenlose Dienst Hotmail fügte an jede versendete Nachricht „P.S.: Get your private, free email at Hotmail“.
Idee: Bekommt jemand *viele* solcher E-Mails wird er auch zu Hotmail wechseln. $\rightsquigarrow$ Dominoeffekt

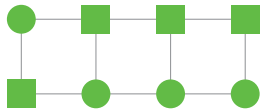
Es stellt sich die Frage: „Wer versendet die ersten E-Mails?“
Ziel: Möglichst wenige Benutzer auf herkömmlichen, teuren Weg von dem Produkt überzeugen, um möglichst viele Benutzer durch den „Dominoeffekt“ anzuwerben.

(Passives) Virales Marketing II

Modellierung als Graphproblem:

- Knoten $\hat{=}$ Menschen.
- Kanten $\hat{=}$ E-Mailkontakte zwischen Personen.
- Knoten sind grün (nutzt Hotmail) und rot (nutzt Hotmail nicht) gefärbt; Zu Beginn sind alle Knoten rot.
- einfache Färbungsregel: Pro „Zeittakt” ändert ein roter Knoten seine Färbung zu grün falls die Mehrheit seiner Nachbarn grün ist.

Fragestellung: Färbe eine minimale Anzahl an Knoten grün sodass nach wiederholter Anwendung obiger Färbungsregel alle Knoten grün gefärbt werden.



Virales Marketing III

Weitere Anwendungen obigen Modells:

- Verbreitung von ansteckenden Krankheiten:
- Fehlertoleranz in verteilten Systemen
- Verbreitung von Meinungen / Einstellungen in sozialen Netzwerken wie z.B. Facebook.

Beobachtung: Obiges Graphproblem ist also allgemein genug um in ganz verschiedenen Anwendungskontexten als Modell zu dienen...

Lobbying I

Natürlich sind nicht nur Graphen für die Modellbildung (Abstraktion) nützlich... Betrachte ein **Referendum zu mehreren Themen**:

Beispiel:

	Diplom statt Master	Direktwahl des Präsidenten	Internet-zensur
Wähler 1	✔	✔	✔
Wähler 2	✘	✔	✔
Wähler 3	✔	✘	✘
Wähler 4	✘	✔	✘
Wähler 5	✔	✘	✘
	⇓		
Ergebnis	✔(3:2)	✔(3:2)	✘(2:3)
Lobbyist	✘	✘	✔

Aufgabe:
Wähler so beeinflussen damit Ziel erreicht wird.

Triviale Lösung:
Mehr als Hälfte der Wähler beeinflussen.

Ziel:
Aus „Kostengründen“ so wenige Wähler wie möglich beeinflussen!

Lobbying II

Mathematische Modellierung als „Matrixmodifikationsproblem“:

	Diplom statt Master	Direktwahl des Präsidenten	Internet-zensur
Wähler 1	1	1	1
Wähler 2	0	1	1
Wähler 3	1	0	0
Wähler 4	0	1	0
Wähler 5	1	0	0
	⇓		
Ergebnis	1(3:2)	1 (3:2)	0 (3:2)
Lobbyist	0	0	1

Beobachtung:
Stimmabgaben haben nur **zwei** Werte: ✘ oder ✔.

Modellierung:
Referendum mit n Wählern und m Themen $\hat{=}$ binärer Matrix $\in \{0,1\}^{n \times m}$

Lobbying III

	Diplom statt Master	Direktwahl des Präsidenten	Internet-zensur
Wähler 1	1	1	1
Wähler 2	0	1	1
Wähler 3	1	0	0
Wähler 4	0	1	0
Wähler 5	1	0	0
	⇓		
Ergebnis	1(3:2)	1 (3:2)	0 (2:3)
Lobbyist	0	0	1

Problemstellung:
So **wenig Wähler wie möglich** beeinflussen um das Ziel des Lobbyisten zu erreichen.

Mitteilung: Das Problem ist „**NP-schwer**“, d.h. es existiert wahrscheinlich kein effizienter Lösungsalgorithmus.

Mitteilung: In aktueller Forschung konnten wir einen Lobbying-Algorithmus entwerfen, der für den Spezialfall von höchstens vier abzustimmenden Themen das Problem effizient löst!

Fazit: Immer darauf achten, ob vielleicht auch Spezialfälle des Modells bereits ausreichend „reich“ für die Anwendung in der Praxis sind.

Reguläre Ausdrücke

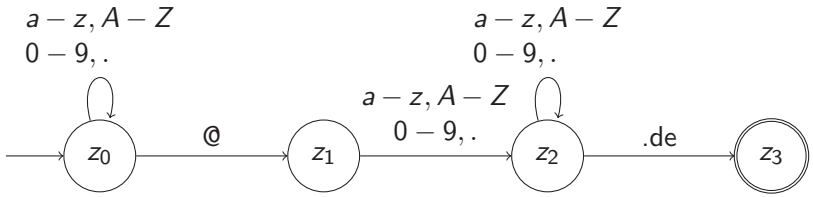
Beschreibung von gültigen E-Mailadressen mittels regulärem Ausdruck.
Vereinfachtes Beispiel:

$$(a-z, A-Z, 0-9, .)^{+}@(a-z, A-Z, 0-9, .)^{+}.de$$

Bedeutung: Eine E-Mailadresse entspricht der Form:

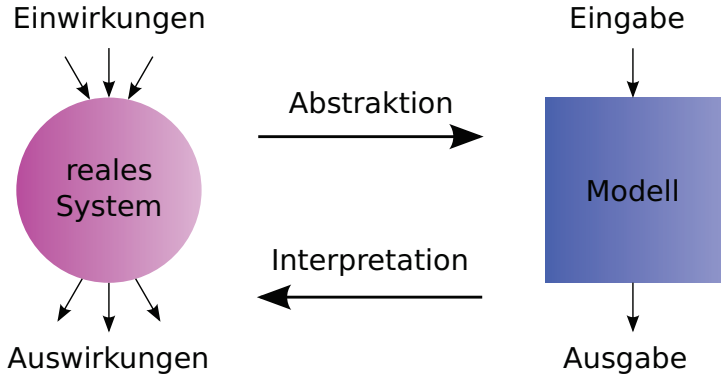
„Zeichenkette“ @ „Zeichenkette“ .de

Modellierung als „**nichtdeterministischer endlicher Automat**“:



Fazit: Abstraktion in der Informatik

Allgemeines Bild:



Modellierung realer Strukturen nach Reinhard Wilhelm (1996)

Abstraktion beim Softwareentwurf

- Von der Maschinensprache zu Hochsprachen und Skriptsprachen ($\rightsquigarrow$ Abstraktionshierarchien).
- Modularität: Programmmodule.
- Datenabstraktion: Datentypen etc.
- Objektorientierte Konzepte: Zusammenspiel kooperierender „Objekte“ mit Attributen und Methoden...
- Wiederverwendung von Software; Softwarebibliotheken...
- Entwurfsmuster („design patterns“) zur Softwaremodellierung...
- ...

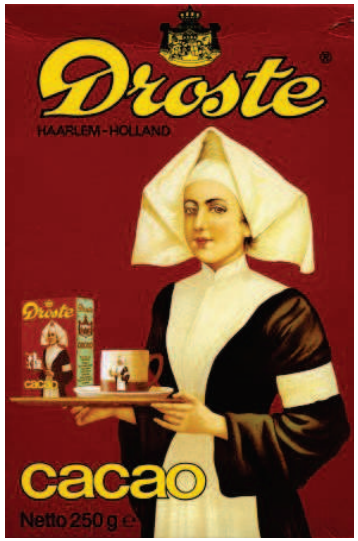
Abschließende Zitate zur Abstraktion in der Informatik

„Computer Science and Engineering is a field that attracts a different kind of thinker. I believe that one who is a natural computer scientist thinks algorithmically. Such people are especially good at dealing with situations where different rules apply in different cases; they are individuals who can rapidly change levels of abstraction, simultaneously seeing things ‘in the large’ and ‘in the small’.“ – Donald E. **Knuth**

„One of the defining characteristics of computer science is the immense difference in scale of the phenomena computer science deals with. From the individual bits of program and data in the computers to billions of operations per second on this information by the highly complex machines, their operating systems and the various languages in which the problems are described, the scale changes through many orders of magnitude“ . – Juris **Hartmanis**

„We all know that the only mental tool by means of which a very finite piece of reasoning can cover a myriad cases is called ‘abstraction’; as a result the effective exploitation of her/his powers of abstraction must be regarded as one of the most vital activities of a competent programmer.“ – Edsger W. **Dijkstra**

Rekursion



Der „Droste-Effekt“, 1904

Eine rekursive Definition

Eine Matrjoschka besteht aus einer Puppe, die eine Matrjoschka enthält, oder es ist die kleinste Matrjoschka.

Was ist Rekursion?

Wikipedia: Als Rekursion (lateinisch recurrere „zurücklaufen“) bezeichnet man die Technik in Mathematik, Logik und Informatik, eine Funktion bzw. ein Objekt durch sich selbst zu definieren.

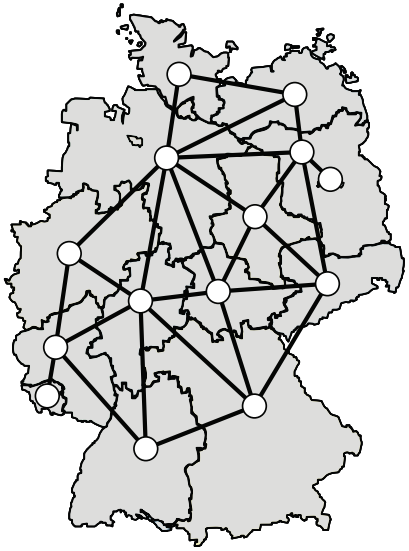
Beispiele:

1. Eine **Liste** besteht aus dem Erst-Element und der Rest-Liste, oder es ist die leere Liste.
2. Ein **Binärbaum** besteht aus einem Element (der Wurzel) und zwei binären Teil-Bäumen, oder es ist der leere Baum.

In der *Algorithmik* ist Rekursion eine mächtige „Reduktionstechnik“, der im wesentlichen folgende zwei Aspekte innewohnen:

- Falls eine Instanz eines Problems klein oder einfach genug ist, so löse sie einfach.
- Ansonsten führe das Problem auf eine oder mehrere *einfachere Instanzen des selben Problems* zurück.

Landkartenfärben revisited I



Beobachtung: Genaue Form der Bundesländer nicht wichtig, nur Nachbarschaftsrelation entscheidend.

Modellierung als Graphproblem:
Bundesländer $\hat{=}$ Knoten
Nachbarschaft $\hat{=}$ Kanten

Ziel: Finde Färbung der Knoten mit möglichst wenigen Farben, sodass keine zwei benachbarten Knoten die gleiche Farbe haben!

Vier-Farben-Satz: Jeder planare Graph ist vierfärbbar.

Landkartenfärben „revisited“ II

4-Färbung ist schwierig effizient zu finden, nicht aber 6-Färbung:

Konsequenz aus dem **Euler'schen Polyedersatz:** Für (einfache) planare Graphen gilt: Kantenzahl $\leq 3 \cdot$ (Knotenanzahl -2).

Folgerung: Jeder planare Graph hat einen Knoten mit max. fünf Nachbarn.

$\rightsquigarrow$**Rekursiver Algorithmus zum Berechnen einer 6-Färbung:**
Solange der Graph noch > 6 Knoten hat,

1. Finde einen Knoten v mit max. fünf Nachbarn,
2. Lösche v aus dem Graphen und färbe *rekursiv* den entstehenden Graphen mit höchstens sechs Farben.
3. Weise v eine der sechs Farben zu, die keiner von v's Nachbarn trägt.

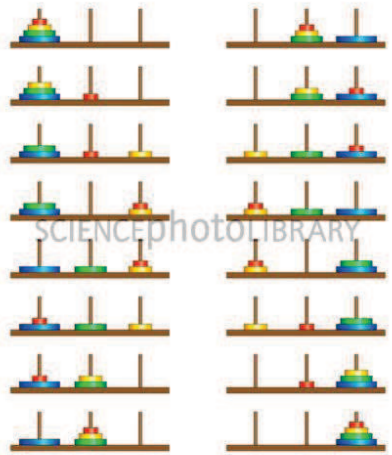
Hinweise:

- Nach Löschen eines Knoten bleibt ein Graph planar.
- Da v nur fünf Nachbarn hat, so muss eine der sechs Farben reichen, um v verschieden von all seinen Nachbarn zu färben.

Türme von Hanoi I

Ein klassisches Beispiel für die Eleganz von Rekursion:

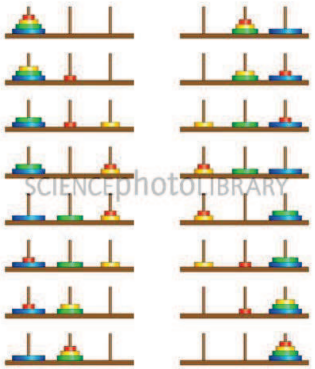
Gegeben n Scheiben (alle verschieden groß), die sich zu Beginn der Größe nach sortiert auf einem Stapel befinden.



Ziel ist nun die Bewegung des Stapels unter Zuhilfenahme eines „Zwischenstapels“ auf einen weiteren Stapel, wobei

- immer nur eine Scheibe transportiert werden kann und
- nie eine kleinere Scheibe unter einer größeren Scheibe liegen darf.

Türme von Hanoi II



Rekursive Lösungsidee: Zerteile das Bewegungsproblem in drei Phasen:

1. Bewege die oberen $n-1$ Scheiben zum Zwischenstapel ($\rightsquigarrow$ **Rekursion!**).
2. Bewege die letzte und größte Scheibe vom Start- zum Zielstapel.
3. Bewege alle Scheiben vom Zwischenstapel zum Zielstapel ($\rightsquigarrow$ **Rekursion!**).

Bemerkung: Wenn zur Bewegung von n Scheiben insgesamt $T(n)$ Einzelbewegungen nötig sind, so gilt $T(n) = 2 \cdot T(n-1) + 1 = 2^n - 1$, wobei $T(0) = 0$.

Bei $n = 64$ Scheiben würde man bei einer Scheibenbewegung pro Sekunde also 585 Milliarden Jahre brauchen...

Fibonacci-Zahlen I

Fibonacci-Zahlen II

Leonardo Fibonacci beschrieb 1202 das Wachstum einer Kaninchenpopulation mit Hilfe der Fibonacci-Folge.

Fibonacci-Folge: 0,1,1,2,3,5,8,13,21,34,...

Rekursives Konstruktionsprinzip: Die n-te Fibonacci-Zahl ergibt sich vermöge $F_n := \begin{cases} 0 & \text{falls } n = 0, \\ 1 & \text{falls } n = 1, \\ F_{n-1} + F_{n-2} & \text{falls } n > 1. \end{cases}$

$\rightsquigarrow$**Rekursiver Algorithmus zur Berechnung von F_n in Pseudocode**:

function $\mathrm{fib1}(n)$
1 **if** $n = 0$ **return** 0
2 **if** $n = 1$ **return** 1
3 **return** $\mathrm{fib1}(n-1) + \mathrm{fib1}(n-2)$

Fibonacci-Zahlen III

Wieviele Rechenschritte führt obiger Algorithmus aus?
Bei Eingabe n gibt es „offenbar“ F_n viele rekursive Aufrufe...
Können wir F_n leicht von unten *abschätzen*?

Ja!: Es gilt $F_n > 2^{(n-1)/2}$ (Beweisskizze siehe Tafel), also wächst die Zahl der rekursiven Aufrufe exponentiell in n und der rekursive Algorithmus ist somit hochgradig **ineffizient**!
Schnellster Supercomputer „Sequoia“ (Juni 2012) schafft 16 PetaFLOPS.
1 Rekursionsaufruf $\hat{=}$ 1 FLOP (Floating Point Operations Per Second)
$\rightsquigarrow$ Rechenzeit für F_{101}: < 1 Sekunde; F_{201}: > 2 Millionen Jahre!

Hoffnung (?): Bessere Hardware, sprich schnellere Rechner helfen!
„Moore's Gesetz“: *Die Rechengeschwindigkeit verdoppelt sich innerhalb von 18 Monaten.*
Leider hilft das nur unwesentlich bei exponentiell wachsenden Laufzeiten!

Aber es gibt eine viel wirksamere Lösung: **Iteration statt Rekursion** zur Vermeidung der überflüssigen Mehrfachberechnung gleicher Zwischenergebnisse...

Fibonacci-Zahlen IV

Entrekursivierung (Iteration statt Rekursion) verhilft zu einer drastischen Effizienzsteigerung:
Zentrale Ineffizienz bei $\mathrm{fib1}(n)$:

Mehrfachberechnung ein und desselben Ergebnisses!

$\rightsquigarrow$ Idee: Speicherung von Zwischenergebnissen:

Iterativer Algorithmus zur Berechnung von F_n:

function $\mathrm{fib2}(n)$
1 **if** $n = 0$ **return** 0
2 create array $f[0...n]$
3 $f[0] := 0$; $f[1] := 1$
4 **for** $i = 2, \ldots, n$:
5 $\quad f[i] := f[i-1] + f[i-2]$
6 **return** $f[n]$

Die Laufzeit von $\mathrm{fib2}(n)$ ist linear in n, d.h. es wird eine Anzahl von Rechenschritten benötigt, die direkt proportional zu n ist.

Tiefensuche I

Suche im Labyrinth:

- Aufgabe: Durchmustern eines Labyrinths.

- Präzisierung der Aufgabe:
  - Es soll sichergestellt werden, dass jede Kreuzung und jede Sackgasse irgendwann besucht wird.
  - Es soll verhindert werden, dass unbemerkt im Kreis gegangen wird.

Tiefensuche II (Durchmusterung eines Labyrinths)

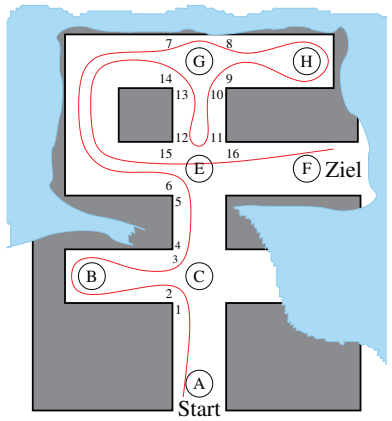
Lösung mit Hilfe von Markierungen an den Kreuzungen:

- Sackgasse: Umdrehen.
- Kreuzung: Beim Betreten Markierung an die Wand des Ganges über den man kam malen. Außerdem:

  Nicht im Kreis laufen! Hat der Gang, durch den man gekommen ist, eben seine erste Markierung bekommen, und sind weitere Markierungen vorhanden: Zweite Markierung malen und umdrehen.

  Sonst: Unerkundete Gänge suchen! Falls Gänge ohne Markierungen vorhanden: Davon den ersten von links nehmen, Markierung an die Wand malen.

  Sonst: Zurückgehen: Den Gang nehmen, der nur eine Markierung hat.



Tiefensuche III

- Grundidee:
  Immer möglichst tief in das Labyrinth hineingehen. Erst wenn dabei auf eine Sackgasse gestoßen wird, zur letzten Kreuzung zurückgehen und von dort aus erneut versuchen.

- Implementierung:

  function *Tiefensuche*(X):
  if *Zustand*[X] = „entdeckt“ **then return**;
  if X = *Ziel* **then exit** „Ziel gefunden!“;
  Zustand[X] := „entdeckt“;
  for each benachbarte Kreuzung Y von X
  Tiefensuche(Y);

Tiefensuche IV

Eigenschaften

- Vorteile:
  - Einfachheit.
  - Effizienz.
- Nachteile:
  - Es wird nicht der kürzeste Weg zum Ziel gefunden.
  - In unendlich großen Labyrinthen wird evtl. ewig in der falschen Richtung gesucht.

Weitere Anwendungen

Tiefensuche kann nicht nur in Labyrinthen, sondern in beliebigen „labyrinth-artigen“ Suchräumen angewendet werden, z.B.

- in Labyrinthen mit „Einbahnstraßen“,
- im Internet,
- im „Labyrinth“ der Spielzüge bei Solitairespielen,
- ...

Aspekte der Rekursion in der Informatik

Folgende Aspekte von Rekursion sind in der Informatik besonders wichtig:

- Funktionale Programmierung (mit Vorteilen wie modularer Struktur und Freiheit von Nebeneffekten) beruht wesentlich auf Rekursion!
- Rekursive Datentypen (z.B. Listen bestehen aus Datum und Zeiger auf eine Liste).
- Um rekursive Algorithmen auf Rechnern zu realisieren benutzt das „Laufzeitsystem“ die Datenstruktur „Keller“.
- Hilfsmittel zur Entwicklung eleganter und oft auch effizienter Algorithmen (wie z.B. Schnelle Fourier-Transformation vermöge „Teile und Herrsche“).
- Oft erzielt man auch Effizienzgewinne durch „Entrekursivierung“ (vgl. Fibonacci-Zahlen).
- Wesentliche Rolle in der Theorie der Berechenbarkeit.
- „To iterate is human, to recurse divine.“ – L. Peter **Deutsch**

Knobelaufgabe: Party-Problem

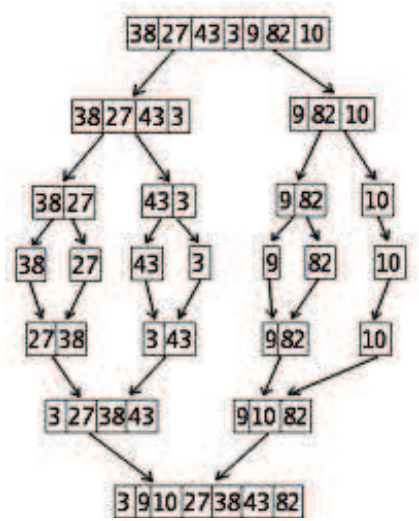
Problem: Es treffen sich fünf Paare auf einer Party. Unter Ihnen ist Tom. Weil sich noch nicht alle kennen, schüttelt jede Person allen ihr noch unbekannten Personen die Hand.

Wir kennen von allen Personen, außer von Tom, die Anzahl geschüttelter Hände. Dabei gilt, dass keine zwei aller anderen Personen die gleiche Anzahl von Händen geschüttelt haben.

Frage: Wieviele Hände hat Tom geschüttelt?

Lösung Party-Problem

Teile und Herrsche I:



Sortieren durch Mischen

Teile und Herrsche II

Prinzip: Teile und Herrsche (Divide and Conquer) ist eine rekursive Programmiertechnik, bei der man das zu lösende Problem in kleinere Teilprobleme zerlegt, die Teilprobleme einzeln per Rekursion löst, und dann die erzielten „Teillösungen“ zu einer Gesamtlösung kombiniert.

Ein besonders **häufiger Spezialfall** (z.B. zutreffend für Sortieren durch Mischen) ist, wenn man ein Problem in etwa *zwei gleich große Teilprobleme* zerlegt...; dies führt zu besonders effizienten Algorithmen. (Weitere Beispiele: Quicksort, Schnelle Fourier-Transformation.)

Bemerkung: Teile und Herrsche (lat. divide et impera) ist angeblich ein Ausspruch des französischen Königs Ludwigs XI. Er steht für das Prinzip, die eigenen Gegner und ihre Uneinigkeit für eigene Zwecke zu verwenden. Wichtig ist dabei, dass Uneinigkeit bei den Gegnern gefördert oder sogar verursacht wird, so dass diese in einzelnen, kleineren Gruppen leichter zu besiegen sind.

Bereits Sun Zi (um 500 v. Chr.) beschreibt sinngemäß Teile und Herrsche als eine Strategie der chinesischen Kriegskunst.

Teile und Herrsche III: Schnelles Potenzieren

Zur Berechnung der Zahl a^n für großes n verwende folgende Rekursion:

$$a^n := \begin{cases} 1 & \text{falls } n = 0, \\ (a^{n/2})^2 & \text{falls } n > 0 \text{ und } n \text{ gerade}, \\ a \cdot (a^{(n-1)/2})^2 & \text{falls } n > 0 \text{ und } n \text{ ungerade}. \end{cases}$$

Somit benötigt man nur $\log_2 n$ rekursive Aufrufe; dies ist für großes n viel effizienter als trivial $n-1$ Multiplikationen auszuführen...

Teile und Herrsche IV: Schnelles Multiplizieren

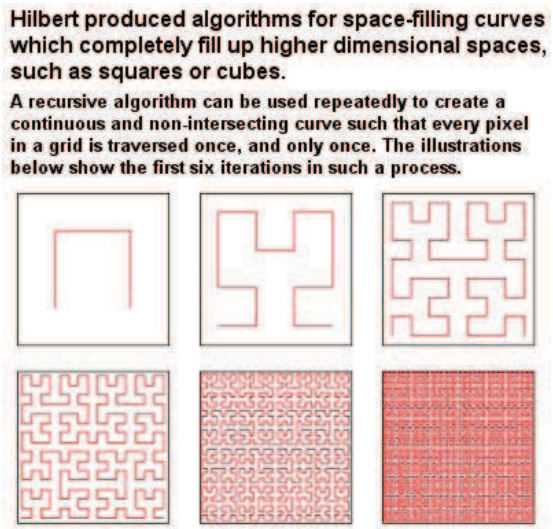
Aufgabe: Multipliziere zwei n-Bit-Zahlen x und y:

Beispiel: $1100 \cdot 1101 = 10011100$.
Mit der naiven Schulmethode braucht man ca. n^2 Bitoperationen.

Rekursiver Ansatz führt für große n zu einem schnelleren Algorithmus:
Sei $x = x_1 \cdot 2^{n/2} + x_0$ (wobei x_1 die $n/2$ höherwertigen Bits und x_0 die $n/2$ niederwertigen Bits bezeichne) und analog $y = y_1 \cdot 2^{n/2} + y_0$.
Dann gilt:
$x \cdot y = (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) = x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0 =$
$x_1 y_1 \cdot 2^n + ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) \cdot 2^{n/2} + x_0 y_0$.

Zentrale Beobachtung: Der Effizienzgewinn beruht darauf, dass die Multiplikation zweier n-Bit-Zahlen i.w. auf die Multiplikation dreier (und nicht wie vielleicht naiv angenommen vierer) $n/2$-Bit-Zahlen zurückgeführt werden kann!
(Beachte hierbei, dass Multiplikationen viel „teurer“ sind als Additionen!)

Raumfüllende Kurven



David Hilbert,
1862–1943

Die Berechnungskraft von Rekursion I

Eine insbesondere in der Berechenbarkeitstheorie berühmte rekursive Funktion ist die **Ackermann-Funktion**. Version nach Schöning:

$$\begin{aligned} f(k,1) &= 2, \\ f(1,n) &= n+2 \text{ für } n > 1, \\ f(k,n) &= f(k-1, f(k,n-1)) \text{ für } k,n > 1. \end{aligned}$$

Wilhelm Ackermann,
1896-1962

Bemerkung: Ackermann war promovierter Mathematiklehrer; eine Erklärung, warum er als aktiver Wissenschaftler nicht die akademische Laufbahn einschlug, gibt folgendes Zitat seines Doktorvaters David Hilbert: „Oh, das ist wunderbar. Das sind gute Neuigkeiten für mich. Denn wenn dieser Mann so verrückt ist, daß er heiratet und sogar ein Kind hat, bin ich von jeder Verpflichtung befreit, etwas für ihn tun zu müssen.“

Die Berechnungskraft von Rekursion II

Die Ackermann-Funktion wächst extrem schnell und ist ein berühmtes Beispiel für eine nicht „primitiv-rekursive“ Funktion.

$$\begin{aligned} f(k,1) &= 2, \\ f(1,n) &= n+2 \text{ für } n>1, \\ f(k,n) &= f(k-1, f(k,n-1)) \text{ für } k,n>1. \end{aligned}$$

Es gilt: $f(2,n) = 2n$; $\quad f(3,n) = 2^n$; $\quad f(4,n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{n\text{-mal}}$

Beispiele:
$f(5,1) = 2$.
$f(5,2) = f(4, f(5,1)) = f(4,2) = f(3,2) = f(2,2) = f(1,2) = 4$.
$f(5,3) = f(4, f(5,2))) = f(4,4) = \ldots = 2^{2^{2^2}} = 65536$.
$f(5,4) = f(4, f(5,3)) = f(4, 65536) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{65536\text{-mal}}$.

Die Berechnungskraft von Rekursion III

Nachfolgend betrachten wir Funktionen über den natürlichen Zahlen. Wir erklären zunächst einige einfache Funktionen quasi „per Festlegung“ („per Axiom“) als „berechenbar“: Dazu gehören insbesondere die *konstantwertigen* Funktionen und die „*Nachfolgerfunktion*“ $f(x) = f(x)+1$. Weiterhin dürfen Funktionen ineinander *eingesetzt* werden. D.h. sind f und g berechenbar, so auch $f(g(x))$.

Primitive Rekursion entspricht der **Iteration**:
Wenn eine Funktion f bereits „berechenbar“ ist, so soll es auch die zweistellige Funktion $g(n,x) := f(f(\ldots f(x)\ldots))$ sein, wobei f n-mal hintereinander ausgeführt wird.

Iterativ ließe sich das so mit einer FOR-Schleife ausdrücken:
INPUT n, x; $\quad y := x$; $\quad$ FOR $i := 1$ TO n DO $y := f(y)$;
OUTPUT y.

Die Berechnungskraft von Rekursion IV

Ein mächtigeres Werkzeug als die primitive Rekursion ist die μ-**Rekursion**, bei der i.w. FOR-Schleifen durch die mächtigeren WHILE-Schleifen ersetzt werden:

Wenn $f(n,x)$ berechenbar ist, dann auch $g(x) := \min\{n \mid f(n,x) = 0\}$.

Mit einer WHILE-Schleife lässt sich das wie folgt ausdrücken:
INPUT x; $\quad n := 0$; $\quad$ WHILE $f(n,x) \neq 0$ DO $n := n+1$;
OUTPUT n.

Mitteilung: Mit μ-Rekursion lässt sich die Ackermann-Funktion berechnen, nicht aber mit primitiver Rekursion. Weiterhin besteht die Hypothese, dass alles was berechenbar ist, auch mit μ-Rekursion berechenbar ist.

Grenzen der Berechenbarkeit

Alan Mathison Turing, 1912–1954

Das Post'sche Korrespondenzproblem I

Betrachte folgendes einfaches „Glasperlenspiel“:

Problem: Gegeben ist eine Menge von Tupeln (x_i, y_i) bestehend aus je zwei Binärzeichenketten.

Frage: Gibt es eine Aneinanderreihung der Tupel, wobei diese beliebig oft wiederholt werden dürfen, sodass das durch die „x-Teile“ gebildete Wort demjenigen der „y-Teile“ entspricht?

Emil Leon Post, 1897–1954

Beispiel 1:
$(x_1, y_1) = (1, 101)$, $(x_2, y_2) = (10, 00)$, $(x_3, y_3) = (011, 11)$.
Dann gilt:

$$x_1 \circ x_3 \circ x_2 \circ x_3 = \langle 1 \circ 011 \circ 10 \circ 011 \rangle = \langle 101110011 \rangle$$
$$y_1 \circ y_3 \circ y_2 \circ y_3 = \langle 101 \circ 11 \circ 00 \circ 11 \rangle = \langle 101110011 \rangle$$

Das Post'sche Korrespondenzproblem II

Beispiel 2:
$(x_1, y_1) = (10, 101)$, $(x_2, y_2) = (011, 11)$, $(x_3, y_3) = (101, 011)$.

Es gibt keine Lösung. Klar: Man muss mit erstem Tupel anfangen:

$$\begin{array}{ll} \langle 10 \ldots \rangle & (1) \\ \langle 101 \ldots \rangle & \end{array}$$

Dann muss das „x-Wort“ mit 1 fortgesetzt werden, also ist das erste oder dritte Tupel zu benutzen.
Das erste Tupel liefert $\langle 1010 \ldots \rangle$ und $\langle 101101 \ldots \rangle$ $\lightning$.

Also betrachte drittes Tupel:

$$\begin{array}{l} \langle 10101 \ldots \rangle \\ \langle 101011 \ldots \rangle \end{array}$$

Das x-Wort muss wieder mit 1 fortgesetzt werden, analog zu (1)
$\rightsquigarrow$ drittes Tupel wieder nehmen $\rightsquigarrow$ Endlosschleife.

Knobelaufgabe zum Post'schen Korrespondenzproblem

Hat folgendes Post'sche Korrespondenzproblem (PCP) eine Lösung? Welche?
$(x_1, y_1) = (1, 1111)$, $(x_2, y_2) = (111, 1)$,
$(x_3, y_3) = (111, 11111111111111111)$.
Eine Lösung: $x_2 \circ x_2 \circ x_2 \circ x_1 \circ x_1 = 11111111111 = y_2 \circ y_2 \circ y_2 \circ y_1 \circ y_1$

Allgemeine Frage:
Gibt es einen Algorithmus, der bei Eingabe beliebiger Tupel immer entscheiden kann, ob das entsprechende PCP eine Lösung hat?

Bemerkung: Die kürzeste Lösung zu folgendem PCP hat eine Länge von 252:

$(1101, 1)$, $(0110, 11)$, $(1, 110)$.

Finden Sie sie? Wie?

Was ist berechenbar (und was nicht)?

Aus den Vorbetrachtungen zum PCP ergeben sich folgende Fragen:

- Ist prinzipiell alles berechenbar (algorithmisch entscheidbar)? Oder gibt es interessante und natürliche Probleme bzw. Funktionen, die nicht berechenbar sind?
- Was heißt überhaupt berechenbar? Wie formalisiert man den Begriff „Algorithmus“?
- Gibt es verschiedene Modelle der Berechenbarkeit? Gibt es verschiedene Typen von Berechenbarkeit oder nur die *eine* (sinnvolle) Definition von Berechenbarkeit?

Zur Formalisierung des Berechenbarkeitsbegriffs I

Bereits im vorigen Kapitel über Rekursion erwähnte Hypothese:
Alle berechenbaren Funktionen (über den natürlichen Zahlen) sind mit μ-**Rekursion** (genauer: **partiell rekursiven Funktionen**) berechenbar.
$\rightsquigarrow$ Rechnen also als Einsetzen und Auswerten von mathematisch definierten Funktionen...

Jedoch ist μ-Rekursion ein sehr mathematisches, sehr an den natürlichen Zahlen „klebendes“ Berechnungsmodell...
Gibt es ein elementares, unmittelbar verständliches, sehr einfaches **Maschinenmodell** für Berechenbarkeit?

Alan Turing hat den Berechenbarkeitsbegriff mit den (heute) sogenannten **Turing-Maschinen** (1936) definiert, nachwievor das wohl wichtigste Berechnungsmodell mit großer Relevanz bis hinein in die Komplexitätstheorie und Algorithmik.

Zur Formalisierung des Berechenbarkeitsbegriffs II

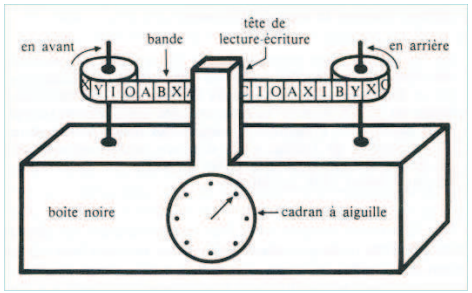
Die Turing-Maschine ist quasi ein zur Rekursion (und anderen Konzepten wie z.B. „λ-Kalkül“) konkurrierendes Modell für Berechenbarkeit. Sie modelliert die Arbeitsweise eines Computers auf besonders einfache und gut analysierbare (!) Weise.

So ergibt sich dann also die weitere Hypothese, dass alles was berechenbar ist, auch Turing-berechenbar ist; und somit insbesondere, dass die Berechnungsformalismen „partiell rekursive Funktionen“ und Turing-Maschinen und viele andere (in der Tat alle bekannten) gleichmächtig sind $\rightsquigarrow$ **Church'sche These**.

Alonzo Church, 1903–1995

Es gilt insbesondere, dass alles was mit Registermaschinen (i.w. unseren Standardrechnern eins-zu-eins entsprechend) berechenbar ist, auch mit Turing-Maschinen berechenbar ist.

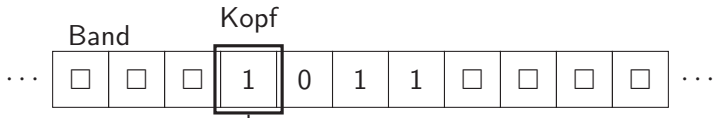
Die Turing-Maschine



Anschaulich gesprochen besteht eine **Turing-Maschine** aus

- einem (potenziell) unendlichen, in Felder unterteiltem **Band**, wobei jedes Feld ein Zeichen eines Alphabets (z.B. $\{0,1\}$) speichern kann,
- einem **Schreib-Lese-Kopf**, der sich schrittweise über die Felder bewegt und jeweils ein Feld beschreiben oder lesen kann und
- einer **endlichen Kontrolleinheit**, die in jeweils genau einem von endlich vielen Zuständen ist.

Turingmaschine – Ein Rechenbeispiel



Programm

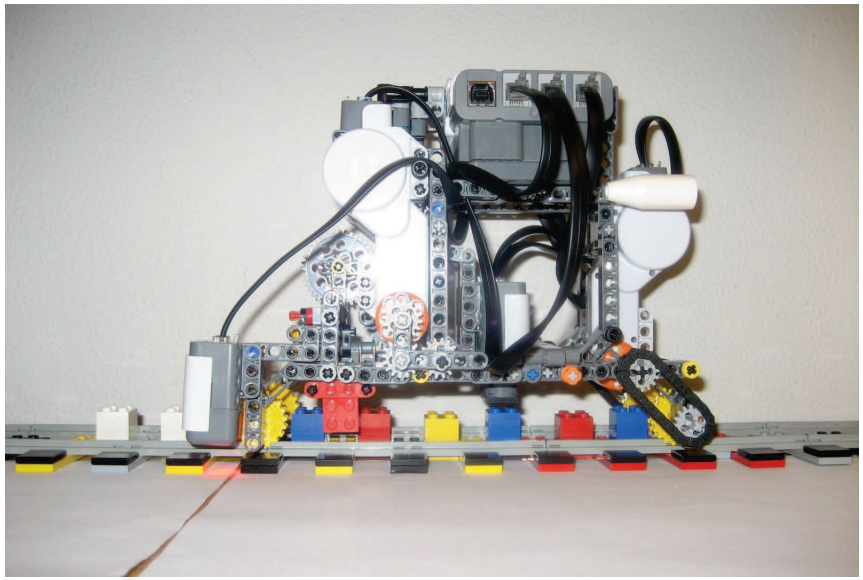
	0	1	□
z_0	(0, R, z_0)	(1, R, z_0)	(□, L, z_1)
z_1	(1, L, z_e)	(0, L, z_1)	(1, R, z_e)
z_e			

Die Turingmaschine hat drei Zustände z_0, z_1 und z_e.

- z_0: Bewege Kopf an das rechte Ende der Eingabebinärzeichenkette.
- z_1: „Flippe“ Bits bis zur ersten gelesenen 0.
- z_e: Endzustand.

Die Turingmaschine addiert eins zu einer gegebenen Binärzahl.

Turing-Maschinen sind real!



Berechenbarkeit und Entscheidbarkeit

Wir halten fest: Eine Funktion heißt **berechenbar**, wenn es eine Turing-Maschine (also einen Algorithmus) gibt, die diese Funktion berechnet.

Eine berechenbare Funktion mit auf $\{0,1\}$ eingeschränkten Wertebereich[4] wird **entscheidbar** genannt.

Weiterhin spricht man hier gleichbedeutend von **Entscheidungsproblemen**.

Die Church'sche These besagt also, dass alle berechenbaren Entscheidungsprobleme durch eine Turing-Maschine entscheidbar sind.

[4] Solche Funktionen definieren 1:1 Sprachen: Genau diejenigen Elemente des Definitionsbereichs sind in der Sprache, die als Funktionswert 1 liefern.

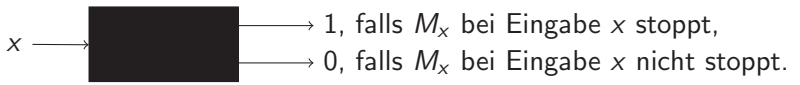
Einige Vorteile der Turing-Machine

- Prinzipiell nicht an Operationen auf natürlichen Zahlen gebunden.
- Maschinencharakter mit sehr intuitiver, leicht verständlicher Funktionsweise.
- Leichte Simulierbarkeit, insbesondere einer Turing-Maschine durch eine andere... $\rightsquigarrow$ Begriff der **universellen Turing-Maschine**.
- Elementar einfache, aber auch gut variierbare Definition.
- Leichte Variation des Berechnungsmodells ohne die Berechnungsmächtigkeit des Modells zu ändern.
- Große Relevanz in der Komplexitätstheorie.
- Kanonische Fortsetzung endlicher Automaten (und Kellerautomaten), die in der Theorie der formalen Sprachen und Grammatiken eine zentrale Rolle spielen.

Zur Unentscheidbarkeit des Halteproblems I

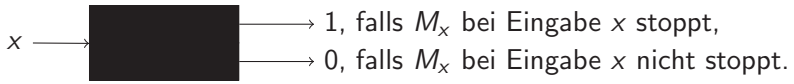
Eine zentrale Einsicht ist, dass jedes „Programm“ (sprich jede Turing-Maschine), die eine Zeichenkette $x \in \{0,1\}^*$ als Eingabe verarbeitet sich selbst wiederum auch als Zeichenkette aus $\{0,1\}^*$ „kodieren“ lässt. Damit kann man aber eine solche Turing-Maschine auf ihre eigene Kodierung als Eingabe anwenden! (Stichwort **Selbstanwendung**; vgl. Compiler.)

Nehmen wir also an, es gäbe eine Turing-Maschine M, die auf jeder Eingabe $x \in \{0,1\}^*$ berechnet, ob die durch x (eindeutig) kodierte Turing-Maschine M_x hält ($\rightsquigarrow$ Ausgabe 1) oder nicht ($\rightsquigarrow$ Ausgabe 0). Wir stellen uns M schematisch wie folgt vor:



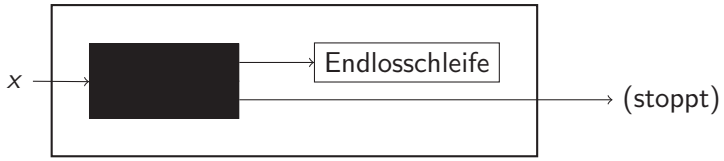
Zur Unentscheidbarkeit des Halteproblems II

Situation: M:



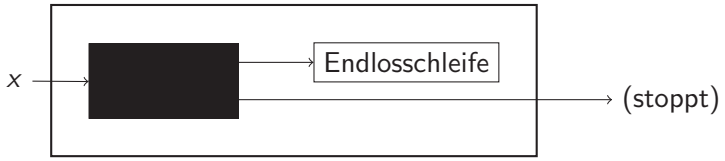
Nun baue mithilfe obiger Black Box eine neue Turing-Maschine M', die genau dann 0 ausgibt und stoppt, wenn die Black Box 0 ausgibt. Falls die Black Box 1 ausgibt, so gehe M' in eine Endlosschleife.

$\rightsquigarrow$ Bild für M':



Zur Unentscheidbarkeit des Halteproblems III

Situation: M':



Sei nun $z \in \{0,1\}^*$ die Kodierung der neuen Turing-Maschine M' als Binärzeichenkette. Was macht M' auf Eingabe $x = z$ (**Selbstanwendung!**)? Stoppt M', ja oder nein?

Beide Antworten führen zu einem logischen Widerspruch! Daraus folgt, dass die Ursprungsannahme dass M existiere falsch gewesen sein muss!

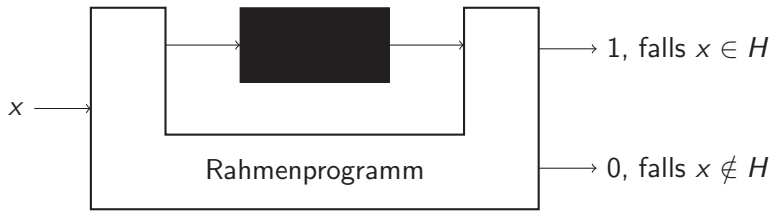
$\rightsquigarrow$ Das **(spezielle) Halteproblem** H ist also unentscheidbar (d.h. nicht berechenbar), wobei H die Menge aller Binärwörter x ist, so dass die durch x kodierte Turing-Maschine auf Eingabe x hält!

Weitere unentscheidbare Probleme I

Mithilfe des Halteproblems kann man eine Vielzahl anderer Probleme (Funktionen) als unentscheidbar (nicht berechenbar) nachweisen.

Die zentrale Idee hierbei beruht auf dem Konzept der **Reduktion**: Falls wir zeigen können, dass wir mithilfe eines Algorithmus A einen neuen Algorithmus B bauen könnten, der das Halteproblem H entscheidet, so muss daraus folgen, dass A nicht entscheidbar sein kann.

$\rightsquigarrow$ Bild (A entspricht der Black Box, B dem Gesamtbild):



Weitere unentscheidbare Probleme II

Mithilfe des Reduktionsprinzips lassen sich nun viele Probleme als unentscheidbar nachweisen:

- Post'sches Korrespondenzproblem.
- Bestimme, ob ein gegegebenes Gleichungssystem (Polynome) eine ganzzahlige Lösung besitzt (Stichwort „Lösbarkeit Diophantischer Gleichungen“).
- Bestimme, ob eine gegebene Funktion bijektiv (oder surjektiv oder injektiv) ist.
- ...

Der **Satz von Rice** besagt, dass es unmöglich ist, irgendeinen nichttrivialen Aspekt des funktionalen Verhaltens einer Turingmaschine (oder eines Algorithmus in einem anderen Berechnungsmodell) algorithmisch zu entscheiden.
($\rightsquigarrow$ „Fast nichts" ist berechenbar!)

Kreative Analyse lässt sich also wohl nicht vollends automatisieren...

Ein Papst der Unentscheidbarkeit und Unvollständigkeit

Kurt Gödel, 1906–1978

Algorithmische Komplexität

Bisher gesehen: $\exists$ nicht-berechenbare Probleme, z.B. das Halteproblem und das Post'sche Korrespondenzproblem.

$\exists$ viele Algorithmen zur Lösung konkreter Probleme wie z.B. den Propose&Reject-Algorithmus für Stable Matching oder den rekursiven Algorithmus zum Potenzieren einer Zahl.

„Offensichtlich" waren diese Algorithmen korrekt, doch wie bewerten bzw. analysieren wir ihre **Effizienz**, sprich die von ihnen benötigte Laufzeit,[5] also die Anzahl der von ihnen ausgeführten Elementaroperationen?

Naheliegend: Laufzeit sollte von der Eingabegröße abhängig sein – große Eingaben benötigen in der Regel mehr Laufzeit...

Zentrale Frage: Wann nennen wir ein Berechnungsproblem effizient lösbar? Wie lässt sich das formalisieren?

[5] Eine zweite wichtige Ressource ist der Speicherplatzbedarf; dieser ist in den meisten (aber nicht allen) Anwendungen gegenüber der Laufzeit sekundär.

O-Notation zur Laufzeitanalyse

Eine genaue Laufzeitangabe für einen Algorithmus in Abhängigkeit von der Eingabegröße n ist oft schwierig und vom jeweiligen Maschinenmodell abhängig. Dem begegnet man (teilweise) mit der Benutzung der O-Notation:

Definition: Seien f und g Funktionen von den natürlichen in die reellen Zahlen. Dann bezeichnet man mit $O(f(n))$ die Menge aller Funktionen g, so dass gilt

$$\exists c > 0 \quad \exists n_0 > 0 \quad \forall n \geq n_0 : g(n) \leq c \cdot f(n).$$

(Intuition: Ignoriere konstante Faktoren.)

Beispiele:

- $3n^4 + 5n^3 + 7\log_2 n \in O(n^4)$.
- $n \log_2 n \in O(n^2)$.
- $n\sqrt{n} \in O(n^2)$.
- $n^{10} \in O(2^n)$.

Propose&Reject für kdk Matching – revisited

Zur Erinnerung:

Propose&Reject [Gale, Shapley 1962]

1 Initialisiere alle $m \in M$ und alle $w \in W$ als „frei"
2 **while** $\exists m \in M$: m ist frei und $\exists w \in W$ der m noch keinen Antrag gemacht hat
3 $w \leftarrow$ erste noch „unbeantragte" Frau in m's Präferenzfolge
4 **if** w ist frei **then:**
5 (m, w) wird Paar, $m \leftarrow$„verlobt", $w \leftarrow$„verlobt"
6 **else if** w zieht m ihrem aktuellen „Verlobten" m' vor **then:**
7 (m, w) wird Paar, $m \leftarrow$„verlobt", $w \leftarrow$„verlobt", $m' \leftarrow$„frei"
8 **else:** w lehnt m ab

Unter der Annahme $|M| = |W| = n$ hat der Propose&Reject-Algorithmus eine Laufzeit von $O(n^2)$.

Hier und nachfolgend: Laufzeitanalyse immer auf den schlimmsten möglichen Fall bezogen (**Worst-Case-Szenario**)!

Einige Laufzeiten von bereits bekannten Algorithmen

Hinweis: Nachfolgend nehmen wir an, dass z.B. eine Zuweisung oder die Addition zweier Zahlen eine Elementaroperation ist.

1. Der rek. Alg. zum Finden einer 6-Färbung einer Landkarte hat eine Laufzeit von $O(n^2)$, wobei n die Anzahl der Länder ist.
2. Der rekursive Algorithmus für die Türme von Hanoi hat eine Laufzeit von $O(2^n)$.
3. Der rekursive Algorithmus zur Berechnung der n-ten Fibonacci-Zahl hat eine Laufzeit von mindestens $O(2^{n/2})$.
4. Der iterative Algorithmus zur Berechnung der n-ten Fibonacci-Zahl hat eine Laufzeit von $O(n)$.

Zentrale Frage: Welche der obigen Algorithmen gelten als **effizient**?

Die vereinfachte Antwort der (theoretischen) Informatik ist, dass alle Algorithmen mit **polynomieller Laufzeit** als effizient gelten, und solche mit **„super-polynomieller" Laufzeit** als ineffizient. Oben sind also der erste und der letzte Algorithmus effizient.

Funktionenwachstum

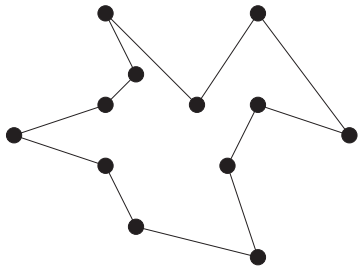
Der Unterschied zwischen „**polynomiellem Wachstum**" einer Funktion und „**exponentiellem Wachstum**" ist enorm:

n	n^2	2^n	$n!$
5	25	32	120
10	100	1024	$\approx 3 \cdot 10^6$
20	400	$\approx 10^6$	$\approx 2 \cdot 10^{18}$
50	2500	$\approx 10^{15}$	$\approx 3 \cdot 10^{64}$
100	10000	$\approx 10^{30}$	$\approx 9 \cdot 10^{157}$

Beispiel eines Berechnungsproblems mit bester bekannter Laufzeit $O(2^n)$: Traveling Sales Person (TSP)...

Traveling Salesperson Problem (TSP) I

Eingabe: n Punkte mit paarweisen Abständen und eine Zahl ℓ.
Aufgabe: Gibt es eine Rundtour der Länge $\leq \ell$, die alle Punkte genau einmal besucht.



Triviale Lösungsfindung: Probiere alle $n!$ Möglichkeiten (Permutationen der Punkte), wähle die Beste aus.

Verbesserung: Mithilfe der Methode des dynamischen Programmierens lässt sich die Laufzeit i.w. zu $O(2^n)$ verbessern; eine weitere beweisbare Verbesserung ist seit fünfzig Jahren offen!

Funktionenwachstum und hypothetische Laufzeiten

Algorithmenlaufzeit (d.h. Anzahl der Elementaroperationen) bei Eingabegröße n und hypothetische Rechenzeit bei Annahme von 10^9 Elementaroperationen pro Sekunde.

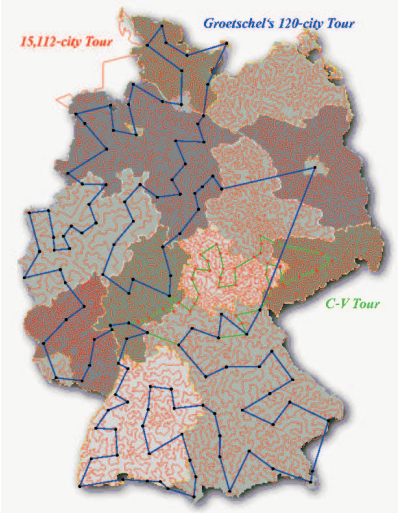
n	$n!$	2^n	Laufzeit $n!$	Laufzeit 2^n
5	120	32	10^{-7} Sekunden	$< 10^{-7}$ Sekunden
10	$\approx 3 \cdot 10^6$	1024	10^{-3} Sekunden	10^{-6} Sekunden
20	$\approx 2 \cdot 10^{18}$	$\approx 10^6$	77 Jahre	10^{-3} Sekunden
50	$\approx 3 \cdot 10^{64}$	$\approx 10^{15}$	$9 \cdot 10^{47}$ Jahre	12 Tage
100	$\approx 9 \cdot 10^{157}$	$\approx 10^{30}$	$3 \cdot 10^{141}$ Jahre	10^{13} Jahre

Bemerkung: In der Praxis gelten für Eingabegröße n selbst Laufzeiten wie $O(n^2)$ oft als ineffizient; für die meisten Probleme ist keine Laufzeit besser als $O(n)$ (sogenannte **Linearzeit**) zu erreichen.

Aber was tun bei exponentiellen Laufzeiten?

Traveling Salesperson Problem (TSP) II

Mit ausgefeilten Heuristiken lassen sich dennoch vergleichsweise große TSP-Instanzen exakt lösen (Fortschritt durch Algorithmik!):



Schnellere Rechner bauen? Hilft praktisch nichts!

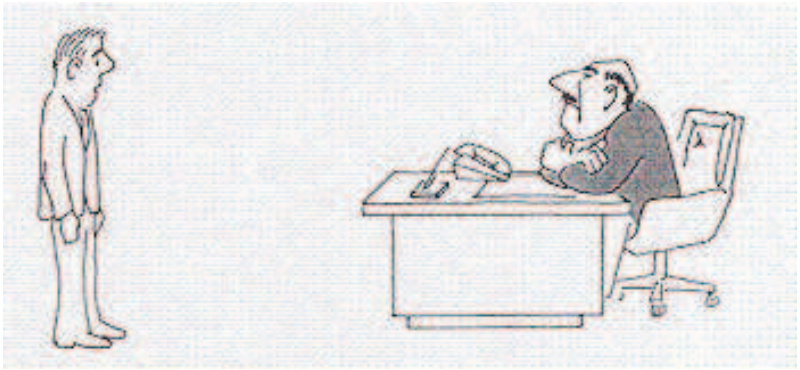
Im Kampf gegen exponentielle Laufzeiten helfen schnellere Rechner kaum:
Nehmen wir an, wir brauchen 2^n Rechenschritte zum Finden z.B. einer kürzesten Rundtour bei einer TSP-Instanz mit n Punkten. Dann könnte man

- mit tausendfach schnelleren Rechnern in etwa der gleichen Zeit Rundtouren in Eingaben mit 10 mehr Punkten finden;
- mit millionenfach schnelleren Rechnern in etwa der gleichen Zeit Rundtouren in Eingaben mit 20 mehr Punkten finden.

$\rightsquigarrow$ Fluch des exponentiellen Wachstums!

Wie wissen wir, ob wir nicht nur zu dumm waren, für wichtige Probleme Polynomzeit- statt Expontialzeitalgorithmen zu finden?
Was, wenn der Chef unbedingt einen schnellen Algorithmus fordert?
Die möglichen Sachlagen sind folgende:

Zu dumm für einen effizienten Algorithmus?

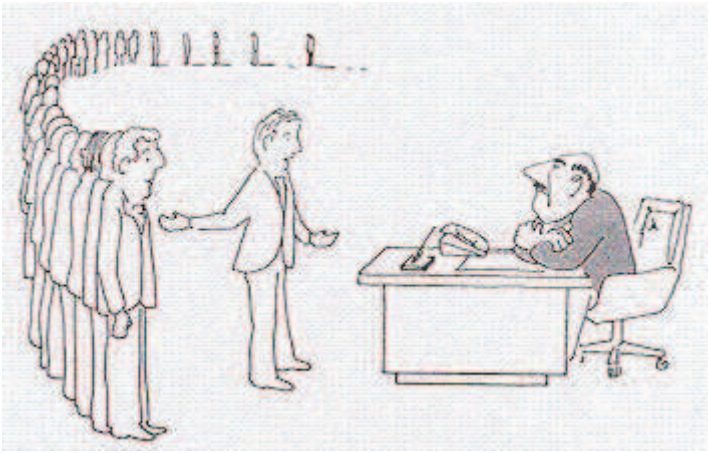


I can't find an efficient algorithm, I guess I'm just too dumb.

Es gibt keinen effizienten Algorithmus!

I can't find an efficient algorithm, because no such algorithm is possible.

Nicht nur ich finde keinen Algorithmus! I



I can't find an efficient algorithm, but neither can all these famous people.

Nicht nur ich finde keinen Algorithmus! II

Für viele wichtige Probleme (z.B. TSP) ist unbekannt, ob sie in polynomieller Laufzeit gelöst werden können; jedoch konnte bislang auch nicht bewiesen werden, dass es einen solchen Algorithmus gar nicht geben kann.

Als „Ausweg“ (Ausrede?) gibt es die **Theorie der NP-Vollständigkeit**. Sie hilft, verschiedene Berechnungsprobleme untereinander in Beziehung zu setzen sodass gilt, dass entweder alle oder keines dieser Probleme (sprich alle **NP-vollständigen Probleme**) effizient lösbar sind.

Da bisher für keines der NP-vollständigen Probleme (das sind quasi die „härtesten Nüsse“ in NP) ein Polynomzeitalgorithmus gefunden werden konnte, besteht die weit verbreitete Meinung, dass kein NP-vollständiges Problem (wie z.B. TSP) effizient lösbar ist.

Berechnungsschwere Entscheidungsprobleme I: 3-SAT

Vielleicht das zentralste und wichtigste NP-vollständige Problem ist 3-SAT:

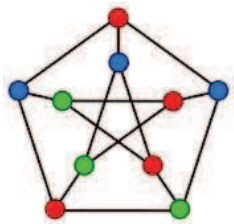
Eingabe: Aussagenlogische Formel F in „konjunktiver Normalform“ („ein großes UND von kleinen ODERn“) mit höchstens drei Literalen pro Klausel.
Frage: Ist F erfüllbar, sprich gibt es eine $\{0,1\}$-wertige Belegung der in F verwendeten Boole'schen Variablen derart, dass F zu 1 ausgewertet wird?

Beispiel: $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_4)$ ist erfüllbar z.B. vermöge $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ (und x_4 beliebig).

Achtung: Im Gegensatz zu 3-SAT ist 2-SAT (nur zwei statt drei Literale pro Klausel) effizient lösbar (nichttrivialer Algorithmus)!

Berechnungsschwere Entscheidungsprobleme II: 3-Coloring

Eingabe: Ein ungerichteter Graph.
Frage: Lassen sich die Knoten des Graphen mit drei Farben so färben, dass keine zwei mit einer Kante verbundenen Knoten die gleiche Farbe haben?

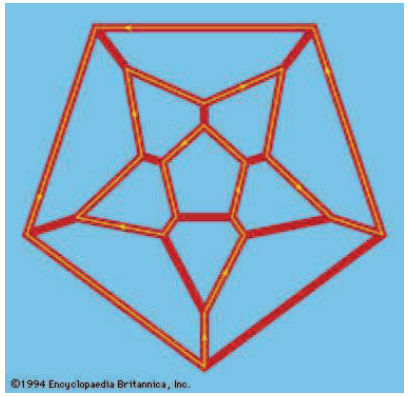


Achtung: Im Gegensatz zu 3-Coloring ist 2-Coloring (nur zwei statt drei Farben) effizient lösbar (einfacher Algorithmus)!

Berechnungsschwere Entscheidungsprobleme III: Hamilton-Kreis

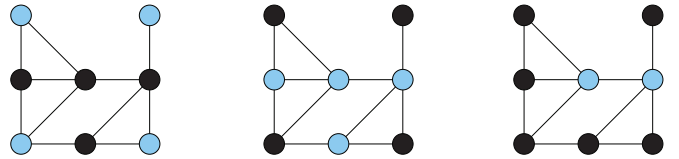
Das Hamilton-Kreis-Problem ist i.w. ein Spezialfall des TSP (warum?):

Eingabe: Ein ungerichteter Graph.
Frage: Gibt es eine Route durch den Graphen, die jeden Knoten genau einmal durchläuft und dann zum Ausgangsknoten zurückkehrt?



Achtung: Im Gegensatz zum Finden eines Hamilton-Kreises ist das Finden eines Euler-Kreises leicht (warum?)!

Weitere berechnungsschwere Entscheidungsprobleme IV



Independent Set Vertex Cover Dominating Set

Eingabe: Ein ungerichteter Graph G und eine Zahl $k > 0$.
Independent Set-Frage: Gibt es k Knoten in G, die paarweise untereinander nicht mit einer Kante verbunden sind?

Vertex Cover-Frage: Gibt es k Knoten in G, sodass jede Kante in G mindestens einen dieser Knoten als Endpunkt hat?

Dominating Set-Frage: Gibt es k Knoten in G, sodass jeder andere Knoten mindestens einen dieser k Knoten als Nachbarn hat.

Bemerkung: Independent Set und Vertex Cover sind sehr eng verwandt und „gleichschwer“ zu lösen (warum?)!

Zum Wesen der Probleme in NP

Alle Probleme in der Klasse NP, also insbesondere TSP, 3-SAT, 3-Coloring, etc. haben folgendes gemeinsam:

Hat man eine Lösung gefunden (dies ist der berechnungsintensive Anteil), so ist leicht zu überprüfen, ob die gefundene bzw. vorgeschlagene Lösung tatsächlich die geforderten Eigenschaften erfüllt!

Insbes. gilt, dass jede Lösung mit nur polynomiell vielen Bits bezogen auf die Eingabegröße beschreibbar und in polynomieller Laufzeit überprüfbar ist (daher N**P**!).

Zusammenfassend heißt das, dass alle Probleme in NP mit einem „**Rate&Verifiziere-Algorithmus**“ gelöst werden können:

1. Rate eine Lösung.
2. Verifiziere ob die geratene Lösung korrekt ist.

Das Raten ist der kritische Teil, da er „**Nichtdeterminismus**“ (daher **N**P!) voraussetzt und bislang i.w. nicht besseres bekannt ist, als alle Ratemöglichkeiten (leider sind das exponentiell viele) durchzuprobieren um eine gültige Lösung zu finden (falls sie existiert).

Rate&Verifiziere am Beispiel 3-SAT

Zu ratende Lösung: Belegung der Boole'schen Variablen mit Werten aus $\{0, 1\}$;

bei n Variablen gibt es 2^n mögliche verschiedene Belegungen, die ggf. alle zu überprüfen sind.
Durch nichtdeterminisisches Raten wird dieser Aufwand „umgangen“.

(Nichtdeterminismus ist kein reales Rechnerkonzept, sondern ein gedankliches Hilfsmittel!)

D.h.: Ein „Beweis“ besteht hier aus genau n Bits, nämlich den Wahrheitswerten 0 oder 1 für die n Variablen der Formel.

Verfikation der Lösung: Belege die Formelvariablen gemäß der jeweiligen Belegung mit den Werten 0 und 1 und überprüfe, ob sich die Formel zu 1 auswertet.

Die Komplexitätsklassen P und NP

Phänomen: Z.B. 2-Coloring ist leicht in polynomieller Zeit zu entscheiden, hingegen 3-Coloring scheinbar nicht. Beide haben polynomiell große „**Beweise**“, sprich die korrekten Färbungen (je Knoten Angabe einer Farbe).
Dies führt zur Einführung der zwei wichtigsten „**Komplexitätsklassen**“ der Informatik:

P: Menge von Entscheidungsproblemen, die sich mit **polynomiell** langen Beweisen lösen lassen, welche auch in **polynomiell** vielen Schritten gefunden werden können.

NP: Menge von Entscheidungsproblemen, die sich mit **polynomiell** langen Beweisen lösen lassen.

2-Coloring liegt in P, 3-Coloring aber lediglich in NP. 3-Coloring gehört zu den schwierigsten Problemen in NP, d.h. zu den „NP-vollständigen Problemen“.

Das P versus NP-Problem

Klar ist, dass $P \subseteq NP$.

Die große offene Frage: Ist

$$\mathbf{P = NP?}$$

Anders gesagt: Besitzen alle Entscheidungsprobleme mit kurzen Beweisen auch schnelle Konstruktionsverfahren (also Algorithmen), um diese Beweise zu finden?

Wichtige Tatsache: Kann man für ein NP-vollständiges Problem zeigen, dass es in P liegt, so folgt $P = NP$.

Bemerkung: Es gibt viele Tausende, in verschiedensten praktischen Anwendungen auftretende NP-vollständige Probleme. Für keines konnte je ein schneller Algorithmus angegeben werden, obwohl sich schon Abertausende von Forschern daran versucht haben.

Zur Soziologie des P versus NP-Problems

Zunächst: Könnte man die Frage **P** = **NP** beantworten, so wäre man ein Weltstar der Wissenschaft wie es ihn bislang nicht gab . . . und auch um eine Million Dollar reicher (Clay Mathematics Institute).

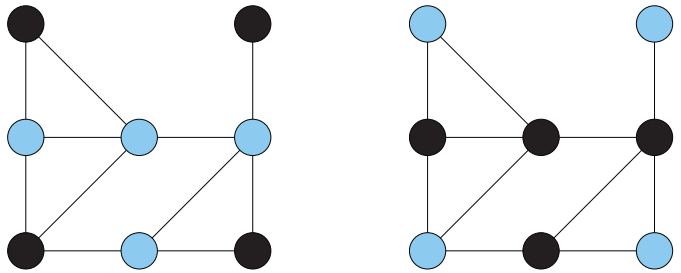
Wohl kein wissenschaftliches Problem wurde von derart vielen Menschen erfolglos bearbeitet.

Eine **Umfrage** im Jahre 2002 in den USA unter 100 namhaften Theoretischen Informatikern ergab folgende Gefühlslage:

1. 5 glaubten an die Beantwortung bis 2009;
2. 35 an die Beantwortung bis 2049;
3. 7 an die Beantwortung zwischen 2100 und 2110;
4. 5 an die Beantwortung zwischen 2200 und 2300;
5. 5 daran, dass es nie gelöst wird.

Weiterhin glaubten 61 der Befragten, dass **P** $\neq$ **NP** gilt, nur 9 glaubten **P** = **NP**, der Rest traute sich nicht. . .
2012 wurde die Umfrage wiederholt mit i.w. gleichem Ergebnis.

Independent Set ist so schwer wie Vertex Cover



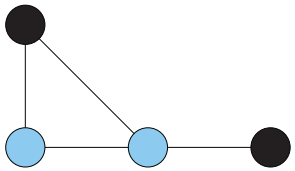
Vertex Cover Independent Set

Einfache Beobachtung: Die Lösungsmengen sind „komplementär“ zueinander („aus schwarz wird blau“ und umgekehrt).

Dominating Set ist mindestens so schwer wie Vertex Cover

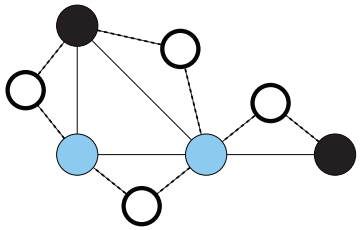
Kann man Dominating Set effizient lösen, so auch Vertex Cover!

Vertex Cover



Knotenmenge V
Kantenmenge E
k

Dominating Set



Knotenmenge $V \cup \{v_e \mid e \in E\}$
Kantenmenge $E \cup \{\{v_e, x\} \mid e \in E, x \in e\}$
k

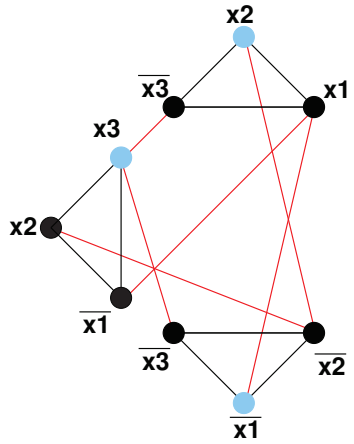
Die Lösung der Dominating Set-Instanz rechts liefert eine Lösung der Vertex Cover-Instanz links!

Independent Set ist mindestens so schwer wie 3-SAT

Konstruktion

- Klausel $F_i := l_{i1} \vee l_{i2} \vee l_{i3} \rightsquigarrow$ Dreieck mit Knoten l_{i1}, l_{i2}, l_{i3}.
- Knoten aus verschiedenen Dreiecken werden durch eine Kante verbunden, wenn sich die entsprechenden Literale „widersprechen“, also eines Negation des anderen ist.
- $k := m$, d.h. Anzahl der Klauseln.

Beispiel:
$(x1 \vee x2 \vee \overline{x3}) \wedge (\overline{x1} \vee x2 \vee x3) \wedge (\overline{x1} \vee \overline{x2} \vee \overline{x3})$



Das Reduktionskonzept und NP-Vollständigkeit

In vorangehenden Beispielen haben wir die Lösung eines Entscheidungsproblems auf die Lösung eines anderen Entscheidungsproblems „**reduziert**“ (besser: zurückgeführt). Folgendes war dabei wichtig:

1. Die „Zurückführung“ („Übersetzung“ des einen Problems in das andere) ist in polynomieller Laufzeit (also effizient) berechenbar.
2. Die Eingabe des einen Problems ergibt dann und nur dann die Antwort „ja“ wenn es auch die neu berechnete Instanz des neuen Problems tut.

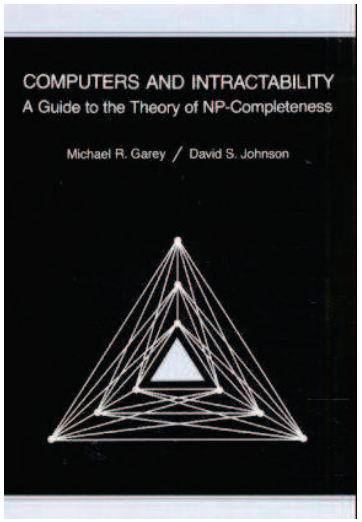
Ein Entscheidungsproblem A heißt **NP-vollständig**, wenn es

1. in NP liegt und
2. jedes andere Probleme in NP sich vermöge einer wie oben beschriebenen Reduktion auf A zurückführen lässt.

Intuition: Die NP-vollständigen Probleme sind die schwersten Probleme in NP! Dazu gehören: TSP, 3-SAT, 3-Coloring, Vertex Cover etc.

Die Bibel der NP-Vollständigkeit

Das bisher wertbeständigste aller wissenschaftlichen Informatikbücher (seit 1979):



Heuristiken

Nach Gerd Gigerenzer:
Welche Stadt hat mehr Einwohner: San Antonio oder San Diego?

Heureka! (Ich hab's gefunden!)

Auch eine Heuristik...

Umgangssprachlich ist oft auch von „Daumenregeln“ oder „Faustregeln“ die Rede.
Vgl. auch „Intuition“ und „Bauchentscheidungen“.

Was sind Heuristiken?

Wikipedia: Heuristik (altgriechisch heurísko, ich finde, zu heuriskein, (auf)finden, entdecken) bezeichnet die Kunst, mit begrenztem Wissen und wenig Zeit zu guten Lösungen zu kommen.

Heuristiken kommen in allen möglichen Wissenschaftsgebieten vor:

- Wirtschaftswissenschaften: Zur Entscheidungsfindung.
- Philosophie: Gewinnung von Erkenntnis über eine Sache durch Übertragung von Wissen von einer anderen, ähnlichen Sache.
- Psychologie: Einfache Regeln für komplexe Situationen; Objekterkennung.
- Chemie: Verständnis und Klassifikation gewisser chemischer Reaktionen; Vorhersage.
- Mathematik: Z.B. Nullstellenraten bei Polynomen; Raten einer Anfangslösung in Optimierungsproblemen.
- Informatik: Künstliche Intelligenz; Intelligente Suche;...

Heuristiken aus Informatiksicht

Zwei zentrale Heuristik-Ansätze:

- Sukzessiver Aufbau einer Lösung beginnend mit „leerer“ Lösung.
- Iteratives Verbessern einer Anfangslösung (z.B. einer trivialen, noch schlechten Lösung).

Motivation: Berechnungsschwere (wie z.B. NP-vollständige) Probleme haben hohe Worst-Case-Komplexität bei ihrer Lösung. Kann man sie trotzdem (oft) gut und schnell in der Praxis lösen?

Heuristiken verzichten in der Regel auf mindestens eine der zwei folgenden Eigenschaften eines Lösungsalgorithmus:

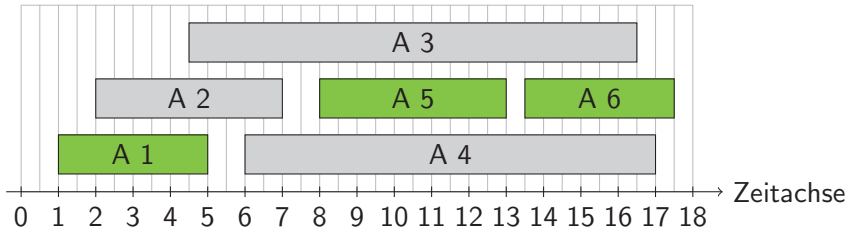
- Beweisbare Optimalität der gefundenen Lösung.
- Beweisbar schnelle Laufzeit des Algorithmus für *jede* Eingabe.

Bemerkung: Inbesondere durch den Einsatz von Heuristiken und ihrer empirischen Bewertung wird Informatik auch zur experimentellen Wissenschaft!

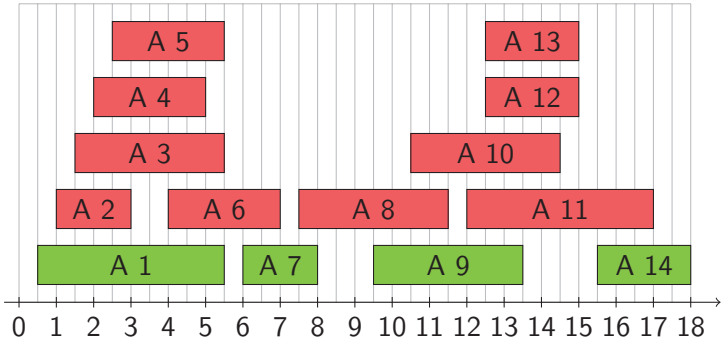
Viele heuristische Verfahren der Informatik sind Alltags- oder Naturprinzipien nachempfunden.

Einstiegsbeispiel Interval Scheduling (Ablaufplanung)

Eingabe: Eine Menge von „Jobs“, jeder mit einer Start- und Endzeit.
Aufgabe: Arbeite so viele Jobs wie möglich ab, wobei immer nur eine Aufgabe auf einmal abgearbeitet wird.



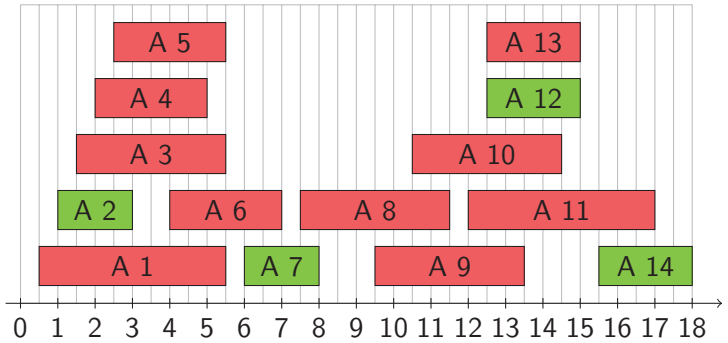
Greedy-Heuristiken für Interval Scheduling I



Größere Instanz: Wie finde ich eine beste Lösung?

Strategie 1: Nimm' Job mit frühester Startzeit.
$\rightsquigarrow$ Vier Jobs können bearbeitet werden.

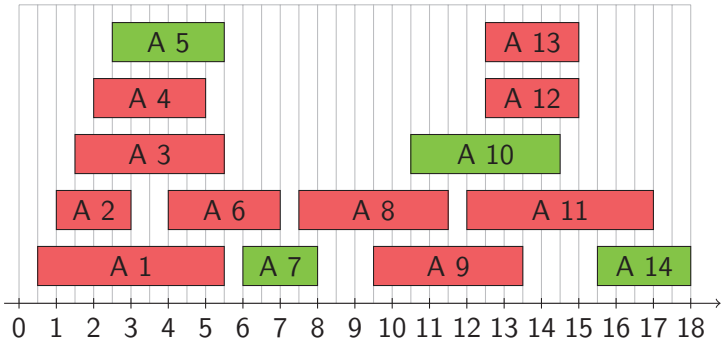
Greedy-Heuristiken für Interval Scheduling II



Größere Instanz: Wie finde ich eine beste Lösung?

Strategie 2: Nimm Job mit kürzester Länge.
$\rightsquigarrow$ Vier Jobs können bearbeitet werden.

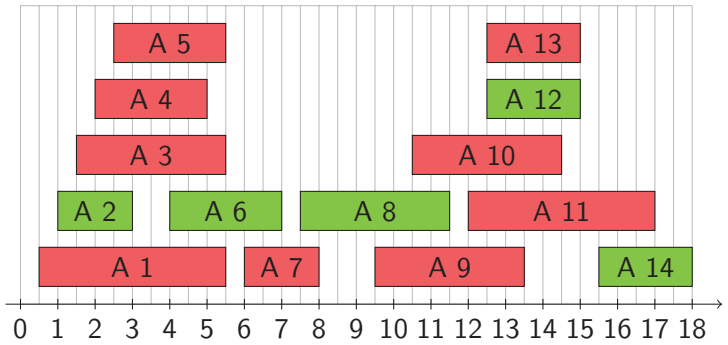
Greedy-Heuristiken für Interval Scheduling III



Größere Instanz: Wie finde ich die eine Lösung?

Strategie 3: Nimm' Job mit wenigsten Überschneidungen.
$\rightsquigarrow$ Vier Jobs können bearbeitet werden.

Greedy-Heuristiken für Interval Scheduling IV



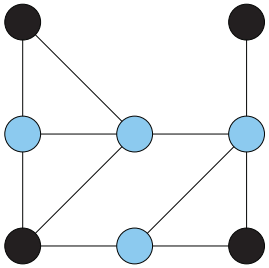
Größere Instanz: Wie finde ich eine beste Lösung?

Strategie 4: Nimm' Job mit frühester Endzeit.
$\rightsquigarrow$ Fünf Jobs können bearbeitet werden.
Mitteilung: Strategie 4 liefert beweisbar immer eine optimale Lösung.

Einstiegsbeispiel Vertex Cover

Vertex Cover,
formuliert als Optimierungs- statt als Entscheidungsproblem:

Finde kleinstmögliche Knotenmenge in Graph, sodass jede Kante mindestens einen dieser Knoten als Endpunkt hat.



Zwei einfache Greedy-Heuristiken:

1. Solange es eine Kante gibt, wähle irgendeine und nimm' *beide* Endpunkte in die Lösungsmenge.
2. Nimm jeweils Knoten mit der höchsten Anzahl anliegender Kanten.

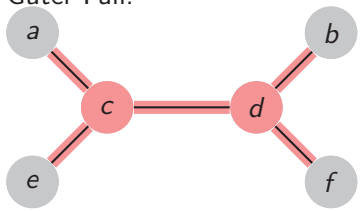
„Offensichtlich" sind beide Verfahren einfach und effizient implementierbar! **Erinnerung:** Die Entscheidungsvariante von Vertex Cover ist NP-vollständig, d.h. es gibt (wahrscheinlich) keinen effizienten Algorithmus zum Finden einer kleinstmöglichen Knotenmenge...

Analyse der Greedy-Heuristik 1 für Vertex Cover

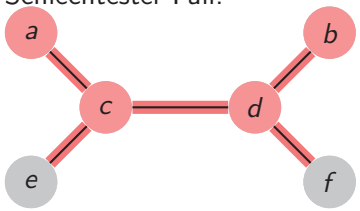
Greedy-Heuristik 1: Solange es eine Kante gibt, wähle irgendeine und nimm' beide Endpunkte in die Lösungsmenge.

Beobachtung: „Faktor-2-Approximation".

Guter Fall:



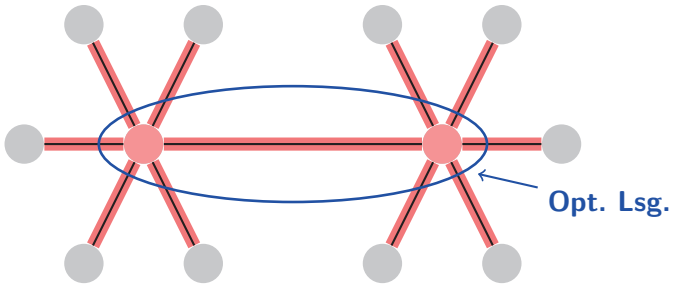
Schlechtester Fall:



Analyse der Greedy-Heuristik 2 für Vertex Cover I

Greedy-Heuristik 2: Nimm' jeweils Knoten mit der höchsten Anzahl anliegender Kanten.

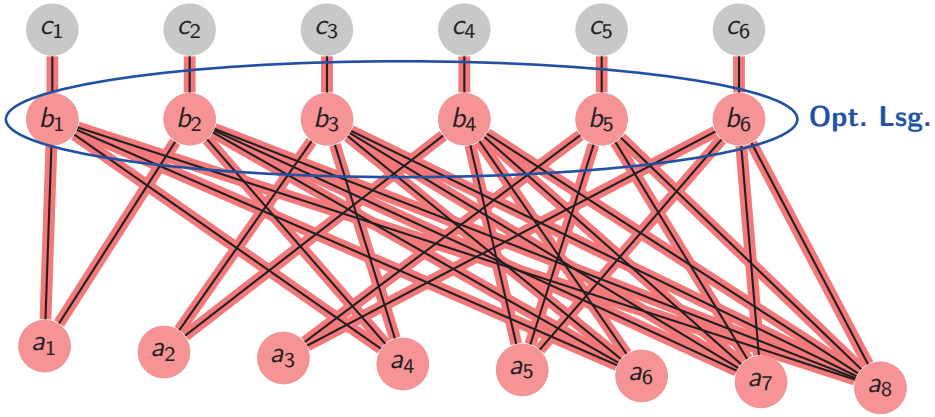
Guter Fall:



Analyse der Greedy-Heuristik 2 für Vertex Cover II

Greedy-Heuristik 2: Nimm jeweils Knoten mit der höchsten Anzahl anliegender Kanten.

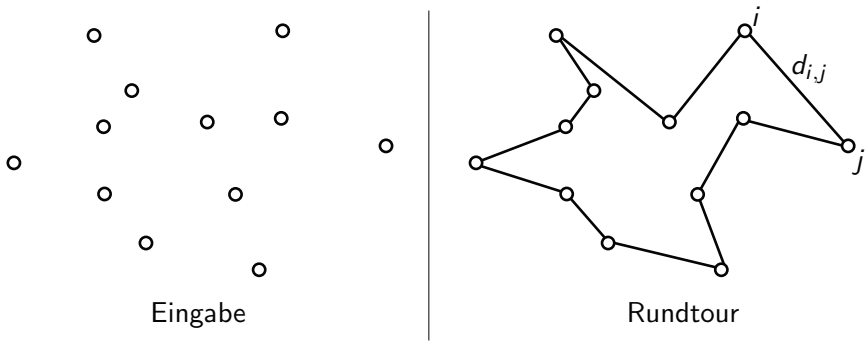
Beobachtung: Bei n Knoten „Faktor-ln n-Approximation“.



Traveling Sales Person (TSP) als fortlaufendes Beispiel

Eingabe: n Punkte mit paarweisen Abständen $d_{i,j}$.
Aufgabe: Finde eine kürzeste Rundtour, die alle Punkte genau einmal besucht.

Formal: Finde eine Permutation π (Rundtour) der Zahlen $1, 2, \ldots, n$, sodass $\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$ minimal ist.



Anwendungen des TSPs

- Routenplanung (Logistik)
- DNS-Sequenzierung (Bioinformatik)
- Layoutfindung für integrierte Schaltkreise (Hardware-Entwurf)
- Steuerung von Robotern in der Industrie (z. B. kürzeste Rundtour durch alle Lötstellen)
- ...

Erinnerung: Die Entscheidungsvariante des TSP ist NP-vollständig, d. h. es gibt (wahrscheinlich) keinen effizienten Algorithmus zum Finden optimaler Rundtouren.

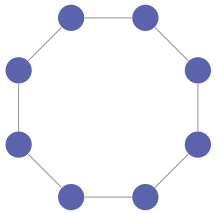
Greedy-Heuristiken für das TSP I

Annahme in folgenden Beispielen: Euklidsche Abstände.

Nächster-Nachbar-Heuristik:

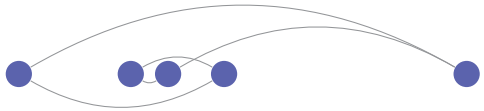
1. Wähle beliebigen Startpunkt. Dieser heiße p.
2. Solange es einen noch nicht besuchten Punkt gibt und p der zuletzt besuchte Punkt ist:
   - Wähle als nächsten zu besuchenden Punkt den zu p am nahesten liegenden, unbesuchten Punkt.
   - Mache diesen Punkt zum neuen p.
3. Die Tour ergibt sich aus der Abfolge der besuchten Punkte.

Einfache und effiziente Heuristik!
Aber immer optimal? **Beispiel**:



Greedy-Heuristiken für das TSP II

Beispiel:
Nächster-Nachbar-Heuristik findet folgende Rundtour bei Start im mittleren Punkt.



Optimal wäre aber offensichtlich folgende Rundtour:

Greedy-Heuristiken für das TSP III

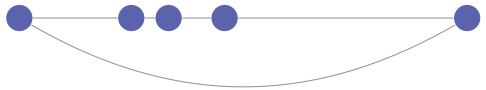
Eine zweite einfache Heuristik:

Annahme: Rundtour durch n Punkte wird durch einen Kantenzug beschrieben. Trivialer Kantenzug besteht aus einem Punkt.

Engstes-Paar-Heuristik:

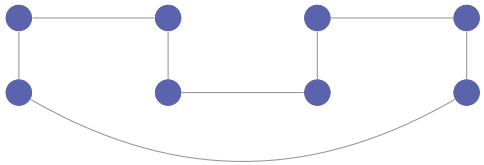
1. Solange es mehr als einen Kantenzug gibt
   - Verbinde zwei Endpunkte s und t zweier *verschiedener* Kantenzüge, sodass $d_{s,t}$ minimal unter allen möglichen Punktepaaren ist.
2. Verbinde die zwei Endpunkte des „Gesamtpfades“.

Beispiel:



Greedy-Heuristiken für das TSP IV

Wie gut ist die Engstes-Paar-Heuristik?
Beispiel: Durch die Heuristik gefundene Lösung:



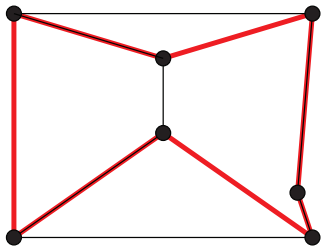
Optimal aber wäre:

Greedy-Heuristiken für das TSP V

Inkrementelles-Einfügen-Heuristik:
Baue Punkt für Punkt die Rundtour auf, indem man

1. für jeden noch nicht besuchten Punkt den minimalen Abstand zu zwei benachbarten Punkten der bisherigen Teiltour bestimmt, und
2. den Punkt mit dem größten minimalen Abstand einfügt („furthest point insertion“).

Intuition: „Erst Grobtour skizzieren, dann Details klären.“
Nicht optimal! Denn z.B. rot markierte Tour ist kürzer.



Euklidsches TSP I

Wichtiger und „angenehmerer“ Spezialfall[6] des TSP:
Alle Punkte liegen in der (Euklidschen) Ebene und ihr paarweiser Abstand ergibt sich aus den Entfernungen, d.h. für je drei beliebige Punkte i, j, und k gilt inbesondere die **Dreiecksungleichung**:

$$d_{i,j} \leq d_{i,k} + d_{k,j}.$$

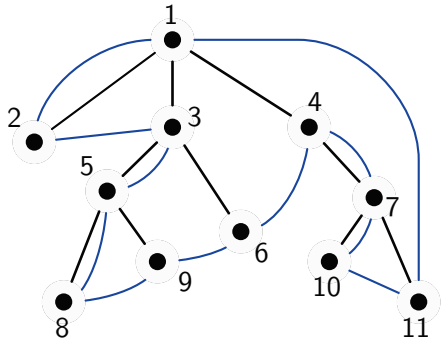
Faktor-2-Approximation mit Hilfe kostenminimaler Spannbäume:
Kostenminimaler Spannbaum = Baum welcher alle Eingabepunkte verbindet und die *Summe* der „Kantenkosten“ (d.h. die jeweilige Entfernung zwischen den zwei Kantenendpunkten) ist kleinstmöglich.

Hinweis: Kostenminimale Spannbäume lassen sich effizient berechnen (vgl. Algorithmen von Boruvka, Kruskal, Prim).

[6] Zugehöriges Entscheidungsproblem bleibt aber weiterhin NP-vollständig.

Euklidsches TSP II

1. Berechne kostenminimalen Spannbaum.
2. Laufe den Spannbaum gemäß Tiefensuche ab.
3. Mache Abkürzungen falls Knoten doppelt besucht werden.



Tiefensuche: 1–2–1–3–5–8–5–9–5–3–6–3–1–4–7–10–7–11–7–4–1
Beobachtung: Manche Punkte treten doppelt auf → Abkürzungen!
⇝ Rundtour: 1–2–3–5–8–9–6–4–7–10–11–1

Euklidsches TSP III

Obige Spannbaumheuristik liefert Faktor-2-Approximation, d.h. die gefundene Tour ist höchstens doppelt so lang wie eine optimale:

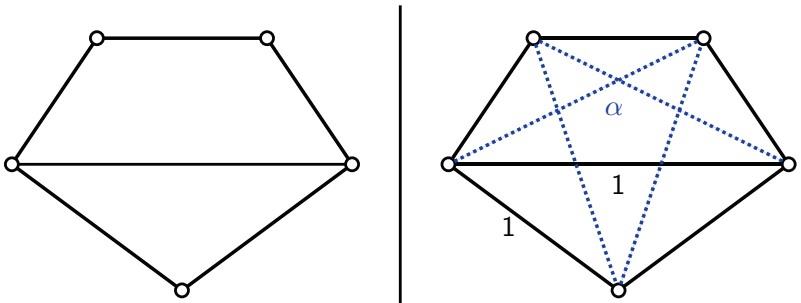
1. Kosten eines Spannbaumes sind höchstens so groß wie die Länge einer optimalen Rundtour! (Warum?)
2. Beim Ablaufen des Spannbaumes wird jede Kante höchstens zweimal überquert (vgl. Tiefensuche), deshalb liefert dies eine Tour die höchstens doppelt so lang ist wie eine optimale Rundtour.
3. Dank Dreiecksungleichung verlängert Abkürzen (sic!) die Tour nicht!

Begründung für Punkt 1: Eine optimale Rundtour ist ein Kreis. Entfernt man eine beliebige Kante so erhält man einen Spannbaum. Wir haben jedoch einen kostenminimalen Spannbaum gewählt...

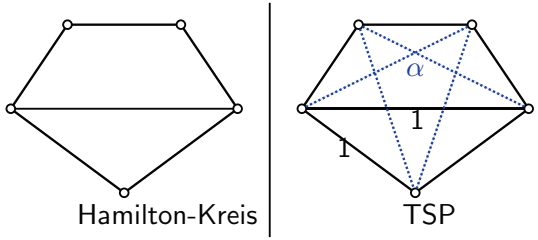
Nichtapproximierbarkeit des TSPs I

Die Approximierbarkeit in polynomieller Laufzeit des allgemeinen TSPs ist dramatisch schlechter als die des Euklidschen TSPs:

Beweis vermöge „Reduktion” vom NP-vollständigen Hamilton-Kreis-Problem auf TSP:
Erzeuge aus einer Hamilton-Kreis-Instanz $G = (V, E)$ eine TSP-Instanz vermöge $d_{i,j} = 1$ falls $\{v_i, v_j\} \in E$ und $d_{i,j} = \alpha > 1$ andernfalls.



Nichtapproximierbarkeit des TSPs II

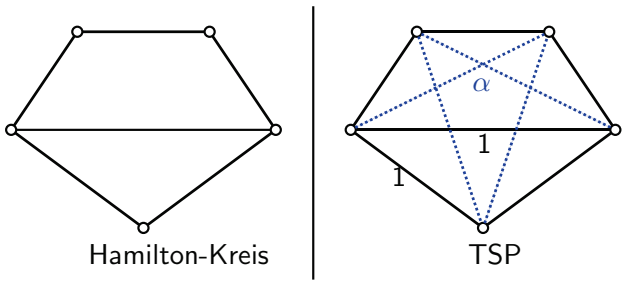


Behauptung: Die konstruierte TSP-Instanz hat eine Rundtour der Länge höchstens $n := |V|$ genau dann, wenn G einen Hamilton-Kreis hat. (Warum?)

Begründung: Ein Kreis enthält immer genau n Kanten, d. h. hat Mindestlänge n. Da $\alpha > 1$ entspricht jede Rundtour der Länge n einem Hamilton-Kreis im (Original-) Graph und umgekehrt.

Beobachtung: Für entsprechende Wahl von α (z. B. $\alpha = n^2$) kann das Verhältnis Länge Rundtour mit „α-Kante“ zu Rundtour ohne „α-Kante“ nicht durch eine Konstante beschränkt werden.

Nichtapproximierbarkeit des TSPs III



Konsequenz aus obiger Reduktion: Könnten wir für das TSP einen effizienten Approximationsalgorithmus mit Approximationsfaktor kleiner als α angeben, so könnten wir das Hamilton-Kreis Problem effizient lösen.

Unter der Hypothese P $\neq$ NP folgern wir damit die Nichtapproximierbarkeit des allgemeinen TSPs.

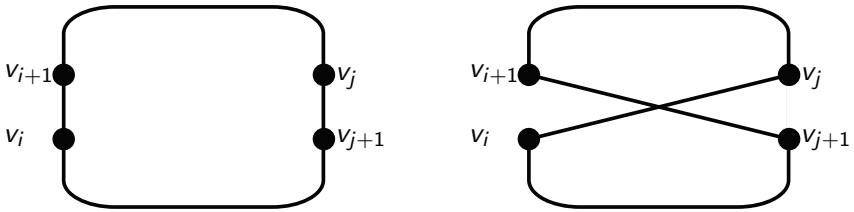
Bemerkung: Die Kosten für Flugtickets erfüllen z.B. in der Regel nicht die Dreiecksungleichung.

TSP und Lokale Suche

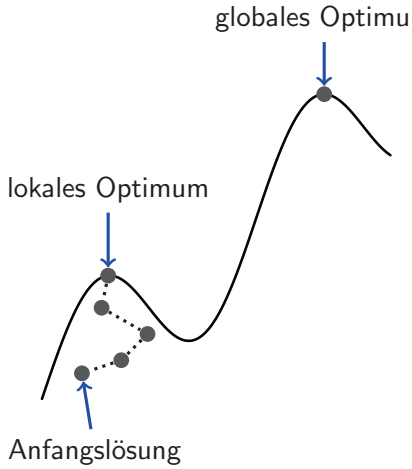
Lokale-Suche-Heuristik: Verbessere schrittweise eine gegebene Rundtour durch „kleine lokale Änderungen“.

Idee: Suche innerhalb der k-Opt-Nachbarschaft nach einer kürzeren Rundtour.

k-Opt: Lösche bis zu $k \in \mathbb{N}$ Kanten innerhalb der Rundtour und füge stattdessen k neue Kanten ein, um wieder eine Rundtour zu erzeugen.



Allgemeines Prinzip der Lokalen Suche



Strategie:

1. Erzeuge Anfangslösung L
2. Solange es innerhalb der „lokalen Umgebung“ von L eine Verbesserung gibt:
   $L \leftarrow$ lok. verbesserte Lsg.

Problem: Algorithmus kann in lokalem Optimum hängenbleiben.
Lösung: Wiederhole Alg. für verschiedene Anfangslösungen oder **Simulated Annealing**...

TSP und Simulated Annealing, Metropolis-Algorithmus

„Simuliertes Abkühlen“:
Idee analog zur lokalen Suche, nur dass mit einer gewissen Wahrscheinlichkeit auch schlechtere Lösungen übernommen werden. Diese Wahrscheinlichkeit nimmt im Laufe des Algorithmus ab und geht gegen Null.

Simulated Annealing [Metropolis et al. 1953]

```
1  Erzeuge Anfangslösung L
2  s ← 0                    ▷ Anzahl Schritte
3  while s ≤ max. Anzahl Schritte
4      s ← s + 1
5      Erzeuge aus L eine zufällig veränderte (bzw. mutierte) Lösung L'
6      if L' besser als L oder e^{-s} ≥ random[0,1] then:
7          L ← L'
```


Fazit zum TSP

Bei der algorithmischen Lösung von TSP-Instanzen gilt es diverse Randbedingungen zu berücksichtigen:

- Der Spezialfall dass alle Punkte paarweise gleichen Abstand haben entspricht dem Hamilton-Kreis-Problem.
- Erfüllt die Eingabe die Dreiecksungleichung (was sich immer erreichen lässt wenn die Eingabepunkte mehrfach besucht werden dürfen...), so lässt sich das TSP (beweisbar) gut approximativ lösen.
- Ist die paarweise Abstandsfunktion asymmetrisch (d.h. $d_{i,j} \neq d_{j,i}$), so ist das TSP in der Regel viel schwerer zu handhaben als im symmetrischen Fall (d.h. $d_{i,j} = d_{j,i}$).
- Es gibt viele gute Heuristiken, die oft nahe am Optimum liegende Lösungen schnell liefern.

Genetische Algorithmen I

Teilthema der Evolutionären Algorithmen...:

Inspiriert durch biologische Evolutionstheorie wurden in den 60'ern und 70'ern **evolutionäre Algorithmen** eingeführt. Maßgeblicher Einfluss (i.w. unabhängig voneinander) durch:

- Lawrence J. Fogel (Univ. of California, LA; 1928-2007) : evolutionäres Programmieren.
- John H. Holland (U. Michigan; 1929-): genetische Algorithmen.
- **Ingo Rechenberg (TU Berlin)** & Hans-Paul Schwefel (TU Dortmund; 1940-): Verfahren zur Evolutionsstrategie.

Ingo Rechenberg, 1934-

Genetische Algorithmen II

Idee: Nicht einzelne Lösungen verändern, sondern eine **Population** von Lösungen verändern.
Veränderungen durch **Mutationen** und **Kreuzungen**: Repräsentation der Lösungen als Bitfolgen.

Mutation: Zufälliges Stören/Flippen einzelner Bits.

$$a_i = 110010100101$$
$$a_i^* = 1100\mathbf{0}0100\mathbf{1}0\mathbf{0}$$

Kreuzung (engl. *cross-over*): Aufteilen am Cross-Over-Punkt und Kreuzen der Teile.

$$a_i = 110010|100101 \qquad a_j = 010101|010110$$
$$\Rightarrow a_i^* = 110010|010110 \qquad a_j^* = 010101|100101$$

Genetische Algorithmen III

Nötig: **Fitnessfunktion** zur Bewertung von Lösungen.

Schema

1 Erzeuge Anfangspopulation $\{L_1, L_2, \ldots, L_r\}$
2 **Repeat**
4 Erzeuge Mutationen von $\{L_1, \ldots, L_r\}$
5 Erzeuge Kreuzungen von $\{L_1, \ldots, L_r\}$
6 Behalte die r fittesten (besten) Lösungen
7 **UNTIL** Fitness verbessert sich nicht mehr

Effiziente heuristische Suche: Der A*-Algorithmus

Der A*-Algorithmus dient zur effizienten Suche in Graphen: Wie finde ich einen optimalen Weg von A nach B?
Optimal kann bedeuten: kürzester, schnellster, billigster, etc.

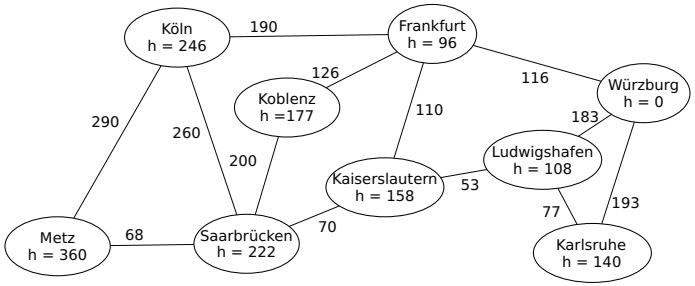
Idee: Benutze Zusatzinformationen als untere Schranke für „Entfernung“ eines Knotens zum Ziel um in der „richtigen Richtung“ zu suchen.

Erklärung am Beispiel: Berechnen des kürzesten Weges von Knoten A nach Knoten B.
Jeder Knoten x speichert als untere Schranke $h(x)$ die Entfernung (Luftlinie) zum Ziel.

Algorithmus:

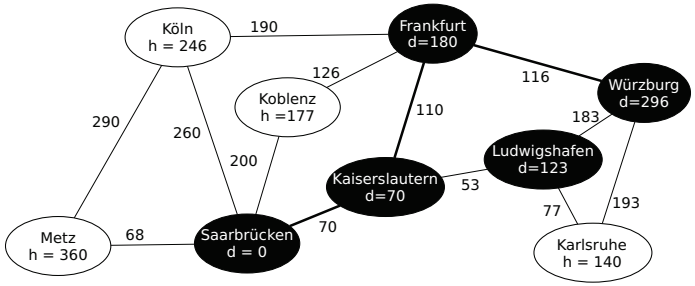
1 Merke alle besuchten Knoten mit berechneter Distanz zu A.
2 Starte in Knoten A.
3 **Repeat**
4 Besuche den Nachbar x eines bereits besuchten Knotens sodass (Distanz x zu A) + ($h(x)$) minimiert wird.
5 **UNTIL** B erreicht

Der A*-Algorithmus: Beispiel I



Gegeben ist obiger Ausschnitt einer Karte.
Zahlen in den Knoten geben die Luftliniendistanz (untere Schranke) zu Würzburg an. Zahlen auf den Kanten sind tatsächliche Entfernungen entlang der jeweiligen Kante.
Gesucht: Kürzeste Strecke von Saarbrücken nach Würzburg.

Der A*-Algorithmus: Beispiel II



Starte in Saarbrücken.
Besuche Kaiserslautern: Distanz + untere Schranke = 70 + 158 = 228.
Besuche Ludwigshafen: Distanz + untere Schranke = 123 + 108 = 231.
Besuche Frankfurt: Distanz + untere Schranke = 180 + 96 = 276.
Besuche Würzburg: Distanz + untere Schranke = 296 + 0 = 296.
Kürzester Weg: Saarbrücken → Kaiserslautern → Frankfurt → Würzburg.

Gerechtigkeit (?!)

Oft konkurrieren mehrere „Agenten“ um die gleichen Ressourcen (vgl. z.B. Scheduling-Beispiele).

Wie erreicht man eine faire Ressourcenverteilung?
Was heißt überhaupt fair?
Welche entsprechenden „Verteilungsprotokolle“ sind nützlich?
Wie beugt man Manipulationen und Betrügereien vor?

Dies sind zentrale Fragen in vielen Kontexten, die in der Informatik insbesondere zum Entwurf und der Analyse entsprechender **Protokolle** (z.B. im Internet) führen.

Nachfolgend zwei Aspekte näher beleuchtet:
Gerechte Aufteilung...

- ...unteilbarer Ressourcen ($\rightsquigarrow$ Scheidungsformel, ...) und
- ...teilbarer Ressourcen ($\rightsquigarrow$ Cake Cutting, ...).

Was ist gerecht?

Wikipedia: Der Begriff der Gerechtigkeit bezeichnet einen idealen Zustand des sozialen Miteinanders, in dem es einen angemessenen, unparteilichen und einforderbaren Ausgleich der Interessen und der Verteilung von Gütern und Chancen zwischen den beteiligten Personen oder Gruppen gibt.

Fairness in der Informatik

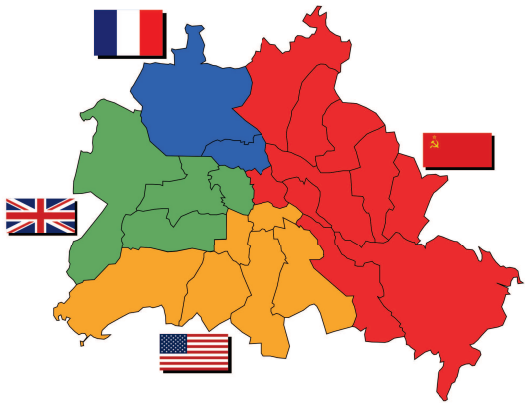
In der Informatik spielen Konzepte der Fairness (und sie realisierende Protokolle und Algorithmen) eine gewichtige Rolle in Gebieten wie

- Scheduling,
- Betriebssysteme,
- verteilte Systeme,
- Computernetze,
- Algorithmische Spieltheorie,
- Computational Social Choice.

Gerechtes Aufteilen

Gerechtes Teilen begleitet uns lebenslänglich (nicht nur in der Informatik), z.B.

- Kindergarten,
- Rechnerkapazitäten,
- Grundstücke.



Aufteilung unteilbarer Ressourcen

Ausgangslage: Mehrere unteilbare Ressourcen sind zu verteilen.

Unteilbar sind die Ressourcen dabei in zweierlei Hinsicht:

- Keine Ressource kann in kleinere Teilstücke zerlegt werden („indivisible“) und
- Keine Ressource kann mehr als einem Agenten zugeordnet werden – verschiedene Agenten können also nicht dieselbe Ressource gemeinsam nutzen („nonshareable“).

Beispiel: Gemeinsames Haustier nach Scheidung muss einem Partner zur alleinigen „Nutzung“ zugeteilt werden.

Aufteilung einzelner Güter bei Scheidung

Beispiel: Ein Paar trennt sich... Streit um die folgenden Dinge, die gemeinsam angeschafft wurden:

- ein Auto,
- eine kostbare
- sieben (

Wie teilt man die Dinge gerecht auf?

Scheidungsformel („Adjusted Winner Procedure“) I

Brams und Taylor (1996):

- Zunächst verkünden beide Parteien ihre jeweilige subjektive Bewertung der aufzuteilenden Objekte, wobei beide jeweils insgesamt 100 Punkte auf diese Objekte verteilen dürfen.

Anzahl	Objekt	Partei 1	Partei 2
1	Auto	10	9
1	Buddha	13	28
7	Gartenzwerg	77	63
Insgesamt		87	28

- Zuerst geht jedes Objekt an die Partei, der es mehr wert ist.
- Die aktuelle Gewinnerin Partei 1 muss so viele ihrer Objekte an Partei 2 abgeben, bis ein Ausgleich erzielt wird. Betrachte dazu die Verhältnisse der jeweiligen Bewertungen (≥ 1), nach Größe sortiert:

Objekt	Auto	Zwerg	Zwerg	Zwerg	Zwerg	Zwerg	Zwerg	Zwerg	Buddha
Verhältnis	10/9	11/9	11/9	11/9	11/9	11/9	11/9	11/9	28/13

Scheidungsformel („Adjusted Winner Procedure“) II

Subjektive Bewertung der aufzuteilenden Objekte:

Anzahl	Objekt	Partei 1	Partei 2
1	Auto	10	9
1	Buddha	13	28
7	Gartenzwerg	77	63
Bekommt		55	55

Verhältnis der Bewertungen der beiden Parteien

Objekt	Auto	Zwerg	Zwerg	Zwerg	Zwerg	Zwerg	Zwerg	Zwerg	Buddha
Verhältnis	10/9	11/9	11/9	11/9	11/9	11/9	11/9	11/9	28/13

- Partei 1 gibt Reihe nach Objekte dieser Liste an Partei 2 bis Ausgleich erreicht wird.
- Auto , Zwerg , Zwerg.
- Beide Parteien haben nun 55 Punkte.

Scheidungsformel („Adjusted Winner Procedure“) III

Eigenschaften Scheidungsformel:
Diese Methode hat eine Reihe von Vorzügen. Zum Beispiel ist die durch sie erzielte Aufteilung

- **pareto-optimal**, d.h. jede Zuweisung der Objekte, die eine der beiden Parteien besser stellt, würde die andere schlechter stellen;
- **gleichverteilt**, d.h. beide Parteien bewerten die ihnen zugeteilten Objekte insgesamt gleich, und
- **neidfrei**, d.h. keine der beiden Parteien würde die ihr zugeteilten Objekte mit denen der anderen tauschen wollen.

Nichttrivialer Beweis!
Problem: Zuweisung der Objekte nicht immer ganz gleichmäßig möglich!

Aufteilung teilbarer Ressourcen

Ausgangslage: Eine teilbare Ressource (z.B. Kuchen) ist zwischen n Spielern gerecht aufzuteilen, wobei jeder Spieler potenziell eine andere Bewertungsfunktion hat.
Der Kuchen als Symbol fuer eine beliebig teilbare Ressource kann als reelles Zahlenintervall $[0, 1]$ repräsentiert werden,

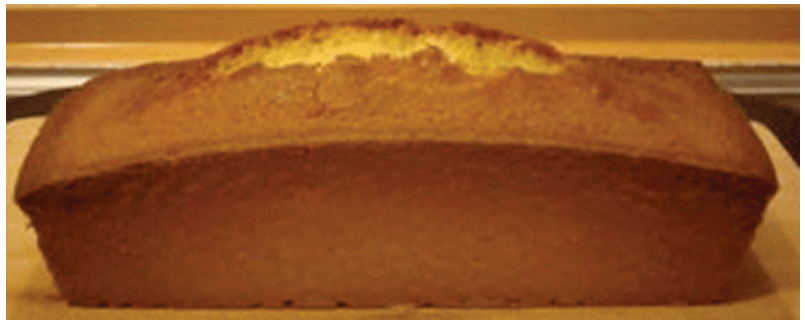
Ziel: Teile Kuchen X durch Schnitte in n Portionen X_i, $1 \leq i \leq n$, sodass der $i$te Spieler Portion X_i erhält.

Nebenbedingungen: Die Vorlieben der Spieler werden durch jeweils „privates“ Maß, d.h. eine individuelle Bewertungsfunktion ausgedrückt und fließen nachfolgend mit ein.

Zwei wünschenswerte **Fairnessbedingungen**:

1. **Proportional:** Jeder Spieler findet, dass er mindestens $1/n$ des Kuchens bekommen hat.
2. **Neidfrei:** Kein Spieler findet das Stück eines Gegenspielers (echt) besser als sein eigenes.

Teilbare Ressourcen: $[0, 1]$ oder...



Teilbare Ressourcen: $[0, 1]$ oder...

Teilbare Ressourcen: $[0, 1]$ oder...

Teilbare Ressourcen: $[0, 1]$ oder...

Kuchenschneiden (Cake Cutting)

Problem: Unterschiedliche Präferenzen der einzelnen Spieler.
(Bei homogenen Präferenzen (alle gleich) ist das Problem trivial.)

Beispiel Kuchen: Lieber mehr Sahne oder (k)eine Kirsche oder...?

Kann man es trotzdem allen Recht machen?

Ein einfaches Protokoll für zwei Spieler: **Cut-and-Choose**:
Ein Spieler schneidet, der andere wählt eines der beiden Stücke.

Dieses Protokoll ist proportional und neidfrei.

Mitteilung: Im Falle von nur zwei Spielern sind Proportionalität und Neidfreiheit äquivalente Begriffe; ab drei Spielern gilt dies nicht mehr, sondern nur noch, dass Neidfreiheit Proportionalität impliziert.

Cake-Cutting-Protokolle

Steinhaus-Protokoll für drei Spieler:

1. Spieler 1 schneidet den Kuchen in drei Stücke (die er als gleich einschätzt).
2. Spieler 2 „passt“ (falls seiner Meinung nach mind. zwei Stücke $\geq 1/3$ sind) oder markiert zwei Stücke als „schlecht“.
   Falls Spieler 2 gepasst hat: Spieler 3, 2, 1 wählen je ein Stück (in dieser Reihenfolge). Fertig.
3. Falls Spieler 2 nicht gepasst hat: Spieler 3 passt oder markiert.
   Falls Spieler 3 gepasst hat: Spieler 2, 3, 1 wählen je ein Stück (in dieser Reihenfolge). Fertig.
4. Falls weder Spieler 2 noch Spieler 3 gepasst hat: Spieler 1 muss das (oder eines der) Stück(e) nehmen, die von 2 und 3 als „schlecht“ markiert wurden.
   Der Rest wird zusammengeworfen, dann Cut-and-Choose. Fertig.

Eigenschaften:

- proportional
- nicht neidfrei
- kein aktiver Schiedsrichter nötig
- nicht immer zusammenhängende Stücke

Cake-Cutting-Protokolle II

Banach-Knaster Last-Diminisher-Protokoll für n Spieler:

1. Spieler 1 schneidet ein Stück ab (das seiner Meinung nach $1/n$ entspricht).
2. Dieses Stück wird nun weitergereicht. Jeder Spieler lässt es entweder weitergehen (wenn er es zu klein findet) oder stutzt es weiter (so dass es danach seiner Meinung nach $1/n$ entspricht).
3. Wenn das Stück einmal bei allen Spielern war, muss der Spieler, der als letzter etwas abgeschnitten hat ("last diminisher") es nehmen.
4. Der Rest des Kuchens (zusammen mit den abgeschnittenen Teilen) wird unter den restlichen $n-1$ Spielern verteilt wie eben.
   Cut-and-Choose sobald $n = 2$.

Eigenschaften:

- proportional
- nicht neidfrei
- kein aktiver Schiedsrichter nötig
- nicht immer zusammenhängende Stücke $\longrightarrow$ Verbesserung möglich!

Cake-Cutting-Protokolle III

Dubins-Spanier-Protokoll für n Spieler:

1. Ein Schiedsrichter bewegt ein Messer langsam von links nach rechts über den Kuchen. Jeder Spieler darf jederzeit „cut!" rufen. Wer dies macht, bekommt das Stück, das links vom Messer liegt.
2. Wurde ein Stück weggeschnitten, so geht es mit den restlichen $n-1$ Spielern weiter, bis nur einer bleibt (der den Rest bekommt).

Eigenschaften:

- proportional
- nicht neidfrei
- aktiver Schiedsrichter benötigt
- zusammenhängende Stücke, min. Anzahl Schnitte
- kein Algorithmus...

Cake-Cutting-Protokolle IV

Even-Paz Divide-And-Conquer-Protokoll für n Spieler

1. Jeder Spieler muss eine Markierung an der Stelle $\lfloor \frac{n}{2} \rfloor \| \lceil \frac{n}{2} \rceil$ anbringen.
2. Der Teil des Kuchens, der links von der $\lfloor \frac{n}{2} \rfloor$ten Markierung liegt, gehört den Spielern, die die $\lfloor \frac{n}{2} \rfloor$ am weitesten links liegenden Markierungen gemacht haben (Gruppe 1), der Rest den anderen (Gruppe 2).
3. Wende die Prozedur rekursiv in jeder der beiden Gruppen an, bis nur noch ein Spieler übrig ist.

Eigenschaften:

- proportional
- nicht neidfrei
- kein aktiver Schiedsrichter nötig
- zusammenhängende Stücke

Cake-Cutting-Protokolle V

Selfridge-Conway-Protokoll für drei Spieler:

1. Spieler 1 schneidet den Kuchen in drei Stücke (die seiner Meinung nach gleichwertig sind).
2. Spieler 2 „passt" (falls er denkt, dass die beiden für ihn größten Stücke gleichwertig sind) oder stutzt ein Stück (so dass danach die beiden für ihn größten Stücke gleichwertig sind).
   Falls Spieler 2 gepasst hat: Spieler 3, 2, 1 wählen je ein Stück (in dieser Reihenfolge). Fertig.
3. Falls Spieler 2 gestutzt hat: Spieler 3, 2, 1 wählen je ein Stück (in dieser Reihenfolge), aber Spieler 2 muss das gestutzte nehmen (außer 3 hat es schon genommen).
   Das abgeschnittene Stück bleibt erst einmal liegen.
4. Verteile das abgeschnittene Stück: Schneiden muss der Spieler (2 oder 3), der das *un*gestutzte Stück bekommen hat. Dann wird ausgesucht in der Reihenfolge: Nicht-Schneider, Spieler 1, Schneider. Fertig.

Eigenschaften:

- proportional
- **neidfrei!**
- kein Schiedsrichter nötig
- nicht immer zusammenhängende Stücke

Cake-Cutting-Protokolle VI

Stromquist-Protokoll für drei Spieler:

1. Ein Schiedsrichter bewegt ein Messer langsam von links nach rechts über den Kuchen (mit dem Ziel, bei 1/3 zu schneiden).
2. Gleichzeitig bewegt jeder Spieler sein eigenes Messer, so dass es das Stück, das momentan rechts vom Schiedsrichtermesser liegt, halbieren würde (seiner Meinung nach).
3. Der erste Spieler, der „cut!“ ruft, bekommt das Stück, das links vom Schiedsrichtermesser liegt. Das rechte Stück wird vom mittleren der drei Spielermesser geschnitten.
   Falls keiner der beiden anderen Spieler das mittlere Messer hat, bekommen sie jeder das Stück, das ihr Messer angibt.
   Falls einer von beiden das mittlere Messer hat, bekommt der andere das Stück, das sein Messer angibt.

Eigenschaften:

- proportional
- **neidfrei!**
- Aktiver Schiedsrichter wird benötigt.
- zusammenhängende Stücke

Frohe Weihnachten!

Fast...:

Praktische Übung

Anwendung des Dubins-Spanier-Protokolls am essbaren Objekt!

Erinnerung: Dubins-Spanier-Protokoll für n Spieler:

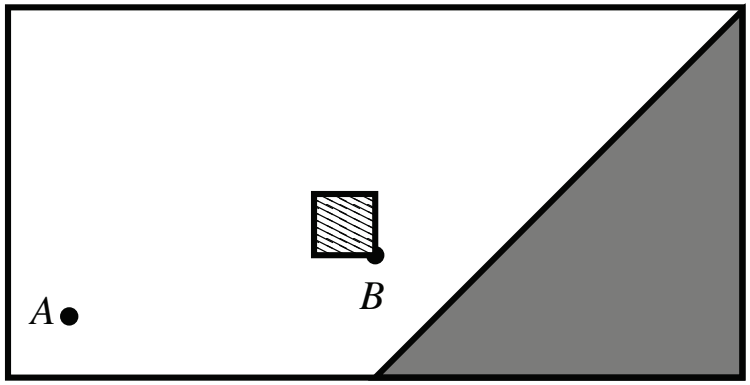
1. Ein Schiedsrichter bewegt ein Messer langsam von links nach rechts über den Kuchen. Jeder Spieler darf jederzeit „cut!“ rufen. Wer dies macht, bekommt das Stück, das links vom Messer liegt.
2. Wurde ein Stück weggeschnitten, so geht es mit den restlichen $n-1$ Spielern weiter, bis nur einer bleibt (der den Rest bekommt).

Literaturhinweis: J. Rothe, D. Baumeister, C. Lindner, I. Rothe: Einführung in Computational Social Choice, Spektrum Akademsicher Verlag 2012.

In der Backstube – Weihnachtsrätsel

Der Weihnachtsmann betreibt unzählige Backstuben. In der Backstube mit der Nummer 182-B72 werden Lebkuchen mit Schokolade überzogen. Die Backstube ist rechteckig und 24m lang und 12m breit; sie liegt tief unter der Erde und kann nur mit dem Lift erreicht werden. Der quadratische Liftschacht hat Seitenlänge 2m, und seine nordöstliche Ecke liegt genau in der Mitte des Raumes; in der folgenden Skizze sehen wir den Liftschacht als kleines schraffiertes Quadrat im Grundriss der Backstube eingezeichnet. In der südöstlichen Ecke des Raumes befindet sich ein riesiges dreieckiges Becken mit flüssiger Schokoladeglasur; Südseite und Ostseite des Beckens sind jeweils 12m lang. Der Punkt A ist 2m von der Südseite und 2m von der Westseite der Backstube entfernt. Der Punkt B ist 4m von der Südseite und 12m von der Westseite der Backstube entfernt (und fällt daher mit der Südostecke des Liftschachts zusammen). An den Wänden der Backstube befinden sich viele Reihen von Regalen, die mit Lebkuchen gefüllt sind.

In der Backstube – Weihnachtsrätsel



In der Backstube – Weihnachtsrätsel

Der Backwichtel Bastian steht im Punkt A der Backstube und liest sich seinen heutigen Arbeitsplan durch. Bastian hat der Reihe nach folgende Aufgaben zu erledigen:

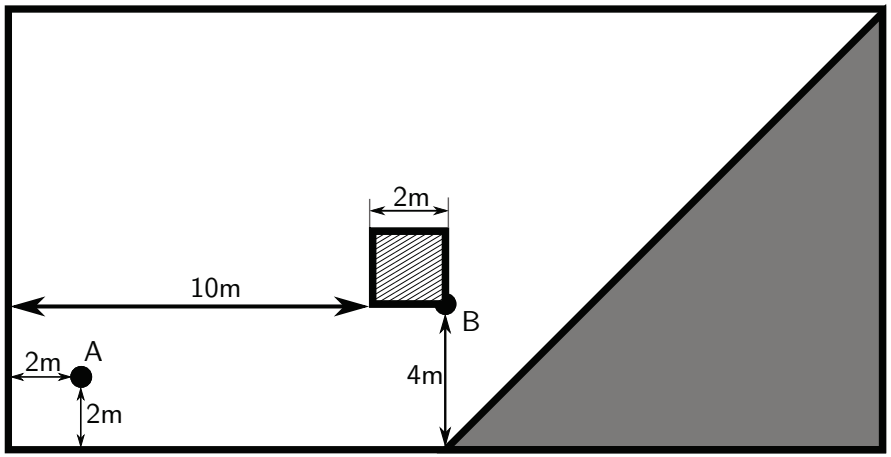
- Zuerst muss Bastian zur Nordseite gehen und sich irgendeinen der Lebkuchen von irgendeinem der oberen Regale nehmen.
- Bastian muss diesen Lebkuchen zum Becken tragen und in die Glasur tauchen.
- Dann muss Bastian den Lebkuchen wieder zur Nordseite tragen und ihn dort an einer beliebigen Stelle in einem beliebigen der unteren Regale ablegen.
- Danach muss Bastian zur Westseite gehen und irgendeinen der dortigen Lebkuchen nehmen.
- Dieser Lebkuchen soll zur Südseite getragen und dort an einer beliebigen Stelle in einem beliebigen Regal abgelegt werden.
- Schliesslich soll Bastian zum Punkt B gehen und in den Lift einsteigen.

In der Backstube – Weihnachtsrätsel

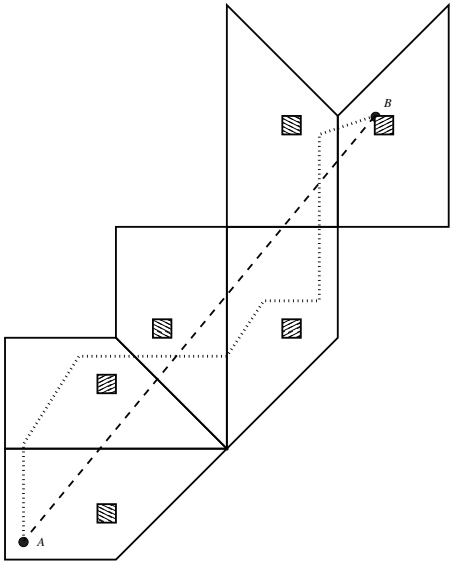
Bastian erledigt seine Aufgaben und geht dabei den kürzest möglichen Weg von A (über Nordseite, Becken, Nordseite, Westseite, Südseite) nach B. Wir nehmen an, dass Bastian punktförmig ist und dass er auf seinem Weg weder durch den Liftschacht hindurchläuft noch ins Schokoladebecken springt.
Wie lange ist der kürzest mögliche Weg?

1. $59,60$m oder weniger
2. zwischen $59,60$ und $59,70$ Meter
3. zwischen $59,70$ und $59,80$ Meter
4. zwischen $59,80$ und $59,90$ Meter
5. zwischen $59,90$ und $60,00$ Meter
6. zwischen $60,00$ und $60,10$ Meter
7. zwischen $60,10$ und $60,20$ Meter
8. zwischen $60,20$ und $60,30$ Meter
9. zwischen $60,30$ und $60,40$ Meter
10. $60,40$m oder mehr

In der Backstube – Weihnachtsrätsel

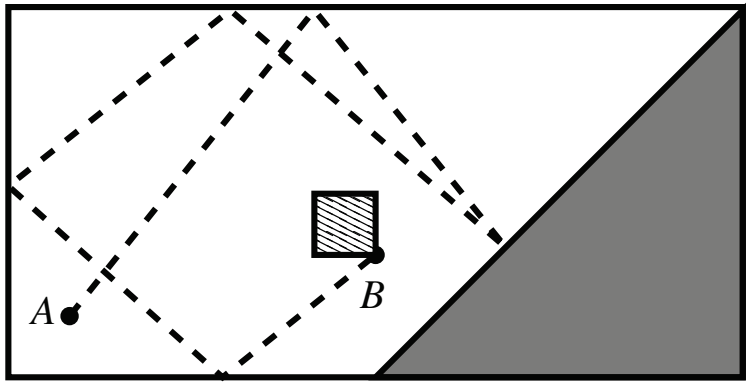


Lösung I



Die Abbildung zeigt uns sechs geeignet gespiegelte und verschobene Kopien der Backstube, die der Reihe nach jeweils eines der sechs Wegstücke enthalten müssen. Wir führen ein Koordinatensystem ein, das der linken unteren Ecke die Koordinaten $(0, 0)$ zuweist und das 1 Meter als Einheitslänge hat. Die rechte obere Ecke liegt dann in $(48, 60)$, der Punkt A in der ersten Kopie liegt in $(2, 2)$, und der Punkt B in der sechsten Kopie liegt in $(40, 48)$. Die Länge des kürzesten Weges ist daher $\sqrt{38^2 + 46^2} = \sqrt{3560} \approx 59,66$.

Lösung II



Zufall

Zufällige Ereignisse geschehen objektiv ohne (erkennbare) Ursache.

Z.B tritt dies auf bei Radioaktivität, d.h. der Zeitpunkt des Zerfalls des nächsten radioaktiven Atoms aus einer Stoffmenge ist nicht vorhersagbar.

Der Begriff des Zufalls (inklusive seiner Existenz) wird heftig in verschiedenen Gebieten wie Philosophie, Physik, Psychologie, Soziologie, etc. diskutiert (z.T. sehr kontrovers, auch aufgrund verschiedener Weltanschauungen).

Nachfolgend gehen wir von der Existenz zufälliger Ereignisse (Basismodell Münzwurf) aus und interpretieren Zufall in erster Linie als Hilfsmittel für diverse Informatikanwendungen.

Rolle des Zufalls in frühen Kulturen

Zufall spielt schon früh in der kulturellen Entwicklung der Menschheit eine Rolle:

- Zufällige Wahl des Jagdgebiets ($\rightsquigarrow$ Vermeidung von Gewohnheitsmustern bei der Beute).
- Typisch war in der Geschichte die enge Verbindung von Zufall mit Schicksal.
- Glückspiele waren schon bei den alten Ägyptern, Chinesen und Indern populär.
- Münzwürfe schon vor über 3000 Jahren in China studiert.
- Erst im 17. Jahrhundert erfolgte eine stärkere „Mathematisierung“ der Betrachtungen.
- Leibniz setzte Zufall in Bezug zu Komplexität.

Zitate zum Zufall

„Das, wobei unsere Berechnungen versagen, nennen wir Zufall.“
Albert Einstein

„Gott würfelt nicht.“
Albert Einstein

„Die zwei größten Tyrannen der Erde: der Zufall und die Zeit.“
Johann Gottfried Herder

„Der Zufall ist der einzig legitime Herrscher des Universums.“
Napoleon

„Das Glück gehört denen, die sich selbst genügen. Denn alle äußeren Quellen des Glückes und Genusses sind ihrer Natur nach höchst unsicher, misslich, vergänglich und dem Zufall unterworfen.“
Arthur Schopenhauer

⇝**Fazit:** Viel Skepsis dem Zufall gegenüber, aber für die Informatik ist der Zufall in erster Linie eine wichtige zu nutzende *Ressource*...

Zufall in der (bzw. für die) Informatik

In dieser Vorlesung schon gesehen:

- **Simulated Annealing:** Mit einer gewissen Wahrscheinlichkeit eine schlechtere Lösung akzeptieren.
  ⇝ Vermeidet das Hängenbleiben in lokalen Optima.
- **Genetische Algorithmen:** Zufällige Mutationen existierender Lösungen.

Lastverteilung (Load balancing): Zufällige Verteilung von Berechnungen oder Anfragen auf mehrere parallel arbeitende Systeme. Bsp.: Zufällige Zuweisung von HTTP-Requests an einen von mehreren zur Verfügung stehenden Webservern.

Wichtige Gebiete, wo die Ressource Zufall genutzt wird:

- Effiziente (und einfache) Algorithmen
- Simulationsmethoden wie Monte Carlo
- Heuristiken
- Protokollentwurf
  - Internet
  - Kommunikationsnetze
  - Kryptologie

Was ist Zufall?

Zufallsexperimente: Ein (gedanklicher oder tatsächlicher) Versuch, der unter gleichen Bedingungen beliebig oft wiederholbar ist, dessen Ausgang sich aber nicht exakt vorhersagen lässt, d.h. bei Wiederholung des Versuchs jedesmal ein anderes Ergebnis möglich.
Beispiele: Münzwurf, Anzahl der eine Kreuzung zwischen 12 und 13 Uhr befahrenden Fahrzeuge, Roulette, etc.

Kolmogorov-Komplexität: Eine Zeichenkette ist genau dann zufällig, wenn es kein Computerprogramm gibt, das diese Zeichenkette erzeugt und **kürzer** als sie ist.

Pseudozufallszahlengeneratoren

Ein Pseudozufallszahlengenerator ist ein Verfahren, das eine Folge von (vermeintlich) zufälligen bzw. zufällig erscheinenden Zahlen erzeugt. Man beginnt in der Regel mit einem vorgegebenen Startwert.

Beispiele:

- Linearer Kongruenzgenerator (z.B. in Java):
  Startwert $\mathbf{X_0}$
  $\mathbf{X_n} = (a\mathbf{X_{n-1}} + b) \bmod m$
- Multiply With Carry (MWC):
  r Startwerte $\mathbf{X_0}, \ldots \mathbf{X_{r-1}}$, initialer Übertrag $\mathbf{C_{r-1}}$
  $\mathbf{X_n} = (a\mathbf{X_{n-r}} + \mathbf{C_{n-1}}) \bmod m$, $\mathbf{C_n} = \lfloor \frac{a\mathbf{X_{n-r}} + \mathbf{C_{n-1}}}{m} \rfloor$, $n \geq r$
- Weitere Beispiele wie „Mersenne Twister“ und „Linear additive feedback RNG“

Unfaire Münzen

Basismodell des Zufalls: „Fairer Münzwurf“, d. h. Münzwurf liefert mit W'keit $\frac{1}{2}$ Kopf und mit W'keit $\frac{1}{2}$ Zahl.

Hat man Zugriff auf eine faire Münze, so kann man damit andere Wahrscheinlichkeitsverteilungen erzeugen...
Aber was machen falls Münze unfair, d. h. $Pr[\text{Kopf}] \neq Pr[\text{Zahl}]$?

Man kann faire Münzen simulieren $\rightsquigarrow$ **Satz von von Neumann**

John von Neumann, 1903–1957

- Mathematiker.
- Bedeutende Beiträge auf den Gebieten der Logik, Funktionalanalysis, Quantenmechanik, Numerik und Spieltheorie.
- von Neumannsches Rechnerarchitekturprinzip noch heute von Bedeutung $\rightsquigarrow$ gehört zu den Vätern der Informatik.

Satz von von Neumann

Theorem

Eine faire Münze kann durch eine deterministische Turingmaschine (det. Algorithmus) mit Zugriff auf eine Folge „unfairer Münzwürfe“ ($Pr[\text{Kopf}] = \rho \neq \frac{1}{2}$) in erwarteter Laufzeit $O(\frac{1}{\rho(1-\rho)})$ simuliert werden.

Beweis (Idee): Mache jeweils zwei Münzwürfe (mit Münze 1 und 2), bis zwei unterschiedliche Werte (also Kopf und Zahl) auftauchen. Falls Münze 1 Kopf zeigt so gebe Kopf aus und andernfalls Zahl.

Obiger Algorithmus simuliert fairen Münzwurf:
W'keit für Münze 1 = Kopf & Münze 2= Zahl ist $\rho \cdot (1-\rho)$.
Analog W'keit für Münze 1=Zahl & Münze 2=Kopf ist $(1-\rho) \cdot \rho$, also beide gleich.

Beachte das man ρ für obigen Algorithmus noch nicht mal kennen muss....!

Zufall in Internetprotokollen

Protokolle bilden das Rückgrat des Internets zur Versendung und dem Empfang von Datenpaketen.

Modellierung wichtiger Eigenschaften als Zufallsvariablen:

- Paketverlustraten,
- Paketlatenz,
- Transferraten.

Zufallsbasierte Entscheidungen in Algorithmen:

- Aufwachzeiten in drahtlosen Sensornetzwerken.
- Auswahl von Paketen zum „Wegwerfen“ bei Pufferüberlauf in Routern.

Zufall in verteilten Systemen: Speisende Philosophen

Auflösung von Verklemmungen:

Szenario: n Philosophen im Kreis mit je einer Gabel dazwischen. Um zu essen braucht ein Philosoph beide Gabeln links und rechts von ihm.

Problem: Finde Protokoll welches

- **deadlock-frei** ist: falls jemand hungrig ist, dann isst auch irgendwann jemand.
- **lockout-frei** ist: ein Hungriger bekommt irgendwann zu essen.

Speisende Philosophen: Deterministische Protokolle

Deterministische Protokolle benötigen gemeinsamen Speicher oder Kommunikation mittels Nachrichtenübermittlung.

Beispiel: Nummeriere Philosophen durch und lasse die geraden bzw. ungeraden abwechselnd essen.

Lehmann/Rabin 1981: Kein deterministisches Protokoll kann voll verteilt und symmetrisch sein.

- **Voll verteilt**: es gibt keinen zentralen Speicher oder zentralen Prozess auf den alle zugreifen können.
- **Symmetrisch**: alle Prozesse führen das selbe Programm aus, Prozessoren kennen nicht ihre Identität (z. B. Nummer)

Speisende Philosophen: Randomisiertes Protokoll

Voll verteilt, symmetrisch und deadlockfrei (noch nicht lockout-frei):

Protokoll für jeden Philosophen:
Wiederhole für immer:

1. Denke bis hungrig...
2. Entscheide per Münzwurf, ob zuerst linke oder rechte Gabel genommen.
3. Warte, bis gewählte Gabel frei ist und nimm sie dann.
4. Falls andere Gabel nicht frei, so lege Gabel zurück und gehe zu (2). Andernfalls nimm auch die andere Gabel.
5. Kritischer Bereich: Iss!
6. Lege beide Gabeln wieder hin.

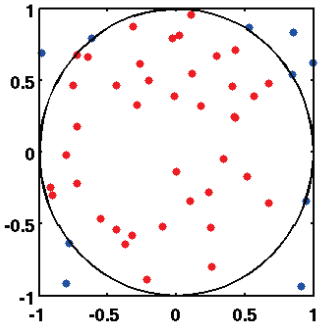
Nicht lockout-frei: Gefräßiger Philosoph kann Nachbarn immer die Gabel wegnehmen...

Monte Carlo-Simulation

Wikipedia: Die Monte Carlo-Simulation ist ein Verfahren aus der Stochastik, bei dem sehr häufig durchgeführte Zufallsexperimente die Basis darstellen.

Z. B. Monte Carlo-Verfahren zur Bestimmung der Kreiszahl π:
Wähle **zufällig** Punkte aus dem Intervall $[-1,1] \times [-1,1]$.

$$Pr[\text{Im Kreis}] = \frac{\text{Kreisfläche}}{\text{Quadratfläche}} = \frac{r^2 \cdot \pi}{(2 \cdot r)^2} = \frac{\pi}{4} = \frac{\text{Treffer in Kreisfläche}}{\text{Anzahl der Punkte}}$$



$\rightsquigarrow$ Gesetz der großen Zahlen:

$$\pi \approx 4 \cdot \frac{\text{Treffer in Kreisfläche}}{\text{Anzahl der Punkte}}$$

Zufall für Fingerabdrücke: Matrixmuliplikation I

Verifikation einer Matrixmultiplikation
Eingabe: $n \times n$ Matrizen A, B, C.
Frage: $A \cdot B = C$?

Algorithmus von Freivalds (1977):

1 Wähle zufällige gleichverteilte Zahl $r \in \{0,1\}^n$
2 $y := A \cdot (B \cdot r)$
3 $z := C \cdot r$
4 **if** $y = z$ **then output** „Ja“ **else output** „Nein“

Beobachtung: Falls $A \cdot B = C$ dann antwortet obiger Algorithmus “Ja”.
Was ist falls $A \cdot B \neq C$?

Zufall für Fingerabdrücke: Matrixmuliplikation II

Mitteilung: Seien A, B, C $n \times n$ Matrizen mit $A \cdot B \neq C$. Dann gilt für eine zufällig per Gleichverteilung gewählte Zahl $r \in \{0,1\}^n$, dass $\Pr[A \cdot B \cdot r = C \cdot r] \leq 1/2$.

Bemerkungen:

- $O(n^2)$ Laufzeit statt $O(n^3)$ vermöge Ausmultiplizieren gemäß Schulmethode.
- Wiederholt man den Algorithmus k-mal, dann sinkt die Fehlerwahrscheinlichkeit auf $\leq 2^{-k}$.
- Sogenannter **Monte Carlo-Algorithmus** mit „einseitigem Fehler“.

Zufall zum Überprüfen

Eingabe: Zwei Polynome $p_1(x_1, x_2, \ldots, x_n)$ und $p_2(x_1, x_2, \ldots, x_n)$.
Frage: Sind die Polynome gleich, d. h. $p_1(x) = p_2(x)$ für alle $x \in \mathbb{R}^n$?
Äquivalente Frage: $p_1(x) - p_2(x) = p(x) = 0$ für alle $x \in \mathbb{R}^n$.

Algorithmus (analog zu Matrixmultiplikation):

1. Für eine endliche Teilmenge $S \subseteq \mathbb{R}$, wähle zufällig & gleichverteilt $x \in S^n$.
2. Falls $p(x) = 0$, so gib „Ja“ aus und andernfalls „Nein“.

Klar: Falls p das Nullpolynom ist, dann gibt obiger Algorithmus immer Ja aus.
Was ist falls p nicht das Nullpolynom ist? Dazu:

Lemma von Schwartz-Zippel: Sei p ein Polynom vom Grad $d \geq 0$ und sei $S \subseteq \mathbb{R}$ eine endliche Menge. Dann gilt für eine zufällige gleichverteilte Zahl $x \in S^n$:

$$Pr[p(x) = 0] \leq \frac{d}{|S|}.$$

Anwendung: Existenzbestimmung von *perfekten Matchings* in allgemeinen Graphen:
Eingabe: Ein Graph $G = (V, E)$.
Frage: Gibt es eine Menge von $\frac{|V|}{2}$ Kanten, sodass jeder Knoten Endpunkt einer dieser Kanten ist?

Schritt 1: Berechnung der *Tutte Matrix* A:

$$A_{ij} = \begin{cases} x_{ij} & \text{wenn } (i,j) \in E \text{ und } i < j \\ -x_{ji} & \text{wenn } (i,j) \in E \text{ und } i > j \\ 0 & \text{sonst} \end{cases}$$

Schritt 2: Berechnung der Determinante von $A \rightsquigarrow$ Polynom p vom Grad n mit Unbekannten $x_{ij}, i < j$.

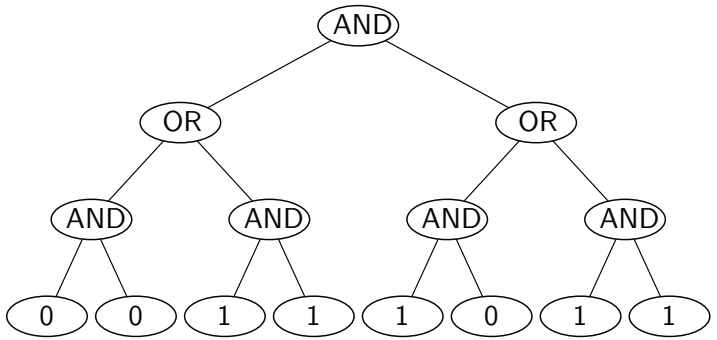
Satz (Tutte 1947) G hat hat ein perfektes Matching genau dann, wenn p das Nullpolynom ist.

Zufall gegen böse Gegenspieler: Spielbaumauswertung

Eingabe: Ein Binärbaum (Spezialfall!) T_k der Höhe $2k$ mit

- Wurzel und interne Knoten auf „geradzahliger Schicht“ sind mit dem logischen AND bezeichnet.
- Alle anderen internen Knoten sind mit dem logischen OR bezeichnet.
- Blätter haben Wert 1 oder 0

Aufgabe: Berechne den Wert der Wurzel.



Spielbaumauswertung – Randomisierter Algorithmus

Frage: Wieviele Blätter anschauen um Wert der Wurzel zu berechnen?
Klar: Deterministischer Alg. muss im schlimmsten Fall alle $n = 2^{2k}$ Blätter anschauen.

Randomisierter Algorithmus:

```
1  if v ist AND then
2      Wähle zufällig Kind u von v & evaluiere rekursiv
3      if u hat Wert 1 then evaluiere anderes Kind von v
4      if u hat Wert 0 then v hat Wert 0
5  if  v ist OR then ...  ▷ analog zu AND, mit 0 und 1 ausgetauscht
```

Mitteilung: Die erwartete Anzahl von zu evaluierenden Blättern ist 3^k.
Zentrale Idee: Falls Spielbaum Wert 0 hat, dann ist mindestens eines der beiden Kinder der Wurzel 0. Mit W'keit $\frac{1}{2}$ wird dieses Kind in Schritt 2) ausgewählt und deshalb das zweite Kind garnicht mehr betrachtet.
Beachte: Zufall hilft gegeben einen „bösen Gegenspieler" welcher erzwingt alle Blätter anzuschauen.
Mitteilung: 3^k kann nicht verbessert werden.

Zufall für die Wettervorhersage

Mittels **Markov-Modellen** können nicht bzw. schwer vorhersagbare Prozesse (wie z.B. der Wetterverlauf) simuliert / analysiert werden.

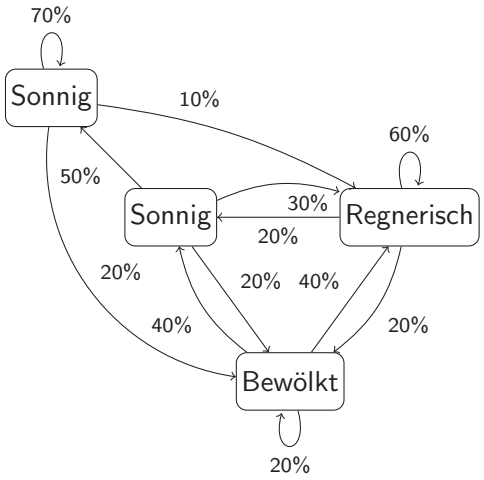
Andrei Andrejewitsch Markov, 1856-1922, Mathematiker

Zufall für die Wettervorhersage – Markov-Modelle

Zustandsgetrieben: Ein Zustand des Modells bildet Zustand in der Welt ab.

Übergangswahrscheinlichkeiten (Erfahrungswerte): Wie wahrscheinlich ist ein Übergang von einem Zustand in einen anderen?

Bsp.: Wahrscheinlichkeit p dass auf einen bewölkten Tag zwei sonnige Tage folgen: $p = 0,4 \cdot 0,5 = 0,2 = 20\%$.



Interaktion und Zufall: Zero Knowledge-Beweise I

Problem: Alice möchte Bob überzeugen ein bestimmtes Wissen oder „Geheimnis" zu kennen ohne Bob das Geheimnis preiszugeben.

Beispiel: Alice kennt eine Lösungsformel für Polynome 3. Grades

1. Alice bittet Bob ihm ein Polynom 3. Grades zu nennen.
2. Alice löst das Polynom und teilt Bob die Nullstellen mit.
3. Bob überprüft die Nullstellen.

Nennt Alice hinreichend oft die korrekten Nullstellen für verschiedene Polynome, so ist Bob wohl überzeugt dass Alice die Lösungsformel bzw. das „Geheimnis" kennt...

Bob hat aber während des gesamten Protokolls nichts gelernt
$\rightsquigarrow$ Zero Knowledge!

Hinweis: Beachte Parallele zu NP-vollständigen Problemen: Schritt 2) Beweise finden und Schritt 3) Beweise verifizieren. Schritt 3) ist effizient durchführbar.

Zero Knowledge-Beweise II

Alice möchte Bob überzeugen dass zwei ähnlich aussehende Bilder nicht identisch sind. Wie kann Alice Bob überzeugen, ohne die Unterschiede zu verraten?

1. Alice wendet sich ab.
2. Bob vertauscht mit Wahrscheinlichkeit $1/2$ die Bilder.
3. Alice schaut sich die Bilder an und sagt Bob ob sie vertauscht wurden.

Die Wahrscheinlichkeit das Alice im Schritt 3) zufällig die richtige Lösung rät ist $\frac{1}{2}$. Wiederholt man das Protokoll k-mal und Alice antwortet immer richtig, dann ist die W'keit dafür 2^{-k}.

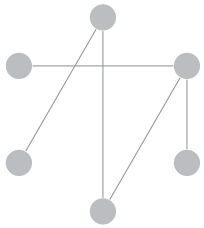
Zufallsgraphen

Zufallsgraphen spielen eine wichtige Rolle in einer Vielzahl von Gebieten.
Frage: Wie erzeugt man zufällige Graphen?

Einfacher Ansatz zum Erzeugen eines Graphen mit n Knoten: Jede (der $\binom{n}{2}$ möglichen) Kanten ist mit einer Wahrscheinlichkeit p vorhanden.
$\rightsquigarrow$ zwei Parameter: n und p.

Beispiel: $n = 6, p = \frac{1}{3}$

Erwartete Anzahl Kanten: $\binom{6}{2} \cdot \frac{1}{3} = 5$

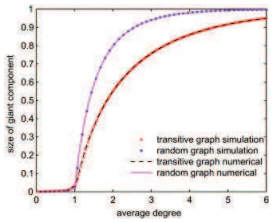


$\rightsquigarrow$ Haben solche zufällig erstellten Graphen bestimmte Eigenschaften?

Zufallsgraphen - Eigenschaften

Haben solche zufällig erstellten Graphen bestimmte Eigenschaften? Ja!

- Knotengradverteilung entspricht einer **Binomialverteilung**, keine Gleichverteilung! $\rightsquigarrow$ Wenige Knoten mit hohem bzw. niedrigem Knotengrad.
- Größe der größten Zusammenhangskomponente:



  $\rightsquigarrow$ Die Größe der größten Zusammenhangskomponente steigt sprunghaft!
- Viele kleine und eine große Zusammenhangskomponente.
- Der erwartete Durchmesser ist $\ln n$ $\rightsquigarrow$ „Small World“

Information

Wikipedia: Information (lateinisch informare „bilden“, „eine Form, Gestalt, Auskunft geben“) ist der strukturelle Aspekt physikalischer Wechselwirkungen, der aktionsprägend auf komplexe Systeme wie Menschen oder auch Maschinen wirkt. Sie vermittelt einen Unterschied.

Information ist Gegenstand verschiedener Struktur- und Geisteswissenschaften, kann mathematischer, philosophischer oder empirischer (etwa soziologischer) Begriff sein.

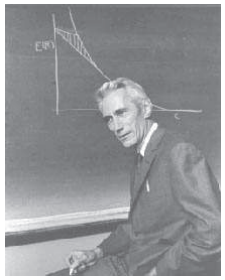
Beliebte Definition:
Informatik = Information + Automatik.
Bisher haben wir uns vorwiegend mit „Automatik“, sprich Algorithmen beschäftigt. Nun zum Begriff der Information.

Wo **Daten** eher als das „Rohmaterial“ der Informatik / Computer gesehen werden, da ist der Begriff der **Information** eher an den Anwendungskontext gebunden, sprich Information wird als „Rohmaterial“ der jeweiligen Anwendung gesehen.

Information aus Informatiksicht

Was ist Information und wie kann man Informationsgehalt messen?

Shannon verknüpfte als erster den Informationsbegriff mit dem Begriff des *Zufalls*: Eine Binärzeichenkette aus lauter Nullen hat sehr geringen Informationsgehalt, eine völlig zufällige Binärzeichenkette aber sehr hohen Informationsgehalt (vergleiche Kolmogorov-Komplexität im vorigen Kapitel).
⇝ Begriff der **Entropie**, den Shannon 1948 einführte und damit zum Vater der Informationstheorie wurde.



Claude E. Shannon, 1916–2001, Mathematiker und Elektrotechniker

Nachfolgend gehen wir kurz ein auf

- Codierungstheorie;
- Informationstheorie;
- Datenkomprimierung.

Codierungstheorie

Wikipedia: Ein Code f über den Alphabeten A und B ist eine **injektive Abbildung** (= Codierung) der Form $f: A \to B$. Er ordnet Wörtern aus Symbolen des Alphabets A Wörter aus dem Alphabet B zu. Als Code wird auch die Bildmenge von f, bezeichnet. Der Code heißt **entzifferbar**, wenn es eine eindeutige Umkehrabbildung f^{-1} gibt, die jedem Nachrichtenwort aus B wieder das ursprüngliche Wort aus A zuordnet.

Beispiele:

- Postleitzahlen.
- Binärcodierung. (Rückblick: Selbst Turing-Maschinen wurden binär codiert, Gödelisierung).
- ASCII-Code.
- Morsecode.
- Strichcodes (QR-Codes) auf Produkten im Supermarkt.
- ...

Binärcodierung

Binärzahl mit k Stellen: $Z = z_k z_{k-1} \dots z_1$, $z_i \in \{0,1\}$, z. B. 1011
Der Wert W von Z bzw. die Darstellung von Z im Dezimalsystem ist:

$$W = \sum_{i=1}^{k} z_i 2^{i-1}$$

Umrechnung von Binärzahlen in Dezimalzahlen durch wiederholtes Rechnen modulo 2:

$$\begin{array}{rcrcl} 41 & :2 & = & 20 & \text{Rest } \mathbf{1} \\ 20 & :2 & = & 10 & \text{Rest } \mathbf{0} \\ 10 & :2 & = & 5 & \text{Rest } \mathbf{0} \\ 5 & :2 & = & 2 & \text{Rest } \mathbf{1} \\ 2 & :2 & = & 1 & \text{Rest } \mathbf{0} \\ 1 & :2 & = & 0 & \text{Rest } \mathbf{1} \end{array} \uparrow$$

$\rightsquigarrow [41]_{10} = [101001]_2$

Fehlerentdeckende Codes I

Problem: Bei der Übertragung von Informationen können Fehler auftreten (z.B. Bits werden „geflippt“). Wie erkennt man ob die Übertragung korrekt war?
Lösung: Fehlerentdeckende Codes durch Ausnutzung von *Redundanz*.

Bei ISBN (International Standard Book Number) ist die letzte Ziffer der Zahl eine *Prüfziffer*.

Berechnung der Prüfziffer bei ISBN-10 (10 Stellen $b_1 b_2 \dots b_{10}$):

$$b_{10} = \left(\sum_{i=1}^{9} i \cdot b_i \right) \bmod 11$$

(Anstelle von $b_{10} = 10$ wird ggf. $b_{10} = X$ geschrieben.)
Beispiel: Die ISBN 3-411-05232-5 ist gültig:

$$(3 + 8 + 3 + 4 + 0 + 30 + 14 + 24 + 18) \bmod 11 = 104 \bmod 11 = 5$$

⇝ Wird eine ungültige ISBN empfangen, so kann dies **erkannt** werden! Aber der Fehler kann nur durch wiederholtes Übertragen behoben werden.

Fehlerentdeckende Codes II

Auch bei Kreditkartennummern gibt es eine Gültigkeitsprüfung:
Prüfungsvorschrift:

1. Ersetze Ziffern an ungerader Position durch folgende Ersetzung:
   $0 \mapsto 0$, $1 \mapsto 2$, $2 \mapsto 4$, $3 \mapsto 6$, $4 \mapsto 8$,
   $5 \mapsto 1$, $6 \mapsto 3$, $7 \mapsto 5$, $8 \mapsto 7$, $9 \mapsto 9$
2. Falls die Quersumme der neuen Zahl durch 10 teilbar ist, *könnte* die Nummer zulässig sein, ansonsten ist die Nummer ungültig.

Beispiel:
4544 7000 1234 6789
$\overset{(1)}{\rightsquigarrow}$ 8584 5000 2264 3779
$\rightsquigarrow$ Quersumme ist 70, also durch 10 teilbar
$\rightsquigarrow$ Nummer könnte zulässig sein.

Fehlerkorrigierende Codes I

Bisher: Fehler in der Übertragung können festgestellt werden, aber nicht ohne wiederholte Übertragung behoben werden.
Jetzt: Fehlerkorrigierende Codes.

Grundlegende Idee: Den „Abstand“ der Codewörter untereinander erhöhen!

Beispiel: Codierung zweier Wörter: Ja $\mapsto$ 111, Nein $\mapsto$ 000.
$\rightsquigarrow$ zwei der drei Stellen der Codewörter sind redundant.

Die Übertragung liefert das Codewort „101“.
$\rightsquigarrow$ Wahrscheinlich wurde das Wort „Ja“ übertragen.

Fehlerkorrigierende Codes II

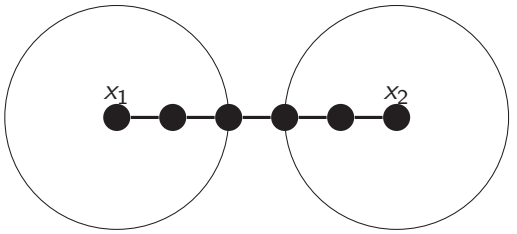
Die Anzahl der Bits, in denen sich zwei gleich lange 0-1-Wörter x und y unterscheiden, nennt man die **Hamming-Distanz**, in Zeichen $d_H(x, y)$.

Die **Hamming-Distanz eines Codes** (also einer Menge von Codewörter) ist die kleinste vorkommende Hamming-Distanz $d_H(x, y)$ zwischen je zwei verschiedenen Codewörtern x und y.

Beispiele:
$d_H(111, 000) = 3$; $d_H(11000110, 01101100) = 4$
Die Hamming-Distanz der Binärcodierung von Dezimalzahlen ist eins.

Fehlerkorrigierende Codes III

Mitteilung: Bei einem Code mit Hamming-Distanz d kann man bis zu $\lfloor \frac{d-1}{2} \rfloor$ Bitfehler korrigieren.



x_1, x_2: gültige Codewörter
$\rightsquigarrow$ alle möglichen Wörter mit Hamming-Abstand $\lfloor \frac{d-1}{2} \rfloor$ zu x_1 können x_1 zugeordnet werden.
$\rightsquigarrow$ Anzahl gültiger Codewörter reduziert

Mitteilung: Für die Anzahl a der Codewörter in einem k Fehler korrigierenden Code der Länge n muss gelten:

$$a \leq \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{k}}$$

Informationstheorie

Ziel: Quantifizierung von Information.
Claude Shannon: Informationsbegriff wird Zufallsereignis zugeordnet

Münzwurf wird die Maßzahl **1 bit Informationsgehalt** zugeordnet
Werfen von zwei Münzen hat entsprechend 2 bit Informationsgehalt

Sinnvoll? Interpretation: Informationsgehalt eines Zufallsexperiments entspricht der durchschnittlichen Anzahl der Ja/Nein-Fragen, welche notwendig sind um den Experimentausgang zu bestimmen.

Beispiel: Jemand wählt zufällig eine Zahl zwischen 1 und einer Million.
Wieviele Fragen muss man stellen um Zahl sicher zu bestimmen?
Beste Strategie ist binäre Suche $\rightarrow \log_2(1.000.000) \approx 20$ Fragen

Entropie I

Extreme: Zufallsexperiment mit einem Ereignis der W'keit 1 und einem zweiten Ereignis mit W'keit 0. Informationsgehalt? Keiner!
Nun Informationsgehalt von Zufallsereignissen mit beliebiger Verteilung:

Definition: Eine Zufallsereignis mit Ereignissen $a_1, \ldots, a_k$ und entsprechenden W'keiten $p_1, \ldots, p_k$ (wobei $\sum_{i=1}^{k} p_i = 1$) hat den folgenden Informationsgehalt bzw. **Entropie**:

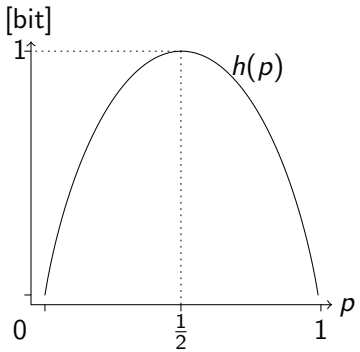
$$H(p_1, \ldots, p_k) = -\sum_{i=1}^{k} p_i \log_2(p_i) \text{ [bit]}$$

Für $p_i = 0$ setzen wir $p_i \log_2(p_i) = 0$.

Beispiel: Zufallsexperiment „Schwangerschaft":
$Pr[\text{Junge}] = 0,48$, $Pr[\text{Mädchen}] = 0,49$ und $Pr[\text{Zwillinge}] = 0,03$
Dann gilt $H(0,48; 0,49; 0,03) \approx 1,164$ bit.

Entropie II

Allgemeine Entropieverteilung für Zufallsexperiment mit zwei Ausgängen, d.h. $h(p) := H(p, 1-p)$.



Interpretation: Entropie als gewichteter Mittelwert.

- Informationsgehalt des Ereignisses a_i ist $-\log_2(p_i)$, d.h. je geringer W'keit für a_i desto größer ist Informationsgewinn bei Eintreten des Ereignisses.
- Informationsgehalt geht mit Gewicht p_i ein, entsprechend $\sum_{i=1}^{k} p_i = 1$.

Datenkomprimierung

Datenkomprimierung ist quasi Gegenspieler zur Fehlerentdeckung und -korrektur. Während man dort die Redundanz erhöht, will man sie bei der Komprimierung verringern.

Ausgangspunkt: Dateien

- enthalten Redundanzen,
- verbrauchen relativ viel Speicherplatz und
- dauern lange zu übertragen.

$\rightsquigarrow$ Datenkomprimierung kann **Speicherplatz** und **Übertragungszeit** sparen.

Beispiel: Ein unkomprimiertes FullHD-Video in 1080p50-Qualität benötigt ≈ 2.08 GBit/s. Zwei Stunden Film benötigen dann ≈ 15000 GB oder ca. 300 BlueRay-Disks.

Anwendungen von Datenkomprimierung

Generische Dateikomprimierung

- Dateien: gzip, bzip2.
- Archivierungsprogramm: pkzip, rar.
- Dateisystem: NTFS (New Technology File System).

Multimedia

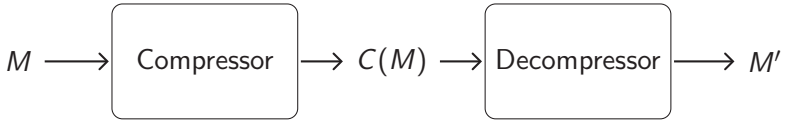
- Bilder: GIF (Graphic Interchange Format), JPEG (Joint Photographic Experts Group)
- Audio: MP3 (MPEG-1 oder MPEG-2 Audio Layer III)
- Video: MPEG (Moving Picture Experts Group), DivX

Kommunikation

- ITU-T T4 Group 3 Fax
- V.42bis modem

Grundbegriffe der Komprimierung

M: Eingangsdaten
C(M): Komprimierte Daten
M′: Rekonstruierte Daten



Wünschenswert: $|C(M)| < |M|$
$\rightsquigarrow$ **Kompressionsrate** $= |C(M)|/|M|$

Die Komprimierung ist

- **verlustfrei** (engl. „lossless“), falls $M' = M$;
- **verlustbehaftet** (engl. „lossy“), sonst.

Verlustfreie Kompression

Typische Anwendungen:
Texte, Sourcecode, ausführbare Dateien.

Bekannte verlustfreie Kompressionsprogramme:
gzip, bzip2, pkzip, rar.

Wichtige Kompressionsverfahren:

- Lauflängencodierung;
- Statistisch: Huffmankodierung und arithmetische Kodierung;
- Wörterbuchbasiert: Lempel-Ziv und ihre Varianten

Typische Kompressionsraten: ca. 50%, für Text bis zu 25%.

Lauflängencodierung („Run-Length Encoding“, RLE)

Idee: Ausnutzung von Wiederholungen in den Eingabedaten.

Beispiel „Bitmaps“: Eingabe besteht nur aus ‘0’ und ‘1’ $\rightsquigarrow$ Abwechselnd die Anzahlen der aufeinanderfolgenden ‘0’ oder ‘1’ speichern!
Diese bezeichnen wir als Lauflänge.

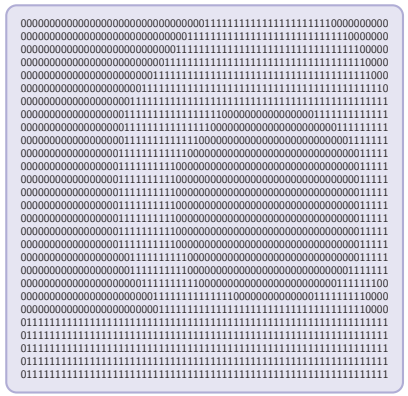
Frage: Wie kodiert man die Lauflänge?
Möglichkeiten:

- Kodierung mit fester Bitzahl
- Kodierung mittels variabler Bitzahl $\rightsquigarrow$ häufiger auftretende Lauflängen werden mit weniger Bits kodiert

Lauflängencodierung Beispiel

M: Gescanntes ‘b’, s/w, gedreht

Unkomprimiert: 1 Bit/Pixel



28 Zeilen

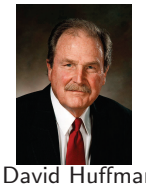
64 Spalten

RLE komprimiertes C(M)

```
32 22 10
29 28 7
27 32 5
25 35 4
23 38 3
21 42 1
19 45
18 17 16 13
18 15 22 9
18 13 26 7
17 12 30 5
17 10 32 5
17 10 32 5
17 10 32 5
17 10 32 5
17 10 32 5
17 10 32 5
17 10 32 5
19 10 30 5
19 10 28 7
21 10 24 7 2
23 14 14 9 4
24 36 4
1 63
1 63
1 63
1 63
1 63
```

Speicherverbrauch: $28 \cdot 64 + 6 = 1798$ Bits $95 \cdot 6 + 6 = 576$ Bits

Huffman-Codierung



David Huffman
1925—1999

Grundidee

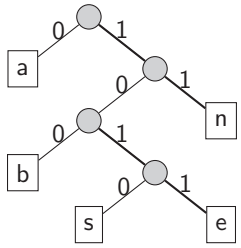
- Kodiere jedes Zeichen x mit einem Binärcode $C(x)$.
- Zeichen kommen im Text unterschiedlich häufig vor.
  $\rightsquigarrow$ Häufigere Zeichen werden mit weniger Bits kodiert.

Huffman-Baum

- Kodierung wird durch einen Binärbaum beschrieben.
- Zeichen entsprechen Blättern im Baum.
- Pfad von Wurzel zum Blatt definiert Code (links = ‘0’, rechts = ‘1’)

Beispiel: Kodiere das Wort ‘bananenananas’.

Zeichen	a	b	e	n	s	Länge
Naiv	000	001	010	011	100	39 Bits
Huffman	0	100	1011	11	1010	26 Bits



Frage: Wie erstellt man einen Huffman-Baum?

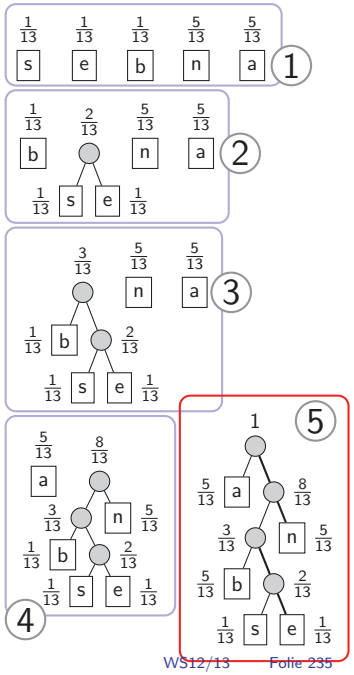
Erstellung des Huffman-Baums

Vorgehensweise: Bottom up

1: Erzeuge für jedes Zeichen x einen Teilbaum mit Knoten x und Gewicht $p_x :=$ relative Häufigkeit von x in M
2: **while** Es gibt mehr als einen Teilbaum **do**
3: Suche zwei Teilbäume mit kleinsten Gewichten p_1 und p_2
4: Vereinige die beiden Teilbäume zu einem mit Gewicht $p_1 + p_2$
5: **end while**

Mitteilung: Huffman-Codierung ist **optimal**, d.h für eine gegebene Eingabe M mit relativen Häufigkeiten p_x gibt es keine Kodierung, die jedem Zeichen eine ganze Anzahl von Bits zuweist und M mit weniger Bits kodiert als die Huffman-Kodierung.

Bsp: 'bananenananas'



Lempel-Ziv

Grundidee

- Zeichenketten sind meist mehrmals in Eingabe vorhanden.
- Kodiere eine Zeichenkette durch ihren Index in einem Wörterbuch.
- Das Wörterbuch wird an die Eingabe angepasst.

Fragen Was steht im Wörterbuch? Was tut man, wenn eine Zeichenkette nicht im Wörterbuch steht?

Eine Lsg. LZW (Lempel-Ziv-Welch von 1984) verwendet Wörterbuch W mit maximal 2^{12} Einträgen, um eine Eingabe M zu komprimieren.

1: W enthält zunächst alle 256 möglichen 1-Byte-Zeichenketten.
2: **while** M ist nicht leer **do**
3: Finde die längste Zeichenkette s in W, die mit dem Anfang von M übereinstimmt.
4: Gib den Index von s in W aus und entferne s vom Anfang von M.
5: Füge s gefolgt vom nächsten Zeichen in M zu W hinzu.
6: **end while**

Beispiel LZW

Eingabe:
'aabaaabaaabaaaab'; Länge in 8-Bit-ASCII: 128 Bit

Ablauf der Komprimierung
Initial.: $W[0] = \backslash 0, \ldots, W[97] = \text{a}, W[98] = \text{b}, \ldots, W[255] = \ddot{\text{y}}$

Längste Zeichenkette s in W	Ausgabe	Eintrag in W
a	97	$W[256] :=$ a**a**
a	97	$W[257] :=$ a**b**
b	98	$W[258] :=$ b**a**
aa	256	$W[259] :=$ aa**a**
ab	257	$W[260] :=$ ab**a**
aaa	259	$W[261] :=$ aaa**b**
ba	258	$W[262] :=$ ba**a**
aaab	261	

Ausgabelänge: Insgesamt $8 \cdot 12 = 96$ Bit $\rightsquigarrow$ Kompressionsrate $= 0.75$

Das Counting-Argument

Frage: Gibt es ein Verfahren, um ALLE Dateien verlustfrei zu komprimieren?

Antwort: Nein! There is no free lunch! Aber warum?

Argumentation: Angenommen, man könnte alle Dateien der Länge n Bits um mindestens 1 Bit komprimieren. Es gibt 2^n Dateien der Länge n, und insgesamt $2^1 + 2^2 + \ldots + 2^{n-1} = 2^n - 2$ Dateien mit weniger Bits. Dann müssen mindestens zwei Eingabedateien auf die gleiche komprimierte Datei kodiert werden $\rightsquigarrow$ keine verlustfreie Dekompression möglich.

Gegenstück in der Physik: Perpetuum Mobile

Verlustbehaftete Kompression

Grundidee
Tausche eine deutlich höhere Kompressionsrate als bei verlustfreien Verfahren gegen (hoffentlich akzeptable) Abweichungen zwischen originalen und rekonstruierten Daten.

Typische Anwendungen
Multimedia-Dateien: Bilder, Musik, Videos.

Ausnutzung der Ungenauigkeit menschlicher Wahrnehmung:

- Psychoakustische Effekte,
- Helligkeitswerte feiner wahrgenommen als Farbwerte,
- Rekonstruktionsfehler bei Details in Bildern meist nicht sichtbar.

Wichtige Kompressionsverfahren MP3, JPEG, MPEG

Typische Kompressionsraten Audio 10%, Fotos 5%, Video 1%.

JPEG

Grundidee

- Entworfen für Komprimierung von Fotos
- Feine Details (hochfrequente Bildkomponenenten) werden weniger stark wahrgenommen als grobe Strukturen (niederfrequente Bildkomponenten)

Kernkomponenten

- DCT: Diskrete Kosinustransformation; transformiert das Bild blockweise in eine Frequenzdarstellung.
- Quantisierung: Reduziert numerische Genauigkeit der Daten $\rightsquigarrow$ weniger Entropie; nicht verlustfrei reversibel; hohe Frequenzen werden stärker reduziert als niedrige.
- RLE+Huffman-Codierer: Effiziente Komprimierung der quantisierten Daten.

JPEG Beispiel

Unkomprimiertes Bild: 512 × 512 Pixel, 256 KB

Abbildung: Schiffe am Strand

JPEG Beispiel

Komprimiertes Bild: 39 KB, Kompressionsrate 15%

Abbildung: Schiffe am Strand

JPEG Beispiel

Komprimiertes Bild: 6 KB, Kompressionsrate 2.3%

Abbildung: Schiffe am Strand (?)

Kryptologie

Wikipedia: Die Kryptologie (griechisch kryptós „versteckt, verborgen, geheim“) ist eine Wissenschaft, die sich mit Informationssicherheit beschäftigt.

Zentrales Thema: Nachrichtenübertragung in derart codierter Form ($\rightsquigarrow$ Begriff der **Verschlüsselung** bzw. **Chiffrierung**), dass ein unbefugter Abhörer eines solchen Codes möglichst nicht auf die eigentliche Nachricht rückschließen kann.

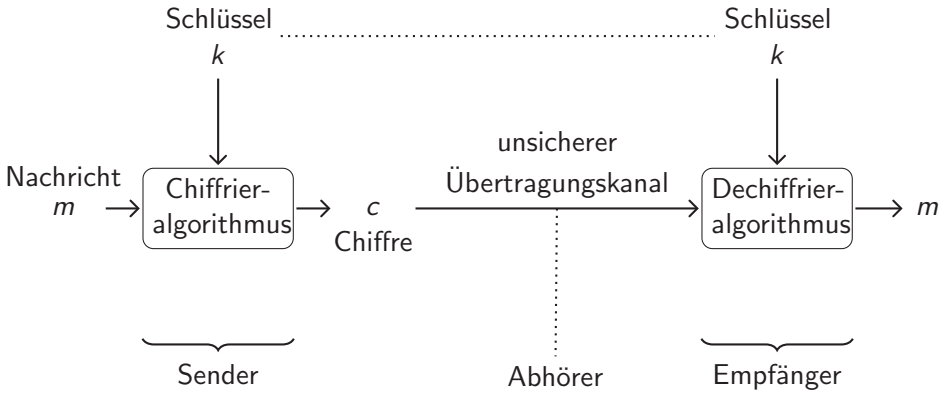
Unterteilung in zwei Teilgebiete:

Kryptographie: Wie man Nachrichten geeignet verschlüsselt.

Kryptoanalyse: Welche Methoden man aus Sicht eines unberechtigten Abhörers einsetzen kann, um evtl. doch an die verschlüsselte Nachricht heranzukommen.

Nebenbemerkung: Die **Steganographie**, sprich das Verstecken von Nachrichten in vermeintlich „harmlosen“ Objekten, wird nachfolgend nicht weiter betrachtet.

Kryptologie schematisch



Kryptologie im Altertum

Spartaner benutzten sog. „Skytalen“ zum Verschlüsseln militärischer Botschaften.

Idee: Lederstreifen mit Buchstaben um Holzstab wickeln...; Entschlüsselung nur mit gleich dickem Holzstab. Prinzip der **Transposition**, d.h. verschlüsselte Nachricht enthält genau die gleichen Buchstaben wie Originalnachricht, nur „umsortiert“.

Julius Cäsar beschrieb in seinen „Anmerkungen zum gallischen Krieg“ ein anderes Verfahren: **Substitutionschiffre**. Jeder Buchstabe durch einen entsprechend verschobenen Buchstaben ersetzt.

Beispiel: Geheimtext „HAL“ gehört zum Originaltext „IBM“ (Verschiebung um eine Alphabetposition; vgl. Film „2001 – Odyssee im Weltraum“).

Sichere Verschlüsselung: „One-Time-Pad“

Annahmen:

- Nachricht ist binär codiert.
- Der Schlüssel wird nur einmal benutzt.
- Der Schlüssel ist eine völlig zufällig gewählte Binärzeichenkette, der genauso lange wie die Nachricht ist.
- Sender und Empfänger verfügen beide über diesen Schlüssel.

Dann: Erzeuge mit Hilfe des Schlüssels verschlüsselten Text (Chiffre) aus dem „Klartext“, in dem positionsweise das „exklusive Oder“ der zwei entsprechenden Bits genommen wird.
Beispiel: Klartext: 1100; Schlüssel: 0110; Chiffre: 1010.

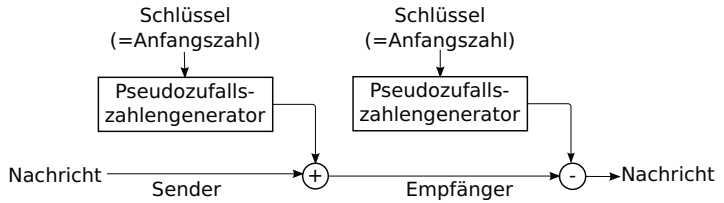
Feststellung (Shannon): Eine per obigem One-Time-Pad-Verfahren verschlüsselte Nachricht ist absolut sicher!
(Grund: Alle Chiffren kommen mit gleicher Wahrscheinlichkeit vor; keine statistischen Rückschlüsse auf Klartext möglich.)

Probleme: Schlüsselaustausch? Schlüssel so lang wie Klartext!

Verschlüsselung mit Pseudozufallszahlengeneratoren

Ansatz für kürzerere Schlüssel als bei One-Time-Pad:
Annahme: Sender und Empfänger besitzen beide den gleichen, „kryptographisch sicheren“ Pseudozufallszahlengenerator, der mit der gleichen Anfangszahl (das ist der Schlüssel!) initialisiert wird.

Dann: Erzeuge mit Hilfe des Pseudozufallszahlengenerators eine pseudozufällige Binärzeichenkette und verschlüssele und entschlüssele dann wie bei One-Time-Pad.



Nachteil: Sicherheit stark von Qualität des Pseudozufallszahlengenerators abhängig.
Anwendung z.B. beim Verschlüsseln eines Fernsehsignals beim Pay-TV.

Kerckhoff'sches Prinzip

Das Prinzip besagt, dass man in der Kryptologie grundsätzlich davon ausgehen muss, dass der verwendete Verschlüsselungs- und Entschlüsselungsalgorithmus (in den vorangehenden zwei Beispielen waren diese identisch) allgemein bekannt (also nicht geheim) sind!

Ein oft verwendeter Ansatzpunkt beim Knacken von Kryptosystemen: Häufigkeitsanalyse. Verschiedene Buchstaben kommen im Text verschieden häufig vor...; Analyse des chiffrierten Textes kann diese Häufigkeiten aufdecken...

$\rightsquigarrow$ Eine mögliche Abhilfe: Datenkompression (vgl. Huffman-Codierung). Benutze für häufiger vorkommende Buchstaben mehr als eine Codierung, um so die relativen Häufigkeiten anzugleichen...

Symmetrische vs. asymmetrische Kryptosysteme

Bislang haben wir symmetrische Kryptosysteme betrachtet: Sender und Empfänger verwenden den gleichen Schlüssel.

Zentrales, hierbei ausgeblendetes Problem: Schlüsselvereinbarung.

In der **modernen Kryptologie** spielen die von Whitfield Diffie und Martin Hellman erstmals 1976 (als Konzept) publizierten asymmetrischen Verfahren eine zentrale Rolle:

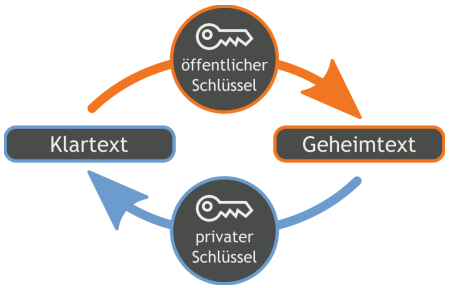
Public Key-Kryptographie: Jeder Teilnehmer am Nachrichtenaustausch besitzt einen *öffentlichen* und einen *privaten* Schlüssel. Ersterer ist jedermann zugänglich und dient zum Verschlüsseln von Nachrichten die an den Besitzer des öffentlichen Schlüssels geschickt werden, die nur dieser mit seinem geheimen Schlüssel dechiffrieren kann.

Zentral wichtig für die Realisierung: Erkenntnisse der *Zahlentheorie*!

Zusätzliche Anwendungen: Digitale Signaturen, Elektronisches Geld, Commitment Schemes, ...

Allgemeine Funktionsweise von Public-Key-Kryptosystemen

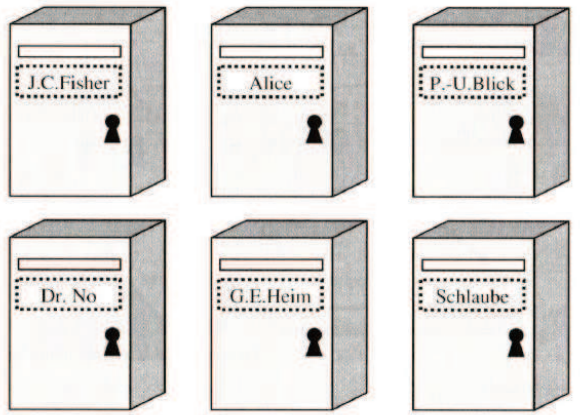
Verschlüsseln kann **jeder** mit einem öffentlichen Schlüssel; entschlüsseln **nur** der Inhaber des zugehörigen privaten Schlüssels.
$\rightsquigarrow$ *asymmetrisches* Kryptosystem.



Mögliche Nutzweise: A erzeugt privaten Schlüssel S_p sowie öffentlichen Schlüssel S_o und stellt S_o auf Homepage online. $\rightsquigarrow$ Jeder kann A eine mit S_o verschlüsselte Nachricht schreiben, die nur A mit S_p entschlüsseln kann.

Verwendung: im E-Mail-Verkehr (OpenPGP, S/MIME), in Protokollen wie SSH, SSL/TLS oder https, ...

Public Key: Briefkastenanalogie



Jeder kann Briefe einwerfen (mit öffentlichen Schlüssel verschlüsseln). Aber öffnen des Briefkastens nur mit (privatem) Schlüssel möglich!

RSA-Verschlüsselung

Shamir, Rivest, Adleman

Prominentestes asymmetrisches Kryptosystem: RSA-Verschlüsselung (nach Ronald **R**ivest, Adi **S**hamir, Leonhard **A**dleman, 1978 – damals alle drei am MIT; belohnt mit Turing Award 2002).

Verschlüsseln eines Textes m (als Zahl interpretiert!) um Geheimtext c zu erhalten:

$$c \equiv m^e \mod N$$

Entschlüsseln eines Geheimtextes c um Text m zu erhalten:

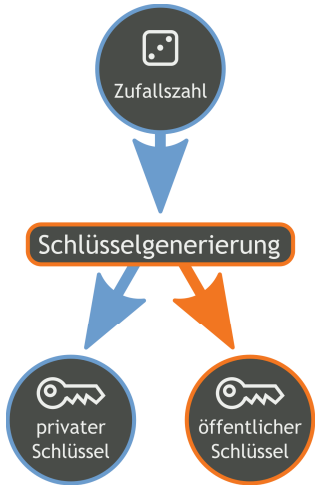
$$m \equiv c^d \mod N$$

$\rightsquigarrow$ Geheimer Schlüssel ist das Paar (d, N), öffentlicher Schlüssel das Paar (e, N).

Frage: Wie werden diese Schlüssel erzeugt?

Schlüsselerzeugung für RSA

Frage: Wie werden die Schlüssel (d, N) und (e, N) erzeugt?



1. Nimm zwei große (mehrere hundert Stellen) *zufällige* Primzahlen p und q.
2. $N := pq$
3. Wähle $1 < e < (p-1)(q-1)$ teilerfremd zu $(p-1)(q-1)$
4. Wähle d sodass $e \cdot d \equiv 1 \mod (p-1)(q-1)$

$\rightsquigarrow$ Sicherheit basiert auf der Annahme, dass die Zerlegung großer ganzer Zahlen in ihre Primfaktoren *nicht effizient* geht.
Anmerkung: Viele benutzte Primzahltests sind randomisiert wie z. B. der „Miller-Rabin-Test“.

„Falsche“ Schlüsselerzeugung bei RSA

Frage: Was passiert wenn Primzahlen p und q nicht zufällig gewählt werden?
Beispiel: *heise Security 13.05.2008:* „Die OpenSSL-Bibliothek der Linux-Distribution Debian erzeugt seit einem fehlerhaften Patch im Jahr 2006 schwache Krypto-Schlüssel. Der Sicherheitsexperte Luciano Bello entdeckte nun in dem OpenSSL-Paket eine kritische Schwachstelle, die die **erzeugten Zufallszahlenfolgen** und somit die erzeugten Schlüssel vorhersagbar macht.“

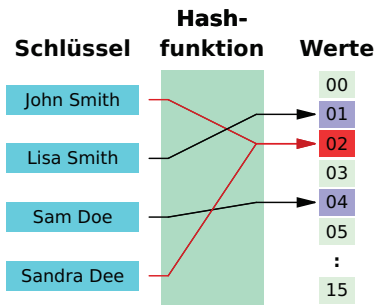
$\rightsquigarrow$ Zentral wichtiger Aspekt: Wie gut ist mein Zufallszahlengenerator?

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```


Hashfunktionen

Funktion die von einer großen (potentiell unendlichen) Schlüsselmenge (z. B. alle Binärzeichenketten) in eine kleinere Wertemenge (z. B. Binärzeichenkette mit max. 512 Bit) abbildet.

Folgerung: Es gibt Schlüssel mit selbem Hashfunktionswert (nicht injektiv).
Anforderung: Für gegeben Hashfunktionswert $f(x)$ soll es schwer sein ein x' zu finden mit $f(x') = f(x)$. Beachte dabei dass mgl. $x \neq x'$.



Digitale Signatur I

Zentrale Frage: Ist eine erhaltene Datei wirklich von einem bestimmten, angegebenen Absender? Wie beweise ich, dass eine Nachricht wirklich von mir kommt?

Antwort im analogen Schriftverkehr: Unterschriften!

$\rightsquigarrow$ Wie können digitale Unterschriften/Signaturen realisiert werden?

Anforderung an digitale Signatur:

1. Besitzer A des Dokuments muss spezifische Unterschrift erzeugen können.
2. Unterschrift darf nicht vom Dokument getrennt werden können.
3. Außenstehende müssen Unterschrift verifizieren können.

Digitale Signatur II

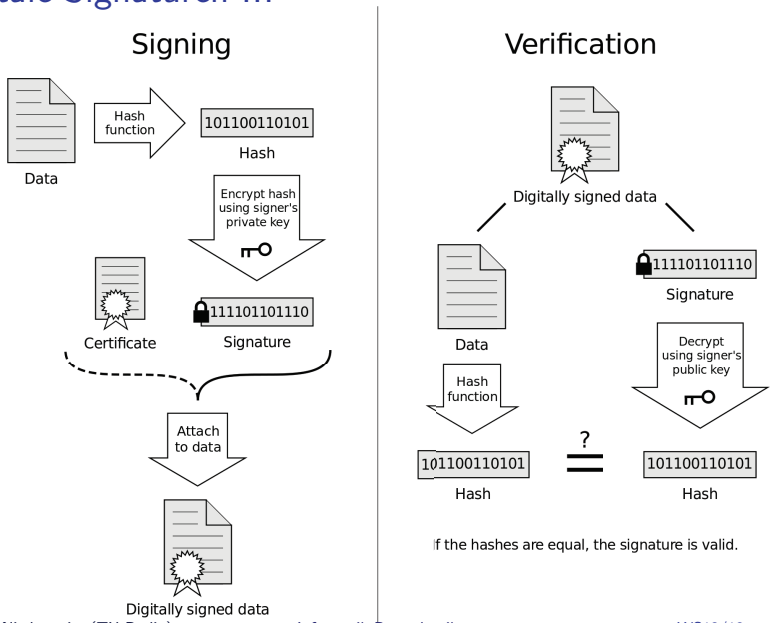
Realisierung mittels asymmetrischer Verschlüsselungsverfahren (z.B. RSA):

1. A hat einen geheimen Schlüssel.
2. Mit dem geheimen Schlüssel wird das Dokument „entschlüsselt“.
3. Mittels des öffentlichen Schlüssels wird das Dokument „verschlüsselt“ (also das Originaldokument wiederhergestellt).
4. Stimmt das wiederhergestellte Dokument mit dem im Klartext versandten Dokument überein, ist die Signatur verifiziert.

Beachtet werden muss:

1. Aus Klartext *und* Geheimtext darf der private Schlüssel nur „schwer“ rekonstruierbar sein.
2. Der öffentliche Schlüssel muss eindeutig dem Absender zugeordnet sein. $\rightsquigarrow$ öffentlicher Schlüssel mittels Zertifizierungsstelle überprüft.
3. Daten doppelt verschickt (als Klar- und Geheimtext). $\rightsquigarrow$ Verwendung von Hashfunktionen um statt des ganzen Dokuments nur den kurzen (eindeutigen) Hashwert zu ent- und verschlüsseln.

Digitale Signaturen III



Aufteilen und Verteilen von Geheimnissen

Ziel: Verteile Geheimnis so auf fünf Parteien, dass man beliebige drei von ihnen zusammenbringen muss, um das Geheimnis zu lüften (z.B. Code zum Öffnen eines Tresors).

Lösung nach Adi Shamir: Betrachte allgemeine Parabelfunktion:
$f(x) = ax^2 + bx + c$.
Bekannt: Kennt man drei Stützstellen (x-Werte für die der Funktionswert $f(x)$ bekannt ist) von f, dann ist f dadurch eindeutig bestimmt.

Weise jeder Partei genau einen Punkt (x-Koordinate verschieden von Null) auf der Parabel zu. Das Geheimnis sei dann als Funktionswert an der Stelle $x = 0$ codiert.

Klar: Auf beliebige Konstellationen (Anzahl beteiligter Parteien, ...) verallgemeinerbar!

Adi Shamir, 1952–

Commitment Scheme

Ziel: Auf eine Ausschreibung hin sollen mehrere Konkurrenten ihr Angebot bis zu einem bestimmten Ausschlusstermin einreichen. Vermeide, dass ein Konkurrent das Angebot eines anderen vorzeitig erfährt.

Lösung: Jeder Bieter schickt an den Ausschreiber sein Angebot verschlüsselt. Erst nach dem Ausschlusstermin wird dann der jeweilige Schlüssel zum Dekodieren des Angebots geschickt!

Dazu muss sichergestellt sein, dass die Verschlüsselungsfunktion injektiv ist (damit es zu einem Chiffre-Text höchstens einen passenden Klartext gibt).

Quantenkryptologie

Ziel: Sichere Schlüsselübertragung für symmetrische Verfahren.

Idee: Nutze quantenphysikalische Effekte von Photonen (Heisenberg'sche Unschärferelation) zur sicheren Signalübertragung.

Entscheidend: Jedwedes Messen bzw. Mithören eines solchen Signals verändert es auch. $\rightsquigarrow$ Ein Abhörer wird bemerkt!

Anwendung zum Beispiel zur sicheren Schlüsselübertragung für One-Time-Pad.

Computational Thinking

In dem Artikel „Computational Thinking“ (Communications of the ACM 2006) propagierte Jeannette M. Wing (damals Professorin an der Carnegie Mellon University, Pittsburgh; seit Ende 2012 Chefin von Microsoft Research International) dass „jedermann“, nicht nur Informatiker, davon profitieren kann informatisch (algorithmisch) zu denken.

Jeannette M. Wing, 1957–

Ein Definitionsversuch: „*Computational Thinking* is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.“ [CunySnyderWing 2010]

Nunmehr unterstützt von Unternehmen wie Google und Microsoft.

Motivation Computational Thinking

Informatikprodukte beeinflussen alle Lebensbereiche: Internet, Suchmaschinen, Mobiltelefonie, Electronic Banking, Einkaufen im Internet, Entzifferung des menschlichen Genoms, Klimavorhersage, Navigationssysteme, soziale Netzwerke, Wikipedia, Digitalkameras, ...

Oft vollziehen sich damit einhergehende Veränderungen in rapidem, bislang nicht gekanntem Tempo... Z.B.:

Wieviele Buchläden wird es in zehn Jahren noch geben? Was ist die Zukunft von Bibliotheken?

Wer kauft noch Musik-CDs im Laden?

Wieviele von Menschen verrichtete Arbeiten werden bald durch Roboter oder Informatiksysteme ersetzt (inkl. Pilot, U-Bahnfahrer, Verwaltungsbeamte, Lehrer, Vorleser, ...)?

Wer bucht seine Reise noch im Reisebüro?

Wer merkt sich noch Telefonnummern?

Zukunftstrends (dank Informatik)

- Automatisierung.
- Kommunikation.
- Datenanalyse.
- Informatik schafft Werkzeuge zur „Intelligenzverstärkung“, nicht nur zur „Kraftverstärkung“.

Credo: Wer bei Zukunftstrends mitreden und vielleicht auch mitgestalten will, braucht ein solides Grundverständnis, braucht Computational Thinking! Z.B. Einsicht in

- Strategien des Problemlösens,
- Prinzipien des Systementwurfs und der Systemanalyse,
- menschliches Verhalten.

Trend Automatisierung

Beispiel Briefe:

„Früher“: Brief in Briefkasten werfen → Einsammeln der Briefe und Transport zur Postzentrale → Sortieren → Transport zur passenden Zielortzentrale → Sortieren → Brief austragen.

„Heute“: Email: Click...

Weitere Beispiele:

- Reiseplanung: Internet statt Reisebüro.
- Fotos: Speicherchip statt Film.
- Hochindividuell gestaltbare technische Produkte wie Autos.
- Navigation statt Kartenlesen.

Trend Kommunikation

Radikale Verhaltensänderungen durch Dinge wie

- Email,
- World Wide Web,
- Smartphone,
- Soziale Netzwerke,
- Internetbasierter Einkauf und Handel.

Geschwindigkeit und Umfang von damit verknüpften Diensten steigt, Preis sinkt!

Trend (massive) Datenanalyse

Massive Datenanalyse ermöglicht

- Maschinelle Sprachübersetzung;
- Maschinensteuerung per Gedanken;
- Entzifferung des menschlichen Genoms (bestehend aus 6 Milliarden Buchstaben);
- Personalisierte Werbung aber auch personalisierte Medizin;
- Aufdecken von Zusammenhängen in der Literatur;
- Empfehlungssysteme („Wer dieses Buch gekauft hat, der hat auch...“)
- ...

Aspekte des Computational Thinkings

Computational Thinking umfasst bzw. verbindet

- algorithmisches Denken;
- mathematisches Denken;
- logisches Denken;
- Denken in Systemen;
- paralleles und verteiltes Denken;
- ingenieurmäßiges Denken.

Eine wesentliche Besonderheit: Mit Software entstehen virtuelle Welten, die nicht durch die physikalische Realität beschränkt sind.

Wichtigster Aspekt des Computational Thinkings: **Abstraktion**! Hinzu kommen weiterhin Rekursion und Hierarchisierung, Automatisierung, Analyse.

Thesen zum Computational Thinking

Computational Thinking beansprucht für sich folgende Charakteristiken:

- Denken in Konzepten und Abstraktionen, nicht bloß Programmieren.
- Ergänzt und kombiniert mathematisches und ingenieurmäßiges Denken.
- Ideen zählen, nicht bloß Artefakte (Gegenstände).
- Grundlegende Fähigkeiten statt Routine sind nötig.
- Mit Computational Thinking (und Informatik) kann man in *allen* Bereichen Erfolg haben, von Medizin über Politik bis Kunst.
- Ist Grundlage zur Lösung vieler intellektuell herausfordernder, zentraler wichtiger wissenschaftlicher Probleme, die der Lösung harren.

Computational Thinking im Alltag

Drei Beispiele:

Sortieren: Wer im häuslichen Keller Muttis umfangreiche Plattensammlung nach Komponisten alphabetisch sortieren soll, wird die Vorzüge von „Bucket Sort“ oder „Quicksort“ zu schätzen lernen...

Hashing: Wer die Legosteinesammlung seiner Nichte strukturieren will, mag gerne auf Hashing zurückgreifen... (z.B. Kategorien wie „quadratisch und dick“, „nichtquadratisch und dick“, ...)

Pipelining: Wer hat sich nicht schon mal über die „dumme“ Anordnung von einzelnen Stationen in einer Mensa aufgeregt (z.B. ist es „vernünftig“ (wie in der TU Mensa), dass man das Besteck erst nach der Kasse nimmt...)

Computational X

Ein starker Beleg für den allumfassenden Anspruch des Computational Thinkings ist das Herauswachsen von „Bindestrichdisziplinen“ aus der Informatik (im englischsprachigen Raum oft mit dem Präfix „Computational“ versehen – was im Deutschen manchmal einschränkend mit „algorithmisch“ übertragen wird), sprich „Computational X“, wobei X für Gebiete steht wie

Biologie, Chemie, Geologie, Medizin, Wirtschaft, Spieltheorie, Geometrie, Recht, Politik und Sozialwissenschaft („Computational Social Choice“), Linguistik, Archäologie, ...

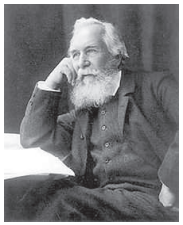
Nachfolgend gehen wir kurz näher ein auf Algorithmische Biologie, Computational Social Choice und Algorithmische Spieltheorie.

Algorithmische Biologie

Algorithmische Biologie betrachtet die Berechnungskomplexität und die algorithmischen Eigenschaften von Berechnungsproblemen aus der Bioinformatik.

Beispiel: Rekonstruktion von phylogenetischen Bäumen

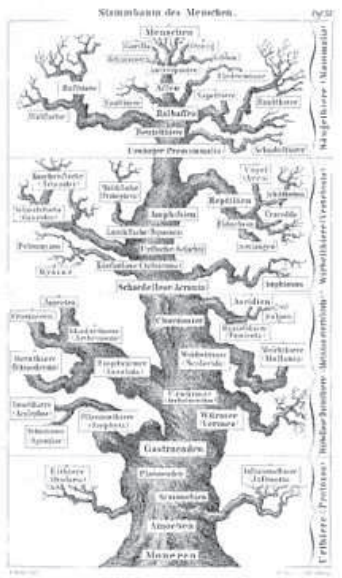
Ernst Heinrich Philipp August Haeckel (1834-1919) prägte den Begriff „Stamm“.



- Deutscher Philosoph und Zoologe;
- verbreitete die Lehren Charles Darwins in Deutschland;
- maßgebliche Arbeiten zur Abstammungslehre.

> *... the great Tree of Life fills with its dead and broken branches the crust of the earth, and covers the surface with its ever-branching and beautiful ramifications.*
>
> — Charles Darwin

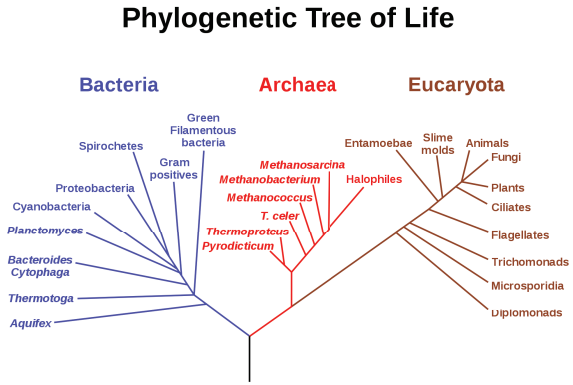
Stammbaum des Menschen nach Haeckel



Rekonstruktion von phylogenetischen Bäumen I

Gegeben eine Menge von verschiedenen biologischen Objekten und Beobachtungen über deren evolutionäre Zusammenhänge.

Ziel: Modell in Form eines Baumes (Graph) der evolutionären Relation zwischen den biologischen Objekten. Blätter sind die Objekte und innere Knoten stellen die evolutionären Zusammenhänge dar.



Evolutionärer Zusammenhang wird z. B. über „Editierdistanz“ zwischen den entsprechenden DNS/RNS Sequenzen oder Proteinen gemessen.

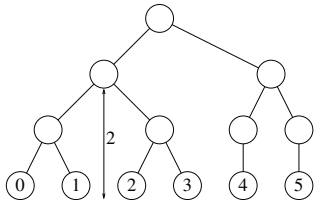
Rekonstruktion von phylogenetischen Bäumen II

Formalisieren der Problemstellung:

Eingabe: Symmetrische Distanzfunktion $d : V \times V \to \mathbb{N}$ für Objektmenge V.

Aufgabe: Finde eine Darstellung von d in einem **ultrametrischen Baum**.

	1	2	3	4	5
0	1	2	2	3	3
1	-	2	2	3	3
2	-	-	1	3	3
3	-	-	-	3	3
4	-	-	-	-	2



Distanz im ultrametrischen Baum ist definiert über die Höhe des „tiefsten gemeinsamen Vorfahrens“.

Mitteilung: Distanzfunktion d kann als ultrametrischer Baum dargestellt werden gdw. d eine **Ultrametrik** ist, d. h.
$d(a, b) \leq \max\{d(a, c), d(b, c)\}$ für alle $a, b, c \in V$.

Computational Social Choice

Social Choice-Theorie studiert die Mechanismen des kollektiven Entscheidungsfindung wie Wählen, Aggregieren von Präferenzen, faires Teilen, Zuordnungsprobleme (Matching), ...

- Vorläufer: Cordorcet, Borda (18. Jahrhundert),...
- „Ernsthafte“ wissenschaftliche Disziplin seit den 1950er Jahren.

Computational Social Choice fügt dem ganzen eine algorithmische Perspektive hinzu, und untersucht auch die Anwendung von Konzepten aus dem Bereich „Social Choice“ in der Informatik und angrenzenden Gebieten.

- „Klassische“ Veröffentlichungen beginnen $\sim$ 1990.
- Aktives Forschungsgebiet mit regelmäßigen Veröffnetlichungen seit etwa 2002.
- Name „COMSOC“ und alle zwei Jahre stattfindende Spezialtagung gleichen Namens.

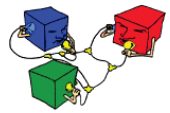
Wir konzentrieren uns hier auf „Wahlprobleme“.

Wo Wahlprobleme auftreten...

Politische Wahlen. Viele Wähler und wenige Kandidaten.

Bundestagswahl 2005

Multiagentensysteme. Einigung auf einen gemeinsamen Arbeitsplan.



Suchmaschinen im Internet. Rank Aggregation mehrerer Suchmaschinen;

bing + Google + YAHOO! + ...

Empfehlungsdienste. Empfehlung von Produkten, z.B. Bücher oder Videos, basierend auf Rankings anderer Benutzer;

amazon.com

Wahlen und Wahlregeln

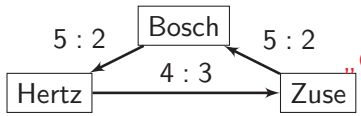
In einer **Wahl** handelt es sich um eine gemeinsame Entscheidung mehrerer **Wähler** zwischen verschiedenen **Kandidaten** gemäß einer **Wahlregel**.

Beispiel (Wissenschaftlerranking an der TU Berlin)

2 Studenten:	Bosch	$\succ$	Hertz	$\succ$ Zuse
2 Studenten:	Hertz	$\succ$	Zuse	$\succ$ Bosch
3 Studenten:	Zuse	$\succ$	Bosch	$\succ$ Hertz

Zwei Wahlregeln:

1. **Mehrheitswahl:** Derjenige Kandidat gewinnt, der am häufigsten an erster Stelle in den Stimmabgaben steht. **Gewinner: Zuse.**
2. **Condorcet-Verfahren:** Derjenige Kandidat gewinnt, der gegen jeden anderen Kandidaten die Mehrheit im direkten Vergleich erhält.



„Condorcet-Zykel“ $\Rightarrow$ Kein Gewinner!

Wünschenswerte Eigenschaften von Wahlregeln

- **Condorcet-Eigenschaft.** Wenn es einen Condorcet-Gewinner gibt, dann soll dieser auch die Wahl gewinnen.
- **Neutralität.** Alle Kandidaten werden gleich behandelt.
- **Unabhängigkeit von unbedeutenden Kandidaten**, engl. **Independence of Irrelevant Alternatives (IIA)**. Wenn ein Kandidat x Gewinner und y kein Gewinner einer Wahl W_1 ist und alle Wähler in W_1 und einer weiteren Wahl W_2 (mit gleicher Kandidatenmenge) darin übereinstimmen, dass x besser ist als y, dann kann y auch kein Gewinner in W_2 sein.
- **Konsistenz.** Wenn ein Kandidat in zwei Wahlen gewinnt, dann muss dieser auch der Gewinner in der Vereinigung beider Wahlen sein.

Im folgenden betrachten wir die einzige Wahlregel, die alle diese Eigenschaften besitzt.

Das Kemeny-Verfahren I

Mindestens dreimal unabhängig voneinander beschrieben:

- Condorcet (1785) für 3 Wähler
- Kemeny (1959), auch Erfinder der Programmiersprache Basic
- Fobes (1991) www.votefair.org

Beim **Kemeny-Verfahren** sucht man eine „Konsensliste“, die die „Gesamtmeinung“ aller Präferenzlisten am besten wiedergibt.

Anwendungen:

- Metasuchmaschinen,
- Suche in Datenbanken,
- Bewerberauswahl/Auswahl von Studenten,
- Bioinformatik,
- ...

Das Kemeny-Verfahren II

Der **Kendall-Tau-Abstand** zwischen zwei Präferenzlisten ist die Anzahl der unterschiedlich geordneten Kandidatenpaare.

Beispiel: Bosch $\succ$ Hertz $\succ$ Zuse
Hertz $\succ$ Zuse $\succ$ Bosch
Konfliktpaare: {Bosch, Hertz} und {Bosch, Zuse}.
Der Abstand ist also 2.

Die **Kemeny-Punktezahl** einer Liste ist die Summe der Abstände zu allen Präferenzlisten.

Eine **Kemeny-Konsensliste** ist eine Liste mit kleinstmöglicher Punktezahl.

Eingangsbeispiel:
2 Studenten: Bosch $\succ$ Hertz $\succ$ Zuse
2 Studenten: Hertz $\succ$ Zuse $\succ$ Bosch
3 Studenten: Zuse $\succ$ Bosch $\succ$ Hertz

Die einzige Kemeny-Konsensliste ist „Zuse $\succ$ Bosch $\succ$ Hertz“ mit 8 Punkten.

Das Kemeny-Verfahren III

Frage: Wie findet man eine Liste mit kleinster Kemeny-Punktzahl?

Naive Lösung: Alle $m!$ möglichen Permutationen der m Kandidaten durchprobieren.
$\rightsquigarrow$ Exponentielle Laufzeit!

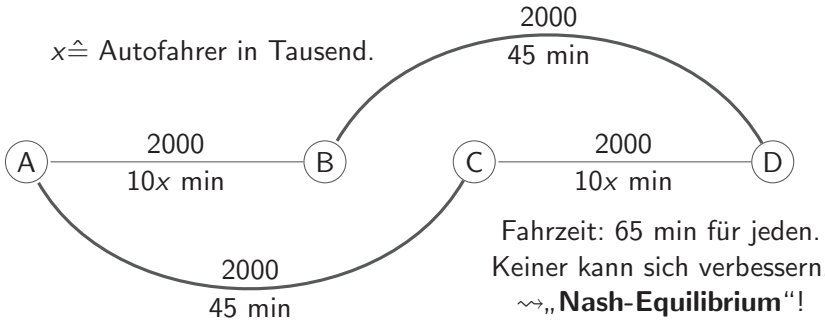
Mitteilung:

- NP-schwer, allerdings noch effizient lösbar für wenige Kandidaten;
- Immer noch NP-schwer, wenn es nur vier Wähler gibt.

Algorithmische Spieltheorie I - Nash-Equilibrium

Wikipedia: In der Spieltheorie werden Entscheidungssituationen modelliert, in denen sich mehrere Beteiligte gegenseitig beeinflussen.

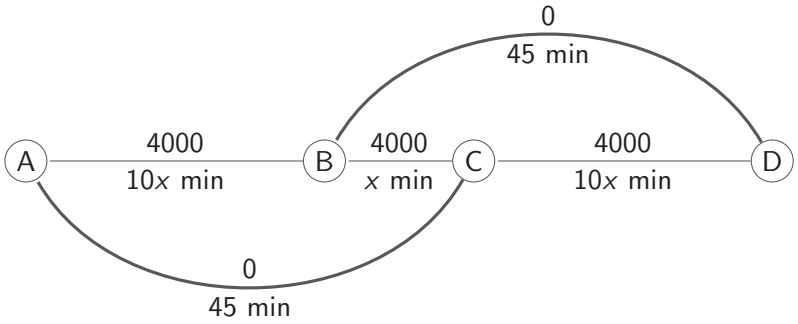
Bsp. Verteilung von Autofahrern in einem Straßennetz:



Zur Feierabendzeit wollen 4000 Pendler von A nach D fahren.
$\rightsquigarrow$ was ist der schnellste Weg?
Klar: Fahrtzeit hängt von Verkehrsdichte ab.

Algorithmische Spieltheorie II - Braess-Paradoxon

fortgesetztes Beispiel: Zwischen B und C wird eine große Brücke gebaut.



Aus Pendlersicht: Autobahn ist nicht der schnellste Weg zwischen A und C bzw. B und D.
$\rightsquigarrow$ Keiner nutzt die Autobahn.
$\rightsquigarrow$ Fahrzeit erhöht sich auf 84 min für jeden Pendler.
Ausbau des Verkehrsnetzes erhöht Fahrzeit! $\rightsquigarrow$ **Braess-Paradoxon!**

Algorithmische Spieltheorie III - Fazit

Vorheriges Beispiel zeigt: Optimiert jeder Fahrer nur die eigene Fahrzeit (**rational egoistisches Vorgehen**), muss dies nicht zu einem **globalem Optimum** führen!

Algorithmische Spieltheorie befasst sich mit der Fragen wie z.B.:

- Wie können solche Nash-Equilibria oder globalen Optima)effizient) berechnet werden?
- Was ist eine optimale Strategie für einen Agenten?
- Ist in einem gegebenen Szenario jedes Nash-Equilibrium ein globales Optimum?

Anwendungen:

- Auktionen, Elektronischer Handel
- Routing (z.B. im Internet)
- Verkehrsplanung
- ...

Erfolgreiches Studieren

Chinesisches Sprichwort:

> *„Tell me and I forget. Show me and I remember. Involve me and I understand.“*