Theoretische Informatik 3
WS 2006/07

Volker Strehl
Informatik 8

19. Oktober 2006

Organisatorisches

- Vorlesungstermine

  Montags und Donnerstag, 16:00–17:30 Uhr im H9

Organisatorisches

- Vorlesungstermine

  Montags und Donnerstag, 16:00–17:30 Uhr im H9
- Übungen in 7 Gruppen, Termine und Eintragung → Webseite

Organisatorisches

- Vorlesungstermine

  Montags und Donnerstag, 16:00–17:30 Uhr im H9
- Übungen in 7 Gruppen, Termine und Eintragung → Webseite
- Webseite:
  `www8.informatik.uni-erlangen.de/IMMD8/Lectures/THINF/`
  enthält Materialien zur Vorlesung, Übungsaufgaben, aktuelle Informationen etc. (NB: nicht alles ist von aussen zugänglich)

Organisatorisches

- Vorlesungstermine

  Montags und Donnerstag, 16:00–17:30 Uhr im H9
- Übungen in 7 Gruppen, Termine und Eintragung → Webseite
- Webseite:
  `www8.informatik.uni-erlangen.de/IMMD8/Lectures/THINF/`
  enthält Materialien zur Vorlesung, Übungsaufgaben, aktuelle Informationen etc. (NB: nicht alles ist von aussen zugänglich)
- Kontakt:
  - Adresse: Haberstrasse 2, 3. Stock
  - Telefon: 85-28712
  - email: `strehl@cs.fau.de`
  - Sprechstunde: Termin per email vereinbaren

Empfohlene Literatur

- M. Aigner, Diskrete Mathematik, Vieweg, 2005 (5. Aufl.).
- T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press, 2001 (2. A.).
- R. Graham, D. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, 1994 (2. A.).
- V. Heun, Grundlegende Algorithmen, Vieweg, 2003 (2. A.).
- D. Knuth, The Art of Computer Programming (1–3), Addison-Wesley, 1962–1997.
- U. Schöning, Algorithmik, Spektrum-Verlag, 2001.
- R. Sedgewick, P. Flajolet, An Introduction to the Analysis of Algorithms, Addison-Wesley, 1996.
- H. Wilf, Algorithms and Complexity, Prentice-Hall 1986.

Übersicht

- Einleitung – Begriffe und Beispiele

Übersicht

- Einleitung – Begriffe und Beispiele
- Mathematische Hilfsmittel

Übersicht

- Einleitung – Begriffe und Beispiele
- Mathematische Hilfsmittel
- Exemplarische Analysen

Übersicht

- Einleitung – Begriffe und Beispiele
- Mathematische Hilfsmittel
- Exemplarische Analysen
- Rekursion und Komplexität

Übersicht

- Einleitung – Begriffe und Beispiele
- Mathematische Hilfsmittel
- Exemplarische Analysen
- Rekursion und Komplexität
- Information und Komplexität

Übersicht

- Einleitung – Begriffe und Beispiele
- Mathematische Hilfsmittel
- Exemplarische Analysen
- Rekursion und Komplexität
- Information und Komplexität
- Arithmetik und Komplexität

Teil I

Einleitung

Theoretische Informatik 3 WS 2006/07

Probleme

- Was ist ein Problem?

Theoretische Informatik 3 WS 2006/07
Szenario
Was ist ein Problem?

Probleme

- Was ist ein Problem?
  - D = domain = Definitionsbereich = "Instanzen"

Probleme

- Was ist ein Problem?
  - D = domain = Definitionsbereich = "Instanzen"
  - R = range = Wertebereich

Theoretische Informatik 3 WS 2006/07
Szenario
Was ist ein Problem?

Probleme

- Was ist ein Problem?
  - D = domain = Definitionsbereich = "Instanzen"
  - R = range = Wertebereich
  - $f : D \to R$ berechenbare Funktion

Theoretische Informatik 3 WS 2006/07
Szenario
Was ist ein Problem?

Probleme

- Was ist ein Problem?
  - D = domain = Definitionsbereich = "Instanzen"
  - R = range = Wertebereich
  - $f : D \to R$ berechenbare Funktion
  - falls $R = \{\texttt{true}, \texttt{false}\}$: Entscheidungsproblem

Theoretische Informatik 3 WS 2006/07
Szenario
Was ist ein Problem?

Probleme

- Was ist ein Problem?
  - D = domain = Definitionsbereich = "Instanzen"
  - R = range = Wertebereich
  - $f : D \to R$ berechenbare Funktion
  - falls $R = \{\texttt{true}, \texttt{false}\}$: Entscheidungsproblem
  - Grösse von Instanzen $\texttt{size} : D \to \mathbb{N}$

Probleme

- Was ist ein Problem?
  - D = domain = Definitionsbereich = "Instanzen"
  - R = range = Wertebereich
  - $f : D \to R$ berechenbare Funktion
  - falls $R = \{\texttt{true}, \texttt{false}\}$: Entscheidungsproblem
  - Grösse von Instanzen $\texttt{size} : D \to \mathbb{N}$
  - oft: $D_n = \{d \in D ; \texttt{size}(d) = n\}$ ist endlich für $n \geq 0$, dann $d_n = \sharp D_n$

Kosten

- Algorithmen und Kosten

Kosten

- Algorithmen und Kosten
  - $\mathcal{A}$: Algorithmus zur Berechnung von $f : D \to R$ (auf TM oder RAM oder dergl.), d.h.

$$\forall d \in D : \mathcal{A}(d) = f(d)$$

Kosten

- Algorithmen und Kosten
  - $\mathcal{A}$: Algorithmus zur Berechnung von $f : D \to R$ (auf TM oder RAM oder dergl.), d.h.

$$\forall d \in D : \mathcal{A}(d) = f(d)$$

  - Kostenfunktion (modellabhängig)

$$\texttt{cost}_{\mathcal{A}} : D \to \mathbb{N} : d \mapsto \texttt{cost}_{\mathcal{A}}(d)$$

    soll Laufzeitverhalten (oder Speicherverbrauch) von $\mathcal{A}$ auf einem Maschinenmodell (TM,RAM) anhand der bestimmenden Einflussgrössen (z.B. Anzahl der elementaren Rechenschritte einer TM, Anzahl der arithmetischen oder Vergleichsoperationen einer RAM) widergeben

- worst-case und average-case

- worst-case und average-case
  - worst-case Komplexität

$$\mathtt{cost}_{\mathcal{A}}(n) = \max_{d \in D_n} \mathtt{cost}_{\mathcal{A}}(d)$$

- worst-case und average-case
  - worst-case Komplexität

$$\mathtt{cost}_{\mathcal{A}}(n) = \max_{d \in D_n} \mathtt{cost}_{\mathcal{A}}(d)$$

  - average-case Komplexität

$$\overline{\mathtt{cost}_{\mathcal{A}}}(n) = \frac{1}{d_n} \sum_{d \in D_n} \mathtt{cost}_{\mathcal{A}}(d)$$

- worst-case und average-case
  - worst-case Komplexität

$$\mathtt{cost}_{\mathcal{A}}(n) = \max_{d \in D_n} \mathtt{cost}_{\mathcal{A}}(d)$$

  - average-case Komplexität

$$\overline{\mathtt{cost}_{\mathcal{A}}}(n) = \frac{1}{d_n} \sum_{d \in D_n} \mathtt{cost}_{\mathcal{A}}(d)$$

  - Hauptproblem:
    Wie skaliert der Berechnungsaufwand mit der Problemgrösse?
    Wie verhalten sich die Folgen $(\mathtt{cost}_{\mathcal{A}}(n))_{n \geq 0}$ bzw. $(\overline{\mathtt{cost}_{\mathcal{A}}}(n))_{n \geq 0}$ asymptotisch, d.h. für $n \to \infty$?

Zwei wichtige Bemerkungen

- Eine exakte Bestimmung ("Formel") von $\texttt{cost}_{\mathcal{A}}(n)$ bzw. $\overline{\texttt{cost}_{\mathcal{A}}}(n)$ ist meist nicht möglich. Man muss sich mit "asymptotischen" Aussagen im Sinne der LANDAU-Notation begnügen, z.B. :

$$\texttt{cost}_{\mathcal{A}}(n) \in \mathcal{O}(n^2), \in \Omega(n^{3/2}), \in \Theta(n \log n), \sim 7n^{\log 3}, \ldots$$

Theoretische Informatik 3 WS 2006/07
Szenario
worst-case vs. average-case

Zwei wichtige Bemerkungen

- Eine exakte Bestimmung ("Formel") von $\texttt{cost}_{\mathcal{A}}(n)$ bzw. $\overline{\texttt{cost}_{\mathcal{A}}}(n)$ ist meist nicht möglich. Man muss sich mit "asymptotischen" Aussagen im Sinne der LANDAU-Notation begnügen, z.B. :

$$\texttt{cost}_{\mathcal{A}}(n) \in \mathcal{O}(n^2), \in \Omega(n^{3/2}), \in \Theta(n \log n), \sim 7n^{\log 3}, \ldots$$

- Bei rekursiven Algorithmen genügen die Aufwandsfunktionen "divide-and-conquer" Rekursionsgleichungen, z.B.

$$T(2n) = 7T(n) + \Theta(n^2)$$

Wie kann man daraus Aussagen über das asymptotische Verhalten von $T(n)$ gewinnen?

Wichtige Unterscheidung

- Komplexität von Algorithmen

Theoretische Informatik 3 WS 2006/07
Szenario
Komplexität von Algorithmen und von Problemen

Wichtige Unterscheidung

- Komplexität von Algorithmen
  - Jeder konkrete Algorithmus $\mathcal{A}$ für ein Problem $f : D \to R$ hat eine einem Berechnungsmodell bezüglich einer Grössenfunktion `size` und eines Komplexitätsmasses `cost` eine Komplexitätsfunktion.

Wichtige Unterscheidung

- Komplexität von Algorithmen
  - Jeder konkrete Algorithmus $\mathcal{A}$ für ein Problem $f : D \to R$ hat eine einem Berechnungsmodell bezüglich einer Grössenfunktion `size` und eines Komplexitätsmasses `cost` eine Komplexitätsfunktion.
- Komplexität von Problemen

Theoretische Informatik 3 WS 2006/07
Szenario
Komplexität von Algorithmen und von Problemen

Wichtige Unterscheidung

- Komplexität von Algorithmen
  - Jeder konkrete Algorithmus $\mathcal{A}$ für ein Problem $f : D \to R$ hat eine einem Berechnungsmodell bezüglich einer Grössenfunktion `size` und eines Komplexitätsmasses `cost` eine Komplexitätsfunktion.
- Komplexität von Problemen
  - Welchen Aufwand benötigt jeder Algorithmus zur Lösung eines Problems (in einem gegebenen Berechnungsmodell)? Das ist eine *inhärente* Eigenschaft des Problems!

- Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — "je kleiner, desto besser".

Theoretische Informatik 3 WS 2006/07
Szenario
Obere und untere Schranken

- Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — "je kleiner, desto besser".
- Untere Schranken treffen auf *alle* Algorithmen für ein Problem zu – "je grösser, desto besser" – sind aber meist sehr schwer zu gewinnen!

- Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — "je kleiner, desto besser".
- Untere Schranken treffen auf *alle* Algorithmen für ein Problem zu – "je grösser, desto besser" – sind aber meist sehr schwer zu gewinnen!
- Für *optimale* Algorithmen stimmt die Algorithmen-Komplexität mit einer unteren Schranke für die Problemkomplexität überein.

Theoretische Informatik 3 WS 2006/07
Szenario
Obere und untere Schranken

- Konkrete Algorithmen liefern obere Schranken für die Problemkomplexität — "je kleiner, desto besser".
- Untere Schranken treffen auf *alle* Algorithmen für ein Problem zu – "je grösser, desto besser" – sind aber meist sehr schwer zu gewinnen!
- Für *optimale* Algorithmen stimmt die Algorithmen-Komplexität mit einer unteren Schranke für die Problemkomplexität überein.
- Für die meisten Probleme sind gute untere Schranke nicht bekannt – und damit keine optimalem Algorithmen!

Begriffe

- *Komplexitätsanalyse* beschäftigt sich mit dem Entwurf der Analyse von möglichst effizienten Algorithmen für konkrete Probleme

Theoretische Informatik 3 WS 2006/07
Szenario
Komplexitätsanalyse und Komplexitätstheorie

Begriffe

- *Komplexitätsanalyse* beschäftigt sich mit dem Entwurf der Analyse von möglichst effizienten Algorithmen für konkrete Probleme
- *Komplexitätstheorie* untersucht die Komplexität von Problemen und die Komplexitäts-Beziehungen zwischen Problemen. *Komplexitätsklassen* (wie `P`, `NP`, `EXPTIME`, `PSPACE`) fassen Probleme "gleicher Schwierigkeit" zu Klassen zusammen und untersuchen Beziehungen zwischen diesen Klassen.

Zwei formale Sprachen

- Reguläre Sprache $F = (a + bb)^* \subseteq \{a, b\}^*$.
  Wieviele Wörter der Länge n enthält F?

Zwei formale Sprachen

- Reguläre Sprache $F = (a + bb)^* \subseteq \{a, b\}^*$.
  Wieviele Wörter der Länge n enthält F?
- Kontextfreie Sprache der korrekten Klammerungen $D \subseteq \{a, b\}^*$, erzeugt durch

$$D \to aDbD \mid \lambda.$$

  Wieviele Wörter der Länge n enthält D?

Zwei formale Sprachen

- Reguläre Sprache $F = (a + bb)^* \subseteq \{a, b\}^*$.
  Wieviele Wörter der Länge n enthält F?
- Kontextfreie Sprache der korrekten Klammerungen $D \subseteq \{a, b\}^*$, erzeugt durch

$$D \to aDbD \mid \lambda.$$

  Wieviele Wörter der Länge n enthält D?

- Allgemein:
  Σ endliches Alphabet, $L \subseteq \Sigma^*$ formale Sprache
  Wie verhält sich $\ell_n(L) := \sharp(L \cap \Sigma^n)$ für $n \to \infty$?
  Welche Rolle spielt dabei der Chomsky-Typ von L?

Beispiel: reguläre formale Sprache

- Für $F = (a + bb)^* \subseteq \{a, b\}^*$ gilt

$$\ell_0(F) = \ell_1(F) = 1$$
$$\ell_{n+2}(F) = \ell_{n+1}(F) + \ell_n(F) \quad (n \geq 0)$$

Beispiel: reguläre formale Sprache

- Für $F = (a + bb)^* \subseteq \{a, b\}^*$ gilt

$$\ell_0(F) = \ell_1(F) = 1$$
$$\ell_{n+2}(F) = \ell_{n+1}(F) + \ell_n(F) \quad (n \geq 0)$$

und daher

$$\ell_n(F) = F_{n+1} \quad \text{Fibonacci-Zahl}$$

Theoretische Informatik 3 WS 2006/07
Beispiele
Zwei formale Sprachen

Beispiel: reguläre formale Sprache

- Für $F = (a + bb)^* \subseteq \{a, b\}^*$ gilt

$$\ell_0(F) = \ell_1(F) = 1$$
$$\ell_{n+2}(F) = \ell_{n+1}(F) + \ell_n(F) \quad (n \geq 0)$$

und daher

$$\ell_n(F) = F_{n+1} \quad \text{Fibonacci-Zahl}$$

Asymptotisch exponentielles Wachstum:

$$F_n = \frac{\phi^n - \widehat{\phi}^n}{\sqrt{5}} \sim \frac{\phi^n}{\sqrt{5}}$$

mit $\phi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$ (goldener Schnitt),
$\widehat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803\ldots$

Theoretische Informatik 3 WS 2006/07
Beispiele
Zwei formale Sprachen

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \to aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen (äquivalent: Binärbäume) gilt

Theoretische Informatik 3 WS 2006/07
Beispiele
Zwei formale Sprachen

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \to aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen (äquivalent: Binärbäume) gilt $\ell_{2n+1}(D) = 0$ für $n \geq 0$ und für $d_n := \ell_{2n}(D)$

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \to aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen (äquivalent: Binärbäume) gilt $\ell_{2n+1}(D) = 0$ für $n \geq 0$ und für $d_n := \ell_{2n}(D)$

$$d_{n+1} = d_0 \cdot d_n + d_1 \cdot d_{n-1} + d_2 \cdot d_{n-2} + \cdots + d_n \cdot d_0$$

Theoretische Informatik 3 WS 2006/07
Beispiele
Zwei formale Sprachen

Beispiel: kontextfreie formale Sprache

- Für die durch

$$D \to aDbD \mid \lambda$$

erzeugte kontextfreie Sprache der korrekten Klammerungen (äquivalent: Binärbäume) gilt $\ell_{2n+1}(D) = 0$ für $n \geq 0$ und für $d_n := \ell_{2n}(D)$

$$d_{n+1} = d_0 \cdot d_n + d_1 \cdot d_{n-1} + d_2 \cdot d_{n-2} + \cdots + d_n \cdot d_0$$

Wir werden später sehen:

$$d_n = \ell_{2n}(D) = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi n^3}}$$

Polynome

Konventionen:

- Polynom $\simeq$ Folge der Koeffizienten

$$a(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_m X^m \quad \leftrightarrow \quad (a_0, a_1, a_2, \ldots, a_m)$$

Theoretische Informatik 3 WS 2006/07
Beispiele
Multiplikation von Polynomen und ganzen Zahlen

Polynome

Konventionen:

- Polynom $\simeq$ Folge der Koeffizienten

$$a(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_m X^m \quad \leftrightarrow \quad (a_0, a_1, a_2, \ldots, a_m)$$

- "Grösse" eines Polynoms = Anzahl der Koeffizienten (i.w. Grad)

Polynome

Konventionen:

- Polynom $\simeq$ Folge der Koeffizienten

$$a(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_m X^m \quad \leftrightarrow \quad (a_0, a_1, a_2, \ldots, a_m)$$

- "Grösse" eines Polynoms = Anzahl der Koeffizienten (i.w. Grad)
- Aufwand für Polynomoperationen wird gemessen in Anzahl der Operationen im Koeffizientenbereich (oft auch nur: Multiplikationen im Koeffizientenbereich)

Multiplikation

$$\begin{aligned} a(X) &= a_0 + a_1 X + a_2 X^2 + \cdots + a_m X^m \leftrightarrow & (a_0, a_1, a_2, \ldots, a_m) \\ b(X) &= b_0 + b_1 X + b_2 X^2 + \cdots + b_n X^n \leftrightarrow & (b_0, b_1, b_2, \ldots, b_n) \\ c(X) &= a(X) \cdot b(X) \\ &= c_0 + c_1 X + \cdots + c_{m+n} X^{m+n} \leftrightarrow & (c_0, c_1, c_2, \ldots, c_{m+n}) \end{aligned}$$

mit

$$c_k = \sum_{i+j=k} a_i \cdot b_j \quad (0 \le i \le m, 0 \le j \le n, 0 \le k \le m+n)$$

Berechnung mittels dieser Formel erfordert $(m+1)(n+1)$ Multiplikationen im Koeffizientenbereich $\Rightarrow$ Aufwand wächst (für $m = n$) *quadratisch* mit der Grösse der Instanzen

Komplexität der Multiplikation

- "klassisch": $\mathcal{O}(n^2)$

Komplexität der Multiplikation

- "klassisch": $\mathcal{O}(n^2)$
- das rekursive (divide-and-conquer) Verfahren von KARATSUBA leistet dies mit einem Aufwand $M_{kara}(n)$, für den gilt

$$M_{kara}(2n) = 3 \cdot M_{kara}(n) + \Theta(n)$$

und das ergibt $M_{kara}(n) \in \mathcal{O}(n^{\log_2 3})$, wobei $\log_2 3 \approx 1.585$.

Bemerkung: analoge Aussagen gelten für die Multiplikation von ganzen Zahlen

Komplexität der Multiplikation

- "klassisch": $\mathcal{O}(n^2)$
- das rekursive (divide-and-conquer) Verfahren von KARATSUBA leistet dies mit einem Aufwand $M_{kara}(n)$, für den gilt

$$M_{kara}(2n) = 3 \cdot M_{kara}(n) + \Theta(n)$$

und das ergibt $M_{kara}(n) \in \mathcal{O}(n^{\log_2 3})$, wobei $\log_2 3 \approx 1.585$.

- Man kann Polynome auch ganz anders multiplizieren: *Evaluation* und *Interpolation*

Bemerkung: analoge Aussagen gelten für die Multiplikation von ganzen Zahlen

Theoretische Informatik 3 WS 2006/07
Beispiele
Multiplikation von Polynomen und ganzen Zahlen

Komplexität der Multiplikation

- "klassisch": $\mathcal{O}(n^2)$
- das rekursive (divide-and-conquer) Verfahren von KARATSUBA leistet dies mit einem Aufwand $M_{kara}(n)$, für den gilt

$$M_{kara}(2n) = 3 \cdot M_{kara}(n) + \Theta(n)$$

und das ergibt $M_{kara}(n) \in \mathcal{O}(n^{\log_2 3})$, wobei $\log_2 3 \approx 1.585$.

- Man kann Polynome auch ganz anders multiplizieren: *Evaluation* und *Interpolation*
- mittels schneller FOURIER-Transformation (FFT=spezielles *rekursives* Evaluations-Interpolationsschema) kann man einen Aufwand $\mathcal{O}(n \log n)$ erreichen

Bemerkung: analoge Aussagen gelten für die Multiplikation von ganzen Zahlen

one-way?

- *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?

Theoretische Informatik 3 WS 2006/07
Beispiele
one-way Funktionen

one-way?

- *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?
- *Exponentiation* ist effizient machbar, aber gilt das auch für das *Logarithmieren*?

one-way?

- *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?
- *Exponentiation* ist effizient machbar, aber gilt das auch für das *Logarithmieren*?
- Antworten auf diese Fragen wären essentiell für die Sicherheit gängiger kryptographischer Verfahren!

Theoretische Informatik 3 WS 2006/07
Beispiele
one-way Funktionen

one-way?

- *Multiplizieren* ist effizient machbar, aber gilt das auch für das *Faktorisieren*?
- *Exponentiation* ist effizient machbar, aber gilt das auch für das *Logarithmieren*?
- Antworten auf diese Fragen wären essentiell für die Sicherheit gängiger kryptographischer Verfahren!
- Faktorisieren und Logarithmieren sind effizient durchführbar mit einem Computer, den es noch nicht gibt: *Quantencomputing*.

Sortieren

- Allgemein: $(A, \leq)$ totalgeordnete Menge
  Sortieren = Umordnen von Elementen einer A-Liste (oder eines A-Feldes) auf- oder absteigend bezgl. $\leq$

Theoretische Informatik 3 WS 2006/07
Beispiele
Sortierkomplexität

Sortieren

- Allgemein: $(A, \leq)$ totalgeordnete Menge
  Sortieren = Umordnen von Elementen einer A-Liste (oder eines A-Feldes) auf- oder absteigend bezgl. $\leq$
- Instanzengrösse = Listenlänge

Sortieren

- Allgemein: $(A, \leq)$ totalgeordnete Menge
  Sortieren = Umordnen von Elementen einer A-Liste (oder eines A-Feldes) auf- oder absteigend bezgl. $\leq$
- Instanzengrösse = Listenlänge
- Aufwand = Anzahl der paarweisen Vergleiche von Listenelementen

Theoretische Informatik 3 WS 2006/07
Beispiele
Sortierkomplexität

Sortieren

- Allgemein: $(A, \leq)$ totalgeordnete Menge
  Sortieren = Umordnen von Elementen einer A-Liste (oder eines A-Feldes) auf- oder absteigend bezgl. $\leq$
- Instanzengrösse = Listenlänge
- Aufwand = Anzahl der paarweisen Vergleiche von Listenelementen
- Vereinfachung (*Permutationsmodell*): Listenelemente sind paarweise verschieden, es kommt nur auf die relativen Ordnungsbeziehungen zwischen den Listenelementen an, nicht auf absolute Werte
  $\rightarrow$ Instanzen der Grösse n = Permutationen von $\{1, 2, \ldots, n\}$

Theoretische Informatik 3 WS 2006/07
Beispiele
Sortierkomplexität

Sortieren

- Allgemein: $(A, \leq)$ totalgeordnete Menge
  Sortieren = Umordnen von Elementen einer A-Liste (oder eines A-Feldes) auf- oder absteigend bezgl. $\leq$
- Instanzengrösse = Listenlänge
- Aufwand = Anzahl der paarweisen Vergleiche von Listenelementen
- Vereinfachung (*Permutationsmodell*): Listenelemente sind paarweise verschieden, es kommt nur auf die relativen Ordnungsbeziehungen zwischen den Listenelementen an, nicht auf absolute Werte
  $\rightarrow$ Instanzen der Grösse n = Permutationen von $\{1, 2, \ldots, n\}$
- NB: es gibt $n! \sim (n/e)^n \cdot \sqrt{2\pi n}$ Instanzen der Grösse n

Theoretische Informatik 3 WS 2006/07
Beispiele
Sortierkomplexität

Sortieralgorithmen

- Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen ($\rightarrow$ Knuth, TAOCP 3).

Sortieralgorithmen

- Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen ($\rightarrow$ KNUTH, TAOCP 3).
- Bei einigen der "simplen" Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.

Sortieralgorithmen

- Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen ($\rightarrow$ KNUTH, TAOCP 3).
- Bei einigen der "simplen" Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.
- Bei "guten" Algorithmen (HEAPSORT, MERGESORT) wächst der Aufand im worst-case (und auch im average-case) wie $\mathcal{O}(n \log n)$.

Sortieralgorithmen

- Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen ($\rightarrow$ KNUTH, TAOCP 3).
- Bei einigen der "simplen" Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.
- Bei "guten" Algorithmen (HEAPSORT, MERGESORT) wächst der Aufand im worst-case (und auch im average-case) wie $\mathcal{O}(n \log n)$.
- Bei QUICKSORT wächst der Aufwand im worst-case quadratisch, im average-case aber wie $\mathcal{O}(n \log n)$.

Sortieralgorithmen

- Es gibt viele verschiedene (experimentell und theoretisch gut untersuchte) Sortieralgorithmen ($\rightarrow$ KNUTH, TAOCP 3).
- Bei einigen der "simplen" Algorithmen (INSERTIONSORT, SELECTIONSORT, BUBBLESORT) wächst der Aufwand im worst- und im average-case quadratisch in der Listenlänge $\mathcal{O}(n^2)$.
- Bei "guten" Algorithmen (HEAPSORT, MERGESORT) wächst der Aufand im worst-case (und auch im average-case) wie $\mathcal{O}(n \log n)$.
- Bei QUICKSORT wächst der Aufwand im worst-case quadratisch, im average-case aber wie $\mathcal{O}(n \log n)$.
- Bei SHELLSORT kann man einen Aufwand wie $\mathcal{O}(n^{1+\epsilon})$ (für beliebig kleines $\epsilon > 0$) erreichen.

Untere Schranke für die Sortierkomplexität

- Jeder auf paarweisen Vergleichen basierende Sortieralgorithmus benötigt auf Listen der Länge n im worst-case und im average-case mindestens

$$\log n! \sim n \log n - \Theta(n)$$

Vergleichsoperationen (*informationstheoretische Schranke*)

Theoretische Informatik 3 WS 2006/07
Beispiele
Sortierkomplexität

Untere Schranke für die Sortierkomplexität

- Jeder auf paarweisen Vergleichen basierende Sortieralgorithmus benötigt auf Listen der Länge n im worst-case und im average-case mindestens

$$\log n! \sim n \log n - \Theta(n)$$

Vergleichsoperationen (*informationstheoretische Schranke*)

- In die average-case-Aussage geht der *Entropie*begriff der Informationstheorie entscheidend ein.

Das Problem k-SAT

- $X = \{x_1, x_2, \ldots, x_n\}$ Menge von a.l. Variablen,
  $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$ negierte Variablen
  $L = X \cup \overline{X}$: Menge der Literale

Theoretische Informatik 3 WS 2006/07
Beispiele
Erfüllbarkeit: ein NP-vollständiges Problem

Das Problem k-SAT

- $X = \{x_1, x_2, \ldots, x_n\}$ Menge von a.l. Variablen,
  $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$ negierte Variablen
  $L = X \cup \overline{X}$: Menge der Literale
- Klausel: Disjunktion von Literalen

$$K = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_s$$

Das Problem k-SAT

- $X = \{x_1, x_2, \ldots, x_n\}$ Menge von a.l. Variablen,
  $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$ negierte Variablen
  $L = X \cup \overline{X}$: Menge der Literale
- Klausel: Disjunktion von Literalen

$$K = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_s$$

- AL-Formel in KNF: Konjunktion von Literalen

$$F = K_1 \wedge K_2 \wedge \cdots \wedge K_t$$

Theoretische Informatik 3 WS 2006/07
Beispiele
Erfüllbarkeit: ein NP-vollständiges Problem

Das Problem k-SAT

- $X = \{x_1, x_2, \ldots, x_n\}$ Menge von a.l. Variablen,
  $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$ negierte Variablen
  $L = X \cup \overline{X}$: Menge der Literale
- Klausel: Disjunktion von Literalen

$$K = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_s$$

- AL-Formel in KNF: Konjunktion von Literalen

$$F = K_1 \wedge K_2 \wedge \cdots \wedge K_t$$

- k-SAT Formel: KNF-Formel mit $s \leq k$ Literalen pro Klausel

Theoretische Informatik 3 WS 2006/07
Beispiele
Erfüllbarkeit: ein NP-vollständiges Problem

Das Problem k-SAT

- $X = \{x_1, x_2, \ldots, x_n\}$ Menge von a.l. Variablen,
  $\overline{X} = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$ negierte Variablen
  $L = X \cup \overline{X}$: Menge der Literale
- Klausel: Disjunktion von Literalen

$$K = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_s$$

- AL-Formel in KNF: Konjunktion von Literalen

$$F = K_1 \wedge K_2 \wedge \cdots \wedge K_t$$

- k-SAT Formel: KNF-Formel mit $s \leq k$ Literalen pro Klausel
- Entscheidungsproblem: Erfüllbarkeit von KNF-Formeln testen!

Theoretische Informatik 3 WS 2006/07
Beispiele
Erfüllbarkeit: ein NP-vollständiges Problem

Algorithmen für k-SAT

- Brute-force Methode:
  alle 2^n Bewertungen der n Variablen testen
  $\rightarrow$ exponentielle Komplexität

Algorithmen für k-SAT

- Brute-force Methode:
  alle 2^n Bewertungen der n Variablen testen
  $\longrightarrow$ exponentielle Komplexität
- Bis heute kein effizientes Verfahren bekannt:
  zwar ist 2-SAT effizient entscheidbar, aber bereits 3-SAT ist NP-vollständig!

Theoretische Informatik 3 WS 2006/07
Beispiele
Erfüllbarkeit: ein NP-vollständiges Problem

Algorithmen für k-SAT

- Brute-force Methode:
  alle 2^n Bewertungen der n Variablen testen
  $\longrightarrow$ exponentielle Komplexität
- Bis heute kein effizientes Verfahren bekannt:
  zwar ist 2-SAT effizient entscheidbar, aber bereits 3-SAT ist NP-vollständig!
- Es gibt wesentlich bessere (aber immer noch exponentielle) Verfahren als die brute-force Methode!

traditionelle Multiplikation, rekursiv betrachtet

$$
\begin{aligned}
981 \times 1234 &= (9 \times 10^2 + 81) \times (12 \times 10^2 + 34) \\
&= \underline{9 \times 12} \times 10^4 \\
&\quad + (\underline{9 \times 34} + \underline{81 \times 12}) \times 10^2 \\
&\quad + \underline{81 \times 34} \times 10^0 \\
&= 108 \times 10^4 + 1278 \times 10^2 + 2754 \\
&= 1080000 + 127800 + 2754 \\
&= 1210554
\end{aligned}
$$

Karatsubas Multiplikationsidee (1962)

$$
\begin{aligned}
981 \times 1234 &= (9 \times 10^2 + 81) \times (12 \times 10^2 + 34) \\
&= (9 \times 12) \times 10^4 \\
&\quad + ((9 + 81) \times (12 + 34) - 9 \times 12 - 81 \times 34) \times 10^2 \\
&\quad + (81 \times 34) \times 10^0 \\
&= \underline{9 \times 12} \times 10^4 \\
&\quad + (\underline{90 \times 46} - 9 \times 12 - 81 \times 34) \times 10^2 \\
&\quad + \underline{81 \times 34} \times 10^0 \\
&= 108 \times 10^4 + 1278 \times 10^2 + 2754 \\
&= 1080000 + 127800 + 2754 \\
&= 1210554
\end{aligned}
$$

Die "Schulmethode" der Multiplikation am Beispiel 981×1234

9×1234	1	1	1	0	6		
8×1234			9	8	7	2	
1×1234				1	2	3	4
981×1234	1	2	1	0	5	5	4

Komplexität der Schulmethode

Die Multiplikation von zwei n-stelligen (dezimal oder binäre oder andere Basis) Zahlen erfordert $n \times n = n^2$ Multiplikationen von Ziffern (zuzüglich ca. n^2 Additionen von Ziffern)

Die sog. "Russische Bauernmultiplikation"

981	×	1234	=	?
981		1234		1234
490		2468		
245		4936		4936
122		9872		
61		19744		19744
30		39488		
15		78976		78976
7		157952		157952
3		315904		315904
1		631808		631808
				1210554

Vergleich der Komplexität von verschiedenen Multiplikationsalgorithmen
$n =$ Anzahl der Ziffern

"Schulmethode"	$c_1 \cdot n^2$
Karatsuba (1962)	$c_2 \cdot n^{\log_2 3}$
Schönhage/Strassen (1971)	$c_3 \cdot n \cdot \log n \cdot \log\log n$

n	n^2	$n^{\log_2 3}$	$n \log n \log\log n$
10^1	10^2	38.46×10^0	19.20×10^0
10^2	10^4	14.79×10^2	70.33×10^1
10^3	10^6	56.83×10^3	13.35×10^3
10^4	10^8	21.87×10^5	20.44×10^4
10^5	10^{10}	84.10×10^6	28.13×10^5
10^6	10^{12}	32.34×10^8	36.27×10^6
10^7	10^{14}	12.43×10^{10}	44.80×10^7
10^8	10^{16}	47.83×10^{11}	53.66×10^8
10^9	10^{18}	18.39×10^{13}	62.81×10^9
10^{10}	10^{20}	70.73×10^{14}	72.22×10^{10}

Schema für die Multiplikation von zwei $2n$-stelligen Zahlen

$$(a \times 10^n + b) \times (c \times 10^n + d) =$$
$$\underline{a \times c} \times 10^{2n} + (\underline{a \times d} + \underline{b \times c}) \times 10^n + \underline{b \times d} =$$
$$\underline{a \times c} \times 10^{2n} + \Big(\underline{(a+b) \times (c+d)} - a \times c - b \times d\Big) \times 10^n + \underline{b \times d}$$

wobei a, b, c, d n-stellige Zahlen sind

Beobachtung:

zwei $2n$-stellige Zahlen können multipliziert werden, indem man 3 (statt 4 (!!)) Multiplikationen von n-stelligen Zahlen ausführt

Folgerung (mittels Iteration dieses divide-and-conquer-Schemas):

zwei 2^n-stellige Zahlen können mittels
3^n (statt 4^n (!!)) Multiplikationen von Ziffern miteinander multipliziert werden.

Die schnellste bislange bekannte Multiplikationemethode von

Schönhage/Strassen (1971)

beruht auf der

"schnellen Fourier-Transformation" (FFT)

und benötigt eine Anzahl von Ziffernoperationen proportional zu

$$n \log n \log\log n$$

für die Multiplikation zweier n-stelligen Zahlen

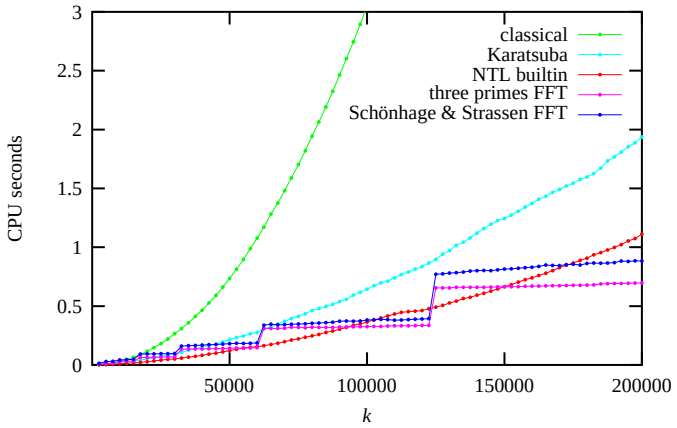


FIGURE 9.8: Multiplication of k-bit integers in NTL.

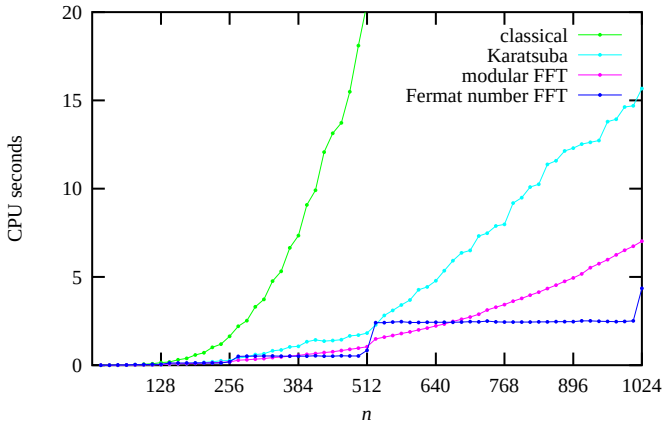


FIGURE 9.9: Multiplication of polynomials of degree $n-1$ with n bit integer coefficients in NTL.

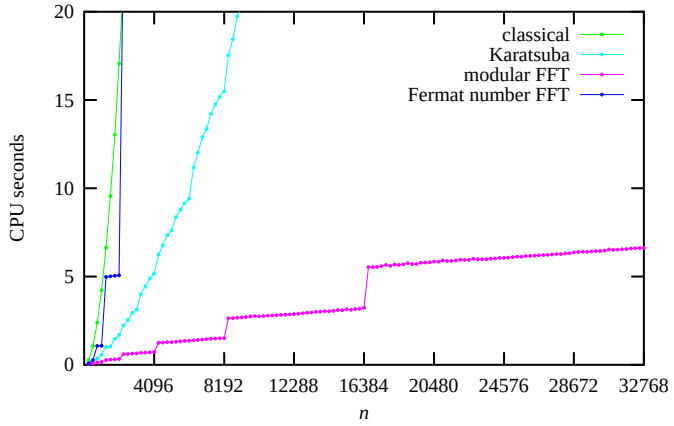


FIGURE 9.10: Multiplication of polynomials of degree $n-1$ with 64 bit integer coefficients in NTL.

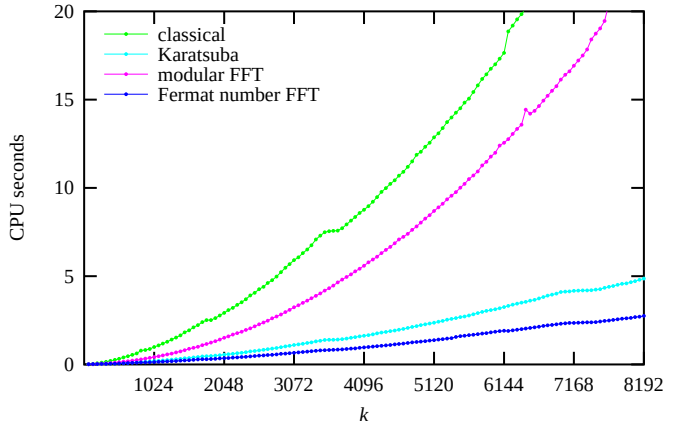


FIGURE 9.11: Multiplication of polynomials of degree 63 with k bit integer coefficients in NTL.

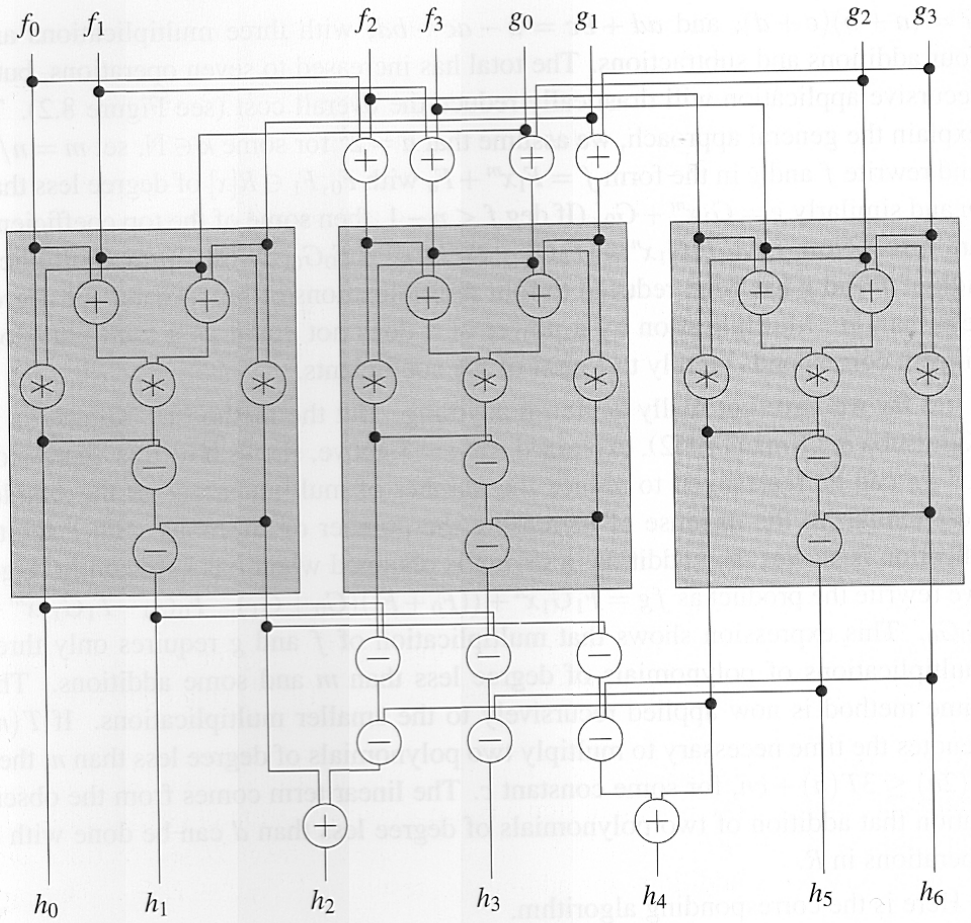


FIGURE 8.1: An arithmetic circuit illustrating Karatsuba's algorithm for $n = 4$. The shaded boxes are Karatsuba circuits for $n = 2$. A subtraction node computes the difference of its left input minus its right input.

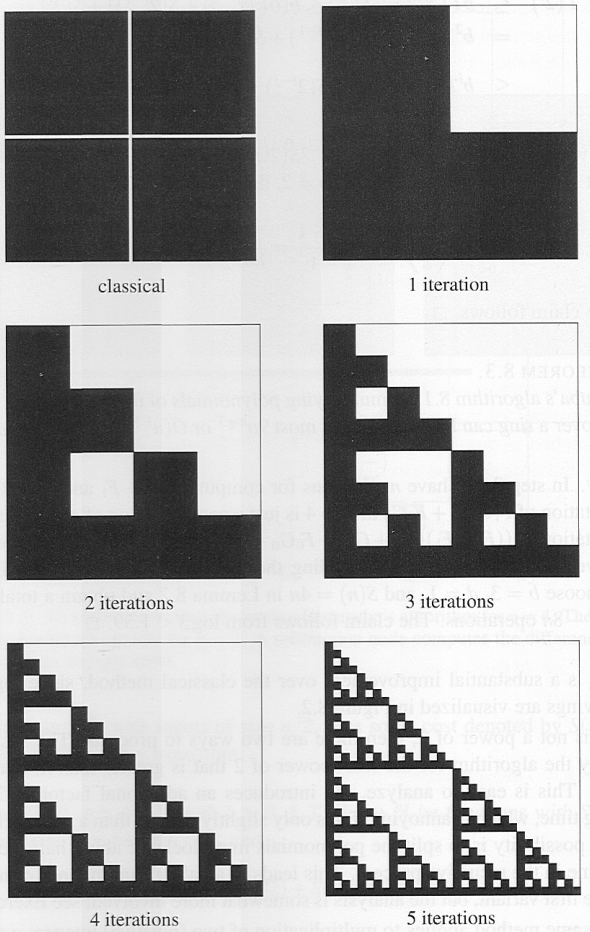


FIGURE 8.2: Cost (= black area) of Karatsuba's algorithm for increasing recursion depths. The image approaches a fractal of dimension $\log 3 \approx 1.59$.

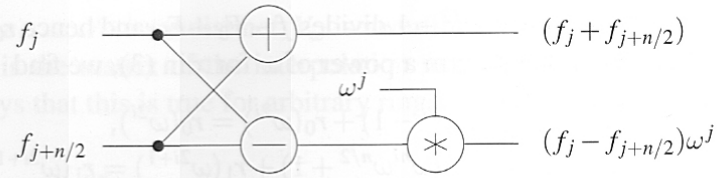


FIGURE 8.4: A butterfly operation. Flow of control is from left to right, and a subtraction node computes the difference of its upper input minus its lower input.

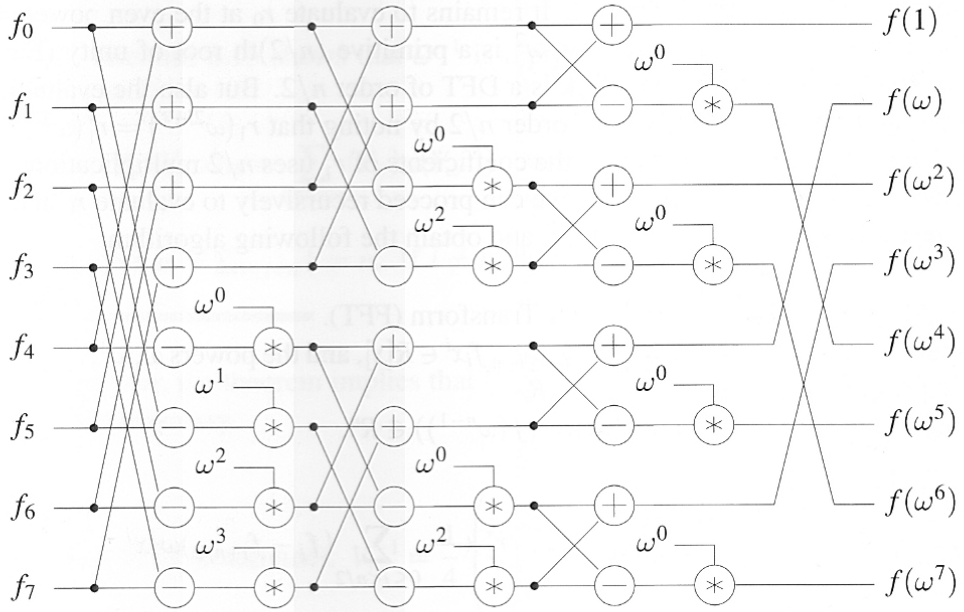


FIGURE 8.5: An arithmetic circuit computing the FFT for $n = 8$

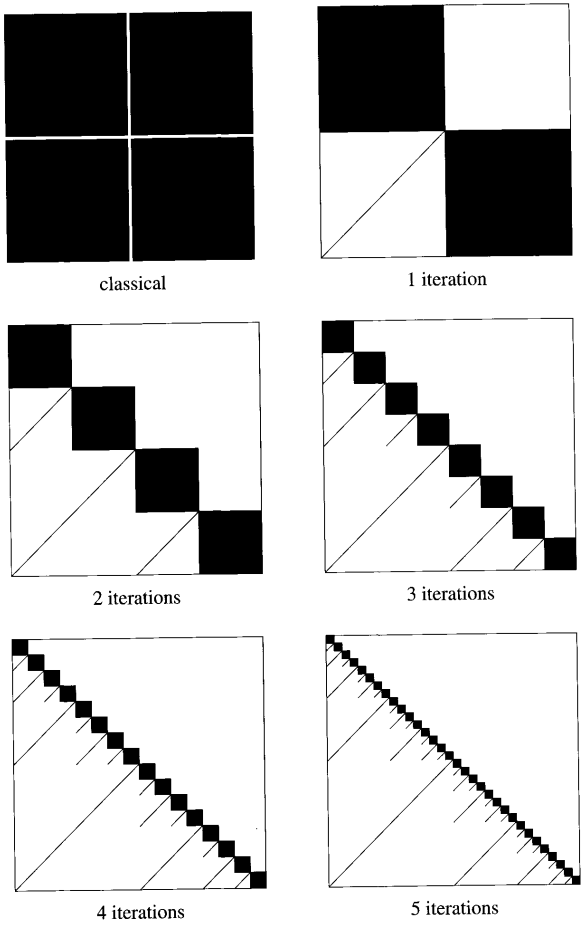


FIGURE 8.6: Cost of the FFT for increasing recursion depths. The black area is proportional to the total work.

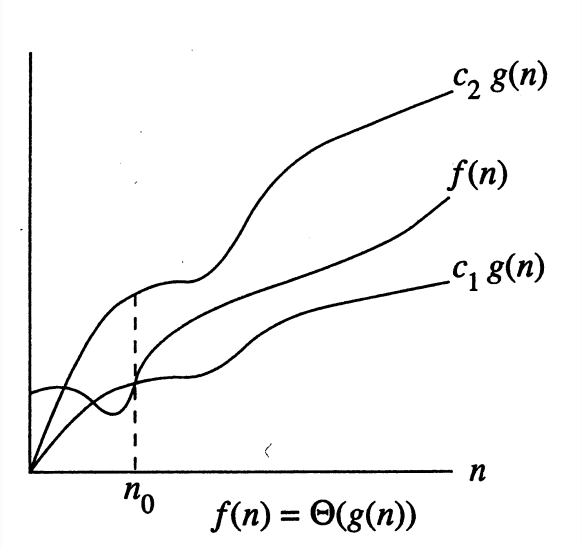
Notation für das asymptotische Verhalten von Funktionen

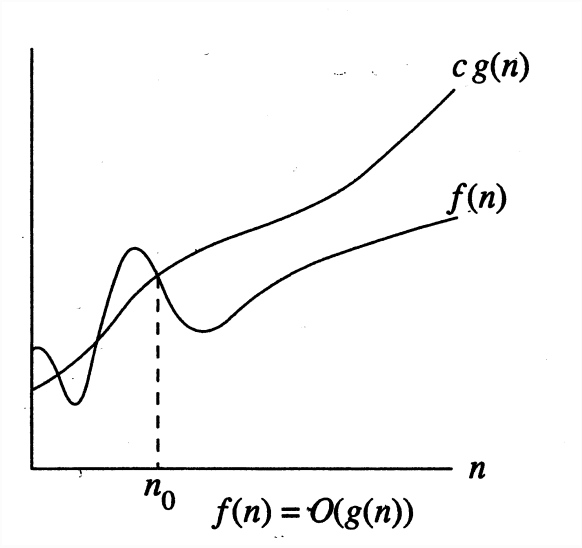
Vorbemerkungen:

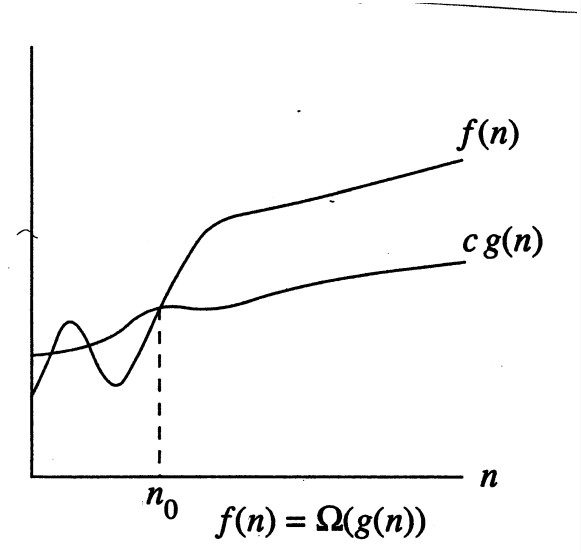
1. Aussagen über die Komplexität von Algorithmen und von Problemen sollen (in der Regel) unabhängig von speziellen Maschinenmodellen und speziellen Eigenschaften eine Implementierung, ebenso von technologischen Details
2. Bei der Untersuchung von Komplexitätsfunktionen interessiert nicht so sehr der exakte Werteverlauf einer Funktion $f : \mathbb{N} \to \mathbb{R}_+$, sondern deren "Tendenz", d.h. das Wachstumsverhalten (*asymptotisches Verhalten*) für wachsendes Argument

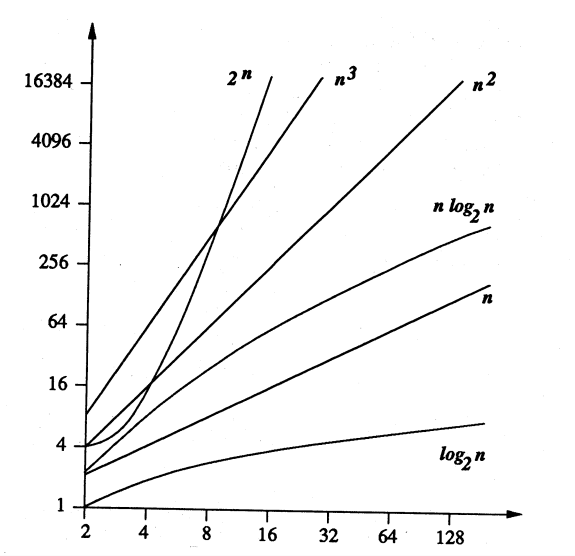
Edmund Georg Hermann Landau (1877–1938)
Professor der Mathematik in Göttingen (1909–1933)
Wichtige Arbeiten zu Zahlentheorie und Analysis ("Analytische Zahlentheorie")

www.math.uni-goettingen.de/Personen/Bedeutende_Mathematiker/landau.html









Landausche Symbole für asymptotisches Verhalten von Funktionen

$$O(f) = \{ g : \mathbb{N} \to \mathbb{R}_+ ; \exists c \in \mathbb{R}_{>0}\, \exists n_0 \in \mathbb{N}\, \forall n \in \mathbb{N}_{\geq n_0} : g(n) \leq c \cdot f(n) \}$$

$g(n) \in O(f(n))$: " $f(n)$ ist *asymptotische obere Schranke* für $g(n)$ "

$$\Omega(f) = \{ g : \mathbb{N} \to \mathbb{R}_+ ; \exists c \in \mathbb{R}_{>0}\, \exists n_0 \in \mathbb{N}\, \forall n \in \mathbb{N}_{\geq n_0} : g(n) \geq c \cdot f(n) \}$$

$$\Omega_\infty(f) = \{ g : \mathbb{N} \to \mathbb{R}_+ ; \exists c \in \mathbb{R}_{>0}\, \forall m \in \mathbb{N}\, \exists n \in \mathbb{N}_{>m} : g(n) \geq c \cdot f(n) \}$$

$g(n) \in \Omega(f(n))$: " $f(n)$ ist *asymptotische untere Schranke* für $g(n)$ "

$$\Theta(f) = \{ g : \mathbb{N} \to \mathbb{R}_+ ; g \in O(f) \wedge g \in \Omega(f) \}$$

$g(n) \in \Theta(f(n))$: " $f(n)$ hat *gleiche Wachstumsordnung* wie $g(n)$ "

$$o(f) = \left\{ g : \mathbb{N} \to \mathbb{R}_+ ; \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0 \right\}$$

$g(n) \in o(f(n))$: " $g(n)$ hat *kleinere Wachstumsordnung* als $f(n)$ "

$$\omega(f) = \left\{ g : \mathbb{N} \to \mathbb{R}_+ ; \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \right\}$$

$g(n) \in \omega(f(n))$: " $g(n)$ hat *grössere Wachstumsordnung* als $f(n)$ "

$f(n) \sim g(n)$: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

"$f(n)$ und $g(n)$ sind *asymptotisch äquivalent* "

Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$

$$f \in O(g) \quad \Leftrightarrow \quad |f| \in O(|g|)$$

wobei $|f| : \mathbb{N} \to \mathbb{R}_+ : x \mapsto |f(x)|$

Ebenso für die anderen Landauschen Symbole.

Rechenregeln

1. $\forall k, \ell \in \mathbb{N} \ : \ k > \ell \Rightarrow n^\ell \in o(n^k)$
2. $\forall k, \ell \in \mathbb{N} \ : \ k > \ell \Rightarrow n^k + n^\ell \in \Theta(n^k)$
3. für Polynome $p(n) = \sum_{i=0}^{k} p_i \, n^i$ mit $p_k > 0$, ℓ eine Konstante

$$\ell \ [\geq, \leq, =, >, <] \ k \quad \Rightarrow \quad p(n) \in \ [O, \Omega, \Theta, o, \omega] \ (n^\ell)$$

4. $\forall k \in \mathbb{N} \ : \ n^k \in o(2^n)$
5. Logarithmen zu verschiedenen Basen

$$\log_a n \in \Theta(\log_b n) \quad (a, b > 1)$$

6. $\forall k \in \mathbb{N} \forall \epsilon \in \mathbb{R}_{>0} \ : \ \log^k(n) \in o(n^\epsilon)$
7. $\forall n \in \mathbb{N} \ : \ 2^n \in o(2^{2n})$

8. $f \in O(g) \Leftrightarrow g \in \Omega(f)$ und $f \in o(g) \Leftrightarrow g \in \omega(f)$
9. $f \in \Omega(g) \Rightarrow f \in \Omega_\infty(g)$
10. $f \in \Omega_\infty(g) \not\Rightarrow f \in \Omega(g)$
11. Transitivität: $f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h) \Rightarrow f \in \mathcal{O}(h)$ für $\mathcal{O} \in \{O, \Omega, \Theta, o, \omega\}$
12. $f \in \Omega_\infty(g) \wedge g \in \Omega_\infty(h) \not\Rightarrow f \in O(h)$
13. $f_1 \in O(g) \wedge f_2 \in O(g) \Rightarrow f_1 + f_2 \in O(g)$
14. falls g nur endlich-viele Nullstellen hat:

$$f \in O(g) \ \Leftrightarrow \ \exists c \in \mathbb{R}_{>0} : \limsup_{n \to \infty} \frac{f(n)}{g(n)} \leq c$$

15. falls g nur endlich-viele Nullstellen hat:

$$f \in \omega(g) \ \Leftrightarrow \ \exists c \in \mathbb{R}_{>0} : \liminf_{n \to \infty} \frac{f(n)}{g(n)} \geq c$$

Das Wachstumsverhalten (asymptotisches Verhalten) einer Funktion $f : \mathbb{N} \to \mathbb{N}$ bezeichnet man als

a — *konstant*, falls $f(n) \in \Theta(1)$

b — *logarithmisch*, falls $f(n) \in \Theta(\log(n))$

c — *polylogarithmisch*, falls $f(n) \in O(\log^k(n))$ für ein $k \in \mathbb{N}$

d — *linear*, falls $f(n) \in \Theta(n)$

e — *quadratisch*, falls $f(n) \in \Theta(n^2)$

f — *polynomiell*, falls $f(n) \in O(n^k)$ für ein $k \in \mathbb{N}$

g — *superpolynomiell*, falls $f(n) \in \omega(n^k)$ für alle $k \in \mathbb{N}$

h — *subexponentiell*, falls $f(n) \in o(2^{cn})$ für alle $c \in \mathbb{R}_{>0}$

i — *exponentiell*, falls $f(n) \in O(2^{cn})$ für ein $c \in \mathbb{R}_{>0}$

Häufig in der Informatik:

Abschätzung des Wachstumsverhaltens von Funktionen $f(n)$, die gegeben sind durch

- Summen, wie z.B.

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \qquad S_k(n) = \sum_{i=1}^{n} i^k \qquad \log n! = \sum_{i=1}^{n} \log i$$

harmonische Zahlen | Potenzsummen | Fakultäten

- Rekursionsgleichungen, wie z.B.

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \qquad \text{mergesort}$$

$$T(n) = a \cdot T(n/b) + f(n) \qquad \text{divide-and-conquer}$$

$$T(n) = (n-1) + \frac{1}{n} \sum_{i=1}^{n} T(i-1) + T(n-i) \qquad \text{quicksort}$$

drei wichtige Beispiele

- harmonische Zahlen $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$

$$H_n \sim \ln n + \gamma + \frac{1}{2n} + O(1/n^2)$$

wobei $\gamma = 0.57721\ldots$ (EULERsche Konstante), also $H_n \in \Theta(\log n)$

- Potenzsummen $S_k(n) = \sum_{j=1}^{n} j^k$ für $k > -1$

$$\frac{n^{k+1}}{k+1} \leq S_k(n) \leq n^{k+1} \quad \text{also} \quad S_k(n) \in \Theta(n^{k+1})$$

- Fakultäten: STIRLINGs Formel

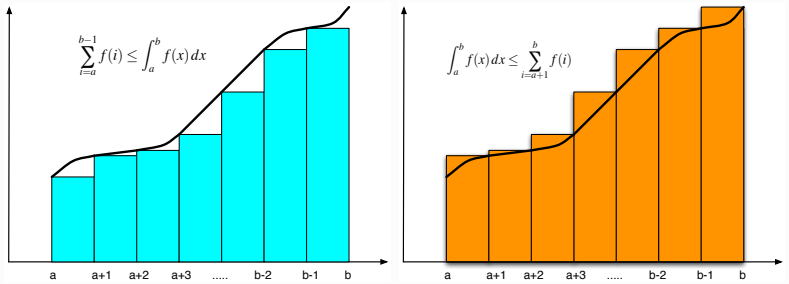
$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \frac{1}{12n} + \frac{1}{288n^2} + O(1/n^3)\right)$$

also $\log n! \in \Theta(n \cdot \log n)$

Wichtiges Hilfsmittel zum Abschätzen von Summationen: Integration

Ist $f[a,b] \to \mathbb{R}$ stetig und monoton wachsend, $a, b \in \mathbb{Z}$, so ist

$$\sum_{i=a}^{b-1} f(i) \;\leq\; \int_a^b f(x)dx \;\leq\; \sum_{i=a+1}^{b} f(i)$$



Anwendung auf die drei wichtigen Beispiele

- harmonische Zahlen H_n

mit $f : [1, n+1] \to \mathbb{R} : x \to 1/x$ ergibt sich

$$\ln(n+1) \;\leq\; H_n \leq\; 1 + \ln n$$

und somit $H_n \in \Theta(\log n)$

- Potenzsummen $S_k(n)$ mit $k > -1$
  mit $f : [0, n] \to \mathbb{R} : x \to x^k$ ergibt sich

$$S_k(n) \geq \frac{n^{k+1}}{k+1} \quad \text{also } S_k(n) \in \Omega(n^{k+1})$$

  Zusammen mit $S_k(n) \leq n \cdot n^k = n^{k+1}$ ergibt sich $S_k(n) \in \Theta(n^{k+1})$

- Fakultäten
  mit $f : [1, n] \to \mathbb{R} : x \to \ln x$ ergibt sich

$$\ln n! = \sum_{i=2}^{n} \ln i \geq \left[x \cdot \ln x - x\right]_1^n = n \cdot \ln n - n + 1 \in \Omega(n \cdot \ln n)$$

  Wegen $\ln n! \leq n \cdot \ln n$ ist $\ln n! \in O(n \cdot \ln n)$ und somit $\ln n! \in \Theta(n \cdot \log n)$

Zur Abschätzung der Fakultäten:

$$n \cdot \ln n - n \;\leq\; \ln n! \;<\; (n+1)\ln(n+1) - n \quad (n > 1)$$

ergibt

$$\left(\frac{n}{e}\right)^n \;<\; n! \;<\; e \cdot \left(\frac{n+1}{e}\right)^{n+1}$$

und wegen $e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^{n+1}$

$$\left(\frac{n+1}{n}\right)^{n+1} \sim e \;\text{ also }\; (n+1)^{n+1} \sim e \cdot n^{n+1}$$

Somit ist

$$\left(\frac{n}{e}\right)^n \;<\; n! \;<\; n \cdot \left(\frac{n}{e}\right)^n$$

STIRLING's Formel macht das noch präziser:

$$n! \;\sim\; \sqrt{2\pi \cdot n}\left(\frac{n}{e}\right)^n$$

oder

$$\log_2 n! \;=\; n \cdot \log_2 n - 1.44\ldots \cdot n + o(n)$$

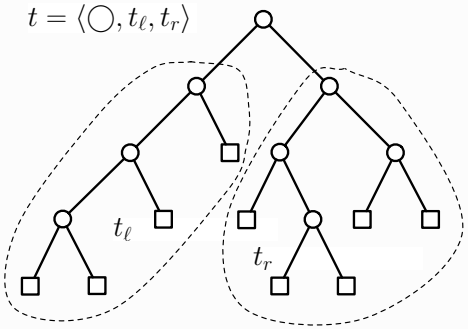
Eine Anwendung der Stirling-Formel:

Wieviele verschiedene Binärbäume mit n inneren Knoten gibt es?

BNF-Grammatik für Binärbäume:

$$B = \square + \langle \bigcirc, B, B \rangle$$

$\square$=äusserer Knoten, $\bigcirc$ = innerer Knoten



c_n = Anzahl der Binärbäumen mit n inneren Knoten

n	0	1	2	3	4	5	6	7	8	9	10	$\cdots$
c_n	1	1	2	5	14	42	132	428	1430	4862	16796	$\cdots$

$c_{100} = 896519947090131496687170070074100632420837521538745909320$

$c_{1000} =$

$204610552146802169264251998299782721717924564233905797584453809957217601019189186396496802615645375244901575056942859509731816363437015463738066688288637520335965324339092971743108044350900750477291297314225320935212694683984479674769763853760010063791881932656973098208302153805708771117628577790927586964863687485680595658005767317365566687003493944650164153396910927037406301799052584663611016897272893305532116292143271037140718751625839812072682464343153792956281748582435751481498598087587586903921577523657477775758899987954012641033870640665444651660246024318184109046864244732001962029120$

Die Zahlen c_n heissen CATALAN-Zahlen, zur Ehre von

Eugene Charles Catalan (1814-1894)
belgischer Mathematiker, Schüler von Liouville an der Ecole Polytechnique
wegen linksextremer politischer Aktivitäten keine akademische Karriere
Lehrer in Chalons-sur-Marne
Beiträge zur Zahlentheorie

der zeigte:

$$c_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)! \cdot n!}$$

Wie gross ist c_n?

Einsetzen der STIRLING-Approximation von $n!$ ergibt

$$c_n = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{(n+1)\cdot\sqrt{\pi n}} \in \Theta(\frac{4^n}{n^{3/2}})$$

Informationen zum quantitativen Verhalten von Algorithmen

1. Laufzeiten von Algorithmen mit verschiedenen Grössenordnungen der Zeitkomplexität (gemessen in Anzahl auszuführender Operationen bei input der Grösse n) und bei verschiedenen input-Grössen.
   Annahme: Rechner mit 10^6 Operationen. pro Sekunde.
2. Bearbeitbare Problemgrösse in gegebener Zeit, wenn der Algorithmus eine Komplexität in einer der angegebenen Grössenordnungen hat.
3. (a) Zeitbedarf in Abhängigkeit von der Komplexität, wenn man die zu bearbeitende Problemgrösse verzehnfacht.
   (b) Veränderung der bearbeitbaren Problemgrösse in Anhängigkeit von der Komplexität bei Verzehnfachung der zur Verfügung stehende Zeit (bzw. zehnfacher Taktrate bei gleicher Zeit).

Abkürzungen: μs = Mikrosekunde, ms = Millisekunde, s = Sekunde, h = Stunde, d = Tag, a = Jahr.
∞ steht für Werte $> 10^{100}$.

Laufzeiten von Algorithmen mit verschiedenen Grössenordnungen der Zeitkomplexität (gemessen in Anzahl auszuführender Operationen bei input der Grösse n) und bei verschiedenen input-Grössen.
Annahme: Rechner mit 10^6 Operationen pro Sekunde.

Komplexität Grösse	1	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
$n = 10^2$	$\simeq 1\mu$s	6.6μs	0.1ms	0.6ms	10ms	1s	$4 \times 10^{16}a$
$n = 10^3$	$\simeq 1\mu$s	9.9μs	1ms	9.9ms	1s	16.6m	∞
$n = 10^4$	$\simeq 1\mu$s	13.3μs	10ms	0.1s	100s	11.5d	∞
$n = 10^5$	$\simeq 1\mu$s	16.6μs	0.1s	1.6s	2.7h	31.7a	∞
$n = 10^6$	$\simeq 1\mu$s	19.9μs	1s	19.9s	11.5d	31.7×10^3a	∞

Bearbeitbare Problemgrösse in gegebener Zeit, wenn der Algorithmus eine Komplexität in einer der angegebenen Grössenordnungen hat.

Komplexität Zeit	1	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
1 Sekunde	∞	∞	10^6	63×10^3	10^3	100	19
1 Minute	∞	∞	6×10^7	28×10^5	77×10^2	390	25
1 Stunde	∞	∞	36×10^8	13×10^7	60×10^3	15×10^2	31
1 Tag	∞	∞	86×10^9	27×10^8	29×10^4	44×10^2	36

- Zeitbedarf in Abhängigkeit von der Komplexität, wenn man die zu bearbeitende Problemgrösse verzehnfacht.
- Veränderung der bearbeitbaren Problemgrösse in Anhängigkeit von der Komplexität bei Verzehnfachung der zur Verfügung stehende Zeit (bzw. zehnfacher Taktrate bei gleicher Zeit).

Komplexität	1	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
Zeit bei $10\times$Grösse	t	$t + 3.329$	$10 \times t$	$(10 + \epsilon) \times t$	$100 \times t$	$1000 \times t$	t^{10}
Grösse bei $10\times$ Zeit	∞	n^{10}	$10 \times n$	$(10 - \epsilon) \times n$	$3.16 \times n$	$2.15 \times n$	$n + 3.32$

1

Holzschnitt aus der frühen Enzyklopädie *MARGARITA PHILOSOPHICA* von Gregor Reisch (Strassburg, 1504).

Symbolisch dargestellt wird die vierhundert Jahre währende Rivalität zweier Rechentechniken:

- Der *Abacist* (rechts), personifiziert durch *Pythagoras*, propagiert das römischen Zahlsystem und die Benutzung des Rechenbretts (abacus) und der Rechensteine (calculi).
- Der *Algorist* (links), personifiziert durch *Boethius*, propagiert das Hindu-arabische Zahlsystem und die damit einhergehenden Rechentechniken, die seit dem 12. Jahrhundert durch Übersetzungen arabischer Lehrbücher (vor allem: *al Khwarizm*) in Europa bekannt wurden.
- Zentrale Figur ist die *Arithmetik*, eine der sieben freien Künste im Rahmen der mittelalterlichen Universitätsbildung.
- Zu Beginn des 16. Jh. hat sich die neue Rechentechnik durchgesetzt.

2

3

Abū Ja'far Muhammad ibn Mūsā *al Khwārizmī* (etwa 780 - etwa 850) muslimischer Mathematiker am "Haus der Weisheit" in Bagdad

- *Al kitab al-mukthasar fi hisab al-jabr w'al-muqabala* (etwa 830)
  [Das umfassende Buch vom Rechnen durch Ergänzung und Ausgleich]
  das erste Buch über Algebra: Lösen von linearen und quadratischen Gleichungen
  lateinische Übersetzungen im 12. Jh.
- *Algoritmi de numero Indorum* (etwa 820)
  arabisches Original nicht erhalten
  lateinische Übersetzungen im 12. Jh.

  Dixit Algoritmi ...
- weitere Bücher zu Astronomie (*Sindhind zij*), geografische Vermessung, astronomische Geräte, Kalender, ...

4

Al-Khwarizmi und das "indische" Zahlsystem

> When I consider what people generally want in calculating, I found that it always is a number. I also observed that every number is composed of units, and that any number may be divided into units. Moreover, I found that every number which may be expressed from one to ten, surpasses the preceding by one unit: afterwards the ten is doubled or tripled just as before the units were: thus arise twenty, thirty, etc. until a hundred: then the hundred is doubled and tripled in the same manner as the units and the tens, up to a thousand; ... so forth to the utmost limit of numeration.

(Beginn der *Algebra*, übersetzt und herausgegeben als *Mohmmed Ben Musa's Compendium on Calculating by Completion and Reduction* von Frederic ROSEN, London, 1831 — Nachdruck Frankfurt, 1997.)

Leonardo von Pisa (1170-1250, Fibonacci = filius bonaccii)

FIBONACCI publizierte nach seinen Studienreisen in Nordafrika 1202 das Buch *Liber abaci*, mit dem er die arabischen Zahlzeichen (bisweilen auch als *algorismen* bezeichnet), das Hindu-arabische Positionssystem und damit verbundene Rechentechniken in Europa bekanntmachte. Das Buch war sehr populär und wurde vielfach kopiert und imitiert. Rechentechnisch geht er bis zur Behandlung von linearen Gleichungssystemen.

Der zweite Teil von *Liber abaci* ist eine Sammlung von kaufmännischen Problemen, wie man Preise und Profite berechnet, Währungen umrechnet etc. Viele Probleme sind chinesischen Ursprungs.

Die berühmten FIBONACCI-Zahlen erscheinen im dritten Teil des *Liber abaci* im Rahmen folgender Problemstellung:

> Ein Mann setzt ein Kaninchenpaar in einen von einer Mauer abgeschlossenen Bereich. Wieviele Kaninchenpaare kann man aus diesem einen Paar innerhalb eines Jahres erzeugen, wenn jedes Paar innerhalb eines Monats ein neues Paar erzeugt, das vom zweiten Monat an fruchtbar wird?

Die so erzeugt Folge (1,) 1, 2, 3, 5, 8, 13, 21, 34, 55, ... tritt an vielen Stellen in den verschiedensten Bereichen von Natur, Wissenschaft und Kunst auf.

1. $k > \ell \geq 0 \Rightarrow n^\ell \in o(n^k)$

$$\lim_{n\to\infty} \frac{n^\ell}{n^k} = \lim_{n\to\infty} \frac{1}{n^{k-\ell}} = 0$$

2. $k > \ell \geq 0 \Rightarrow n^k + n^\ell \in \Theta(n^k)$

$$n^k \leq n^k + n^\ell \quad \Rightarrow \quad n^k + n^\ell \in \Omega(n^k)$$

$$n^k + n^\ell \leq 2 \cdot n^k \quad \Rightarrow \quad n^k + n^\ell \in O(n^k)$$

3. $p(n) = \sum_{i=0}^{k} p_i\, n^i$ mit $p_k \neq 0 \Rightarrow p(n) \in O(n^k)$

$$\begin{aligned}
|p(n)| &= \left| \sum_{i=0}^{k} p_i\, n^i \right| \\
&\leq \sum_{i=0}^{k} |p_i|\, n^i \\
&\leq \max_{0\leq i\leq k} \left(|p_i|\right) \cdot \sum_{0\leq i\leq k} n^i \\
&\leq \max_{0\leq i\leq k} \left(|p_i|\right) \cdot \frac{n^{k+1}-1}{n-1} \\
&\leq \max_{0\leq i\leq k} \left(|p_i|\right) \cdot 2 \cdot n^k \quad (\text{falls } n \geq 2) \\
\\
&\Rightarrow \quad p(n) \in O(n^k)
\end{aligned}$$

4. $p(n) = \sum_{i=0}^{k} p_i\, n^i$ mit $p_k > 0 \Rightarrow p(n) \in \Omega(n^k)$

$$\begin{aligned}
p(n) &= p_k\, n^k + \sum_{i=0}^{k-1} p_i\, n^i \\
&= p_k\, n^k \left(1 + \frac{1}{n^k} \sum_{i=0}^{k-1} \frac{p_i}{p_k}\, n^i\right) \\
&\geq p_k\, n^k \left(1 - \frac{1}{n^k} \sum_{i=0}^{k-1} \left|\frac{p_i}{p_k}\right|\, n^i\right) \\
&\geq p_k\, n^k \left(1 - \frac{1}{n^k} \cdot \max_{0\leq i<k} \left(\left|\frac{p_i}{p_k}\right|\right) \cdot 2 \cdot n^{k-1}\right) \\
&= p_k\, n^k \left(1 - 2 \cdot \frac{\max}{n}\right) \\
&\geq \frac{1}{2}\, p_k\, n^k \quad \text{für } n \geq 4 \cdot \max \\
\\
&\Rightarrow \quad p(n) \in \Omega(n^k)
\end{aligned}$$

5. $n^k = o(2^n)$ $(k \in \mathbb{N})$

$$\forall n, \ell \in \mathbb{N} \quad \ln n \leq \frac{n}{\ell} + \ell$$

$$\Downarrow$$

$$\lim_{n\to\infty} \Big(n - k \cdot \underbrace{\log n}_{\leq 2 \ln n}\Big) \geq \lim_{n\to\infty} \left(n - 2k\left(\frac{n}{4k} + 4k\right)\right) = \lim_{n\to\infty} \left(\frac{n}{2} - 8k^2\right) = \infty$$

$$\Downarrow$$

$$\lim_{n\to\infty} \frac{n^k}{2^n} = \lim_{n\to\infty} 2^{k\cdot\log n - n} = 0$$

6. $\log^k n = o(n^\epsilon)$ $(\epsilon > 0, k \in \mathbb{N})$

   Fall $k = 1$

$$\lim_{n\to\infty} \frac{\log n}{n^\epsilon} = \lim_{n\to\infty} \frac{1/(n \cdot \ln 2)}{\epsilon \cdot n^{\epsilon-1}} = \frac{1}{\ln 2 \cdot \epsilon} \lim \frac{1}{n^\epsilon} = 0$$

   Fall $k > 1$ (Induktion)

$$\lim_{n\to\infty} \frac{\log^k n}{n^\epsilon} = \frac{1}{\ln 2} \lim_{n\to\infty} \frac{k \cdot \log^{k-1} n \cdot \frac{1}{n}}{\epsilon \cdot n^{\epsilon-1}} = \frac{k}{\epsilon \cdot \ln 2} \cdot \lim_{n\to\infty} \frac{\log^{k-1} n}{n^\epsilon} = 0$$

7. $2^n \in o(2^{2n})$ (aber: $n \notin o(2n)$)

$$\lim_{n\to\infty} \frac{2^n}{2^{2n}} = \lim \frac{1}{2^n} = 0$$

8. O vs. Ω, o vs. ω

$$f \in O(g) \Leftrightarrow g \in \Omega(f)$$
$$f \in o(g) \Leftrightarrow g \in \omega(f)$$

9. Transitivität $(\mathcal{O} \in \{O, o, \Theta, \Omega, \omega\})$

$$f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h) \Rightarrow f \in \mathcal{O}(h)$$

10. Additivität $(\mathcal{O} \in \{O, o, \Theta, \Omega, \omega\})$

$$f \in \mathcal{O}(h) \wedge g \in \mathcal{O}(h) \Rightarrow f + g \in \mathcal{O}(h)$$

Landaus asymptotische Notation

$$O, \Omega, o, \omega, \Theta, \sim$$

- wird vorausgesetzt
- siehe Folien auf webseite
- oder einschlägige Literatur (z.B. Cormen, Leiserson, Rivest)

Geometrische Reihe

$$\sum_{k=0}^{n} \alpha^k = \begin{cases} \frac{1-\alpha^{n+1}}{1-\alpha} & \text{falls } \alpha \neq 1 \\ n+1 & \text{falls } \alpha = 1 \end{cases}$$

folgt aus der Polynomgleichung

$$(1 + X + X^2 + X^3 + \cdots + X^n)(1 - X) = 1 - X^{n+1}$$

Die geometrische Reihe

$$\sum_{k=0}^{\infty} \alpha^k = \begin{cases} \text{konvergiert gegen } \frac{1}{1-\alpha} & \text{für } |\alpha| < 1 \\ \text{divergiert für } |\alpha| \geq 1 \end{cases}$$

Anwendung: das Master-Theorem für divide-and-conquer-Rekursionen (einfachste Version)

Die Lösung der Rekursion

$$T(n) = a \cdot T(\frac{n}{b}) + c \cdot n, T(1) = d$$

mit $a, c, d \in \mathbb{R}_{>0}, b \in \mathbb{N}_{>0}$ verhält sich so:

$$T(n) \in \begin{cases} \Theta(n) & \text{falls } a < b \\ \Theta(n \log n) & \text{falls } a = b \\ \Theta(n^{\log_b a}) & \text{falls } a > b \end{cases}$$

Zum Beweis: man lässt sinvollerweise die Rekursion über Potenzen von b laufen. Mit $n = b^k$ und $t_k := T(b^k)$ erhält man

$$t_k = a \cdot t_{k-1} + c \cdot b^k \ (k > 0), t_0 = d$$

Dies ist eine inhomogene lineare Rekursion erster Ordnung für die $(t_k)_{k \geq 0}$.
Rückwärtsentwickeln der Rekursion (Induktion!) liefert

$$t_k = a^\ell \cdot t_{k-\ell} + c \cdot (a^{\ell-1} b^1 + a^{\ell-2} b^2 + \cdots + a^0 b^\ell) \ \ (0 \leq \ell \leq k)$$

und somit für $\ell = k$:

$$t_k = a^k \cdot t_0 + c \cdot b^k \cdot \left(\left(\frac{a}{b}\right)^{k-1} + \left(\frac{a}{b}\right)^{k-2} + \cdots + \left(\frac{a}{b}\right)^0 \right)$$

$$= \begin{cases} a^k \cdot d + c \cdot b^k \cdot \frac{1-\left(\frac{a}{b}\right)^k}{1-\frac{a}{b}} & \text{falls } a \neq b \\ b^k \cdot d + c \cdot b^k \cdot k & \text{falls } a = b \end{cases}$$

Damit ist

$$t_k \sim \begin{cases} \frac{c \cdot b}{b-a} \cdot b^k & \text{falls } a < b \\ c \cdot k \cdot b^k & \text{falls } a = b \\ (d + \frac{c \cdot b}{a-b}) \cdot a^k & \text{falls } a > b \end{cases}$$

Jetzt muss man das nur noch auf die $T(n)$ umformen und dabei $b^k = n$, $k = \log_b n$ und $a = b^{\log_b a}$ beachten.

Binomialformel, Binomialkoefizienten
Für $x \in \mathbb{C}$ (allgemeiner: kommutativer Ring) gilt

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

wobei $$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$
$$= \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!} \ \ (0 \leq k \leq n)$$

Beachte: $n \mapsto \binom{n}{k}$ ist ein Polynom k-ten Grades in n.
Vereinbarung: $\binom{n}{k} = 0$ falls $k < 0$ oder $k > n$.

Einige Formeln

$$\binom{n}{k} = \binom{n}{n-k} \qquad \binom{n+1}{k} = \binom{n}{k} \frac{n+1}{n-k+1}$$

$$\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k} \qquad \binom{n+1}{k+1} = \sum_{m=k}^{n} \binom{m}{k}$$

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \qquad \sum_{k=0}^{n} k \cdot \binom{n}{k} = n \cdot 2^{n-1}$$

Bedeutung

- PASCALsches Dreieck
  (Omar Khayyám (1050–1123), Yuang Hui (1261),
  Chu Shih-Chieh (1303), Blaise Pascal (1623))
- $\binom{n}{k}$ ist
  - die Anzahl der k-elementigen Teilmengen einer n-elementigen Menge
  - die Anzahl der Bitvektoren der Länge n mit HAMMING-Gewicht k
  - die Anzahl der diagonalen Gitterwege in einem $k \times (n-k)$-Gitter

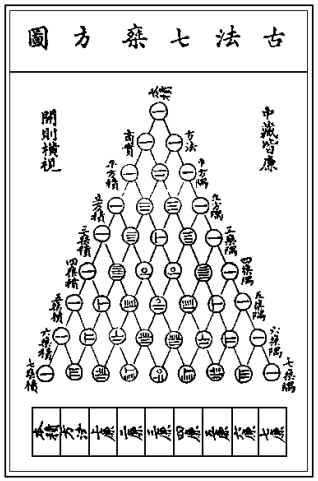


Abbildung: aus Ssu Yuan Yu (1303) von CHU SHIH-CHIEH

Binomialreihe (NEWTON)
Für $\alpha \in \mathbb{C}$ hat die Funktion

$$x \mapsto (1+x)^\alpha$$

die für $|x| < 1$ konvergierende Reihenentwicklung

$$(1+x)^\alpha = \sum_{n \geq 0} \binom{\alpha}{n} x^n$$

wobei

$$\binom{\alpha}{n} = \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-n+1)}{n!}$$

Die Binomialreihe ergibt sich als Taylorentwicklung in $x = 0$ wegen

$$\left(\frac{d}{dx}\right)^n (1+x)^\alpha = \alpha(\alpha-1)(\alpha-2)\cdots(\alpha-n+1)(1+x)^{\alpha-n} \quad (n \in \mathbb{N})$$

$$\frac{1}{n!}\left(\frac{d}{dx}\right)^n (1+x)^\alpha \,|_{x=0} = \binom{\alpha}{n} \quad (n \in \mathbb{N})$$

- Für $\alpha = -1$ hat man die die geometrische Reihe.
- Für $\alpha \in \mathbb{N}$ bricht die Reihe nach dem Term $\ldots + x^\alpha$ ab.
  Das ist die Situation der Binomialformel.
- Für $\alpha \in \mathbb{C} \setminus \mathbb{N}$ hat die Reihe unendlich viele Terme.

Für $\alpha \in \mathbb{R}$ möchte man das asymptotische Verhalten von Potenzsummen

$$S_\alpha(N) = \sum_{n=0}^{N} n^\alpha = 1^\alpha + 2^\alpha + 3^\alpha + \cdots + N^\alpha$$

kennen.

Beispiele:

$$\begin{aligned}
S_1(N) &= 1 + 2 + 3 + \cdots + N &&= \frac{N(N+1)}{2} && \in \Theta(N^2)\\
S_2(N) &= 1^2 + 2^2 + 3^2 + \cdots + N^2 &&= \frac{N(N+1)(2N+1)}{6} && \in \Theta(N^3)\\
S_3(N) &= 1^3 + 2^3 + 3^3 + \cdots + N^3 &&= S_1(N)^2 && \in \Theta(N^4)\\
S_{-1}(N) &= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N} &&= ? && \in \Theta(?)
\end{aligned}$$

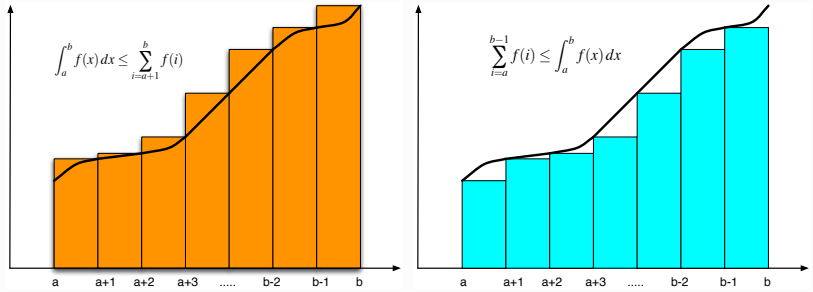


Abbildung: Abschätzung durch Ober- und Untersumme

Für die Funktion $x \mapsto x^\alpha$ ergibt das:

- für $\alpha > 0$:

$$S_\alpha(N-1) \le \int_1^N x^\alpha dx \le S_\alpha(N) - 1$$
$$1 + \int_1^N x^\alpha dx \le S_\alpha(N) \le \int_1^{N+1} x^\alpha dx$$

- für $\alpha < 0$:

$$S_\alpha(N) - 1 \le \int_1^N x^\alpha dx \le S_\alpha(N-1)$$
$$\int_1^{N+1} x^\alpha dx \le S_\alpha(N) \le 1 + \int_1^N x^\alpha dx$$

Beachte

$$\int_1^N x^\alpha dx = \begin{cases} \frac{1}{1+\alpha}(N^{\alpha+1}-1) & \alpha \neq -1 \\ \ln N & \alpha = -1 \end{cases}$$

Daher

$$S_\alpha(N) \begin{cases} \in \Theta(N^{\alpha+1}) & \text{für } \alpha > -1 \\ \in \Theta(\log N) & \text{für } \alpha = -1 \\ \text{konvergiert} & \text{für } \alpha < -1 \end{cases}$$

Beispiele:

$$S_{-2}(N) = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots \to \frac{\pi^2}{6}$$

$$S_{-4}(N) = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \cdots \to \frac{\pi^4}{90}$$

(→ BERNOULLI-Zahlen und -Polynome)

Hinweis: eingehendere Untersuchungen führen auf RIEMANNs *Zetafunktion*

$$\zeta(z) = \sum_{n=1}^{\infty} \frac{1}{n^z} = \prod_{p \text{ prim}} \frac{1}{1-p^{-z}}$$

→ Zahlentheorie, RIEMANNs Vermutung

Folgerung: Ist

$$a(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_m X^m \quad (a_m \neq 0)$$

ein Polynom m-ten Grades, so wächst die Funktion

$$N \mapsto \sum_{n=1}^{N} a(n)$$

wie ein Polynom $(m+1)$-ten Grades, d.h. $\sum_{n=1}^{N} a(n) \in \Theta(N^{m+1})$.

Genauer: es gibt ein Polynom vom Grad $m+1$

$$b(X) = b_0 + b_1 X + b_2 X^2 + \cdots + b_{m+1} X^{m+1} \quad (b_{m+1} \neq 0)$$

mit

$$b(X+1) - b(X) = a(X)$$

und $b(X)$ ist bis auf die *Summationskonstante* b_0 eindeutig bestimmt.
Man schreibt $\Delta b(X) = a(X)$ (Differenzenoperator). $b(X)$ ist die *diskrete Stammfunktion* von $a(X)$.

Der Fall $\alpha = -1$: harmonische Zahlen

$$H_N = S_{-1}(N) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}$$

Man kann zeigen

$$H_N = \gamma + \ln N + \frac{1}{2n} + \sum_{k \geq 2} (-1)^k \frac{1}{k} \int_0^1 \frac{\binom{x}{k}}{\binom{N}{k}} \, dx$$

mit $\gamma = 0.57721\ldots$ EULERsche Konstante.

N	1	2	3	4	5	6	7	8	9	10
H_N	1	$\frac{3}{2}$	$\frac{11}{6}$	$\frac{25}{12}$	$\frac{137}{60}$	$\frac{49}{20}$	$\frac{363}{140}$	$\frac{761}{280}$	$\frac{7129}{2520}$	$\frac{7381}{2520}$

Wie kann man das asymptotische Wachstum der Fakultätsfunktion

$$N \mapsto N! = 1 \cdot 2 \cdot 3 \cdots (N-1) \cdot N$$

beschreiben?

Wichtige Bedeutung:
$N!$ ist die Anzahl der Permutationen von N Elementen.

N	0	1	2	3	4	5	6	7	8	9	10
$N!$	1	1	2	6	24	120	720	5040	40320	362880	3628800

Integralabschätzung für $x \mapsto \ln x$ ergibt

$$\sum_{n=1}^{N-1} \ln n \leq \int_1^N \ln x \, dx \leq \sum_{n=2}^{N} \ln n$$

und somit

$$[x \ln x - x]_1^N \leq \ln N! \leq [x \ln x - x]_1^{N+1}$$

und das ergibt

$$N \ln N - N + 1 \leq \ln N! \leq (N+1)\ln(N+1) - (N+1) + 1$$

und somit

$$\ln N! \in \Theta(N \log N)$$

Zusammen mit

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{N}\right)^N$$

erhält man

$$\left(\frac{N}{e}\right)^N < N! < N \cdot \left(\frac{N}{e}\right)^N$$

Bessere Abschätzungen erhält man, indem man $\int \ln x \, dx$ mit Polygonzügen approximiert.

STIRLINGs Formel

$$\begin{aligned} n! &= \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \frac{1}{12n} + \frac{1}{288n^2} + \cdots\right) \\ &= \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \cdot e^{\alpha_n} \end{aligned}$$

wobei $1/(12n+1) < \alpha_n < 1/(12n)$.

Folgerung

$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

$$\log n! = n \log n - 1.41\ldots \cdot n + o(n)$$

Eingehendere Untersuchungen führen auf die *Gammafunktion* (EULER)

$$\Gamma(z) = \int_0^\infty t^{z-1} e^t dt \quad (\Re\, z > 0)$$

Dies ist eine in die komplexe Ebene fortgesetzte Fakultätsfunktion, denn es gilt

$$\Gamma(z+1) = z \cdot \Gamma(z)$$

also wegen $\Gamma(1) = 1$ insbesondere

$$\Gamma(n) = (n-1)! \quad (n \in \mathbb{N}_{>0})$$

Gammafunktion und Zetafunktion hängen eng zusammen.

Einfache Abschätzung für Binomialkoeffizienten:

$$\left(\frac{n}{k}\right)^k \overset{(*)}{\leq} \binom{n}{k} \overset{(**)}{\leq} \left(\frac{e\cdot n}{k}\right)^k$$

(*) wegen

$$\left(\frac{n}{k}\right)^k = \frac{n\cdot n\cdots n}{k\cdot k\cdots k} \leq \frac{n}{k}\cdot\frac{n-1}{k-1}\cdots\frac{n-k+1}{1} = \binom{n}{k}$$

(**) wegen

$$e^k = e^{\frac{k}{n}\cdot n} \geq (1+\frac{k}{n})^n = \sum_{j=0}^{n}\binom{n}{j}\left(\frac{k}{n}\right)^j \geq \binom{n}{k}\left(\frac{k}{n}\right)^k$$

Mit Hilfe der der Entropiefunktion

$$H(x) = -x\log x - (1-x)\log(1-x) \quad (0\leq x\leq 1)$$

erzielt SHANNON eine sehr genaue Abschätzung:

$$\frac{2^{n\cdot H(\lambda)}}{\sqrt{8n\lambda\mu}} \leq \binom{n}{\lambda n} \leq \frac{2^{n\cdot H(\lambda)}}{\sqrt{2\pi n\lambda\mu}}$$

wobei $0<\lambda<1$ und $\mu = 1-\lambda$.
Damit gilt insbesondere:

$$\lim_{n\to\infty}\frac{1}{n}\log\binom{n}{\lambda n} = H(\lambda)$$

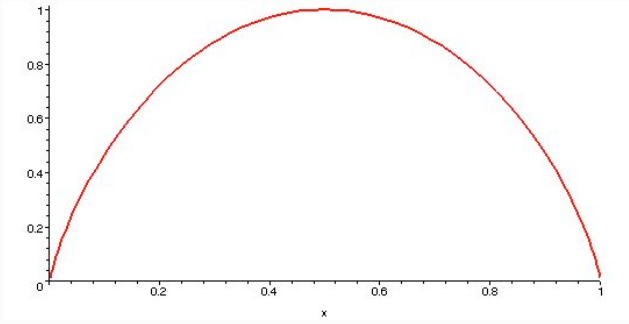


Abbildung: Graph der Entropiefunktion $H(x) = -x\log x-(1-x)\log(1-x)$

Beispiel:

$$\binom{2n}{n} \approx \frac{2^{2nH(1/2)}}{\sqrt{2\pi\frac{1}{2}(1-\frac{1}{2})2n}} = \frac{4^n}{\sqrt{\pi n}}$$

Anwendung: die Anzahl binärer Bäume mit n inneren Knoten ist

$$c_n = \frac{1}{n+1}\binom{2n}{n} \in \Theta\left(\frac{4^n}{\sqrt{\pi n^3}}\right)$$

n	0	1	2	3	4	5	6	7	8	9	10
c_n	1	1	2	5	14	42	132	429	1430	4862	16796

(CATALAN-Zahlen)

- Binärbäume
  - Definition
$$\mathcal{B} = \{\square\} + (\bigcirc \times \mathcal{B} \times \mathcal{B})$$
  - Klassifikation nach Grösse (Knotenzahl)
$$\mathcal{B}_n = \text{Binärbäume mit } n \text{ inneren Knoten}, c_n = \sharp\mathcal{B}_n$$
  - Strukturelle Rekursion
$$\mathcal{B}_{n+1} = \biguplus_{0\le k\le n} \mathcal{B}_k \times \mathcal{B}_{n-k}$$
  - Rekursion der Anzahlen (Segner)
$$c_{n+1} = \sum_{k=0}^{n} c_k \cdot c_{n-k}$$

- Geordnete Bäume
  - Definition
$$\mathcal{T} = \{\bullet\}\times\mathcal{T}^* = \biguplus_{k\ge 0}\{\bullet\}\times\mathcal{T}^k = \{\langle\bullet, t_1, t_2, \ldots, t_k\rangle; k \ge 0, t_i \in T\}$$
  - Klassifikation nach Grösse (Knotenzahl):
$$\mathcal{T}_n = \text{geordnete Bäume mit } n+1 \text{ Knoten}$$

- Klammersprache (Dyck)
  - Definition
$$\mathcal{D} \subseteq \{a, b\}^* \text{ erzeugt durch cfg } D \to \lambda \,|\, a \cdot D \cdot b \cdot D$$
  - Klassifikation nach Wortlänge
$$\mathcal{D}_n = \{w \in \mathcal{D};\ |w| = |w|_a + |w|_b = 2n\}$$
  - Charakterisierung
$$\forall w \in \{a, b\}^* \ :\ w \in \mathcal{D} \Leftrightarrow \begin{cases} |w|_a = |w|_b \\ w = u \cdot v \Rightarrow |u|_a \ge |u|_v \end{cases}$$
  - Eindeutigkeit der Zerlegung
$$\mathcal{D} \setminus \{\lambda\} \ni w = a \cdot u \cdot b \cdot v \ \text{ mit } u, v \in \mathcal{D}$$

- Primitive Klammersprache
  - Definition
$$\mathcal{P} = a \cdot \mathcal{D} \cdot b \subset \mathcal{D}$$
  - Charakterisierung
$$\forall w \in \{a, b\}^* \ :\ w \in \mathcal{P} \Leftrightarrow \begin{cases} |w|_a = |w|_b > 0 \\ w = u \cdot v, v \ne \lambda \Rightarrow |u|_a > |u|_v \end{cases}$$
  - Generierung der Dyck-Sprache $\mathcal{D}$ aus $\mathcal{P}$
$$\mathcal{D} = \mathcal{P}^* = \biguplus_{k\ge 0} \mathcal{P}^k \quad \text{(eindeutig)}$$
  - Alternative Grammatik für $\mathcal{D}$
$$D \to \lambda \,|\, P \cdot D$$
$$P \to a \cdot D \cdot b$$

- Bijektive Abbildung $\Phi : \mathcal{D} \to \mathcal{B}$ mit $\Phi(\mathcal{D}_n) = \mathcal{B}_n$ $(n \geq 0)$

$$\Phi(\lambda) = \square$$

$$\Phi(a \cdot u \cdot b \cdot v) = \langle \bigcirc, \Phi(u), \Phi(w) \rangle$$

(mit $u, v \in \mathcal{D}$ eindeutig)

- Bijektive Abbildung $\Psi : \mathcal{D} \to \mathcal{T}$ mit $\Psi(\mathcal{D}_n) = \mathcal{T}_n$ $(n \geq 0)$

$$\Psi(\lambda) = \langle \bullet \rangle$$

$$\Phi(\underbrace{au_1b}_{w_1} \cdot \underbrace{au_2b}_{w_2} \cdots \underbrace{au_kb}_{w_k}) = \langle \bullet, \Psi(u_1), \Psi(u_2), \ldots, \Psi(u_k) \rangle$$

(mit $w_1, w_2, \ldots, w_k \in \mathcal{P}$ eindeutig)

- Folgerung:

$$\sharp\mathcal{D}_n = \sharp\mathcal{T}_n = \sharp\mathcal{B}_n = c_n \quad (n \geq 0)$$

Illustration der Abbildung $\Phi : \mathcal{D} \to \mathcal{B}$

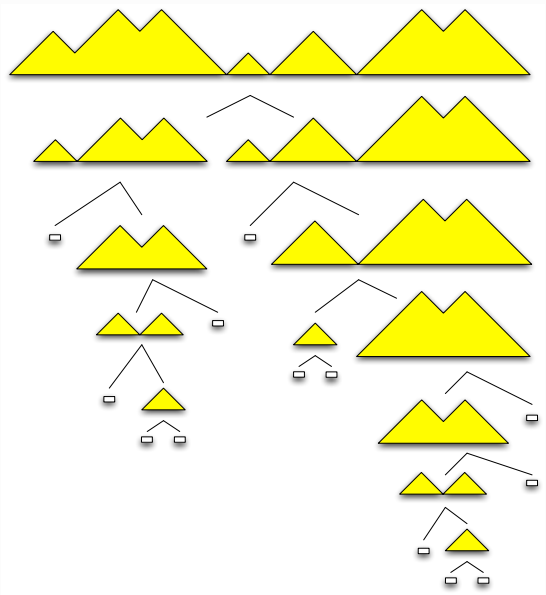
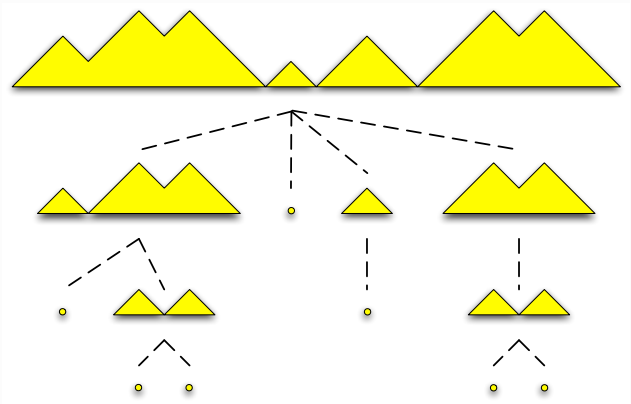


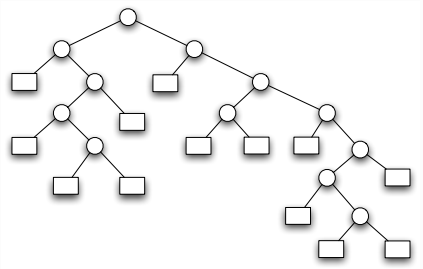
Illustration der Abbildung $\Psi : \mathcal{D} \to \mathcal{T}$

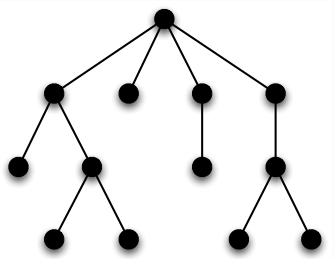


Beispiel: die zu dem Wort

$$w = aabaababbbbabaabbaaababbb \in \mathcal{D}_{12}$$

gehörenden Bäume $\Phi(w) \in \mathcal{B}_{12}$ und $\Psi(w) \in \mathcal{T}_{12}$:





Beweis der Catalan-Formel

- Beweisen wird die Rekursionsformel

$$(n+1)\cdot c_n = (4n-2)\cdot c_{n-1}\ (n \geq 0)$$

Daraus folgt per Induktion

$$c_n = \frac{4n-2}{n+1}c_{n-1} = \frac{4n-2}{n+1}\frac{1}{n}\binom{2n-2}{n-1} = \cdots = \frac{1}{n+1}\binom{2n}{n}$$

- Beweis der Rekursionsformel durch Konstruktion einer Bijektion zwischen

$$X_n = \{(t,a)\,;\ t \in \mathcal{B}_n, a \in E(t)\}$$

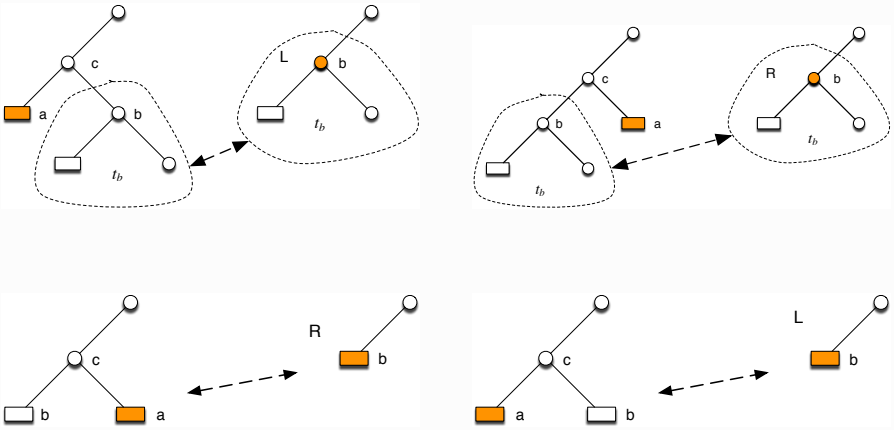
mit $\sharp X_n = (n+1)\cdot c_n$ und

$$Y_n = \{(t,b,r)\,;\ t \in \mathcal{B}_{n-1}, b \in E(t) \cup I(t), r \in \{L,R\}\}$$

mit $\sharp Y_n = 2(2n-1)\cdot c_{n-1}$

- Die Bijektion:
  - sei $t \in \mathcal{B}_n$ und $a \in E(t)$
  - sei $c \in I(t)$ der Vorgängerknoten von a in t,
    b der Bruderknoten von a in t,
    t_b der Teilbaum von t mit Wurzel b
  - entferne a aus t und ersetze c durch t_b:
    dies liefert $t' \in \mathcal{B}_{n-1}$ und $b \in I(t) \cup E(t)$
  - Bild von (t,a) unter dieser Abbildung ist (t',b,r) mit $r = L$ bzw. $L = R$, je nachdem, ob a linker oder rechter Nachfolger von c war
  - diese Konstruktion ist eindeutig umkehrbar

Illustration der Bijektion



Hamming-Geometrie der Bitvektoren

$\mathbb{B} = \{0, 1\}$ mit den Operationen

- $\wedge$ = Konjunktion ("und")
- $\vee$ = Disjunktion ("oder")
- $\neg$ oder $\overline{\ }$ = Negation ("nicht")

ist die zweielementige *boolesche Algebra*.

- $(\mathbb{B}, \oplus, 0)$ mit $\oplus$ = EXOR ist eine kommutative Gruppe.
- $(\mathbb{B}, \oplus, \wedge, 0, 1)$ ist ein Ring (*boolescher Ring*) und ein Körper ($\mathbb{F}_2$).

$\mathbb{B}^n = \{0, 1\}^n$: Bitvektoren $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ der Länge n
$\mathbb{B}^n$ mit den speziellen Elementen $\mathbf{0} = (0, 0, \ldots, 0), \mathbf{1} = (1, 1, \ldots, 1)$
und den komponentenweisen Operationen

- $\wedge$ = Konjunktion ("und")
- $\vee$ = Disjunktion ("oder")
- $\overline{\ }$ = Negation ("nicht")

ist eine *boolesche Algebra* mit 2^n Elementen.

- $(\mathbb{B}^n, \oplus, \mathbf{0})$ mit komponentenweisem $\oplus$ = EXOR ist eine kommutative Gruppe. (NB. $\overline{\mathbf{a}} = \mathbf{a} \oplus \mathbf{1}$).
- $(\mathbb{B}^n, \oplus, \wedge, \mathbf{0}, \mathbf{1})$ ist ein Ring (*boolescher Ring*) und ein Vektorraum der Dimension n über dem Körper $\mathbb{F}_2$.
- Standardbasis: $\mathbf{e}^k = (0, 0, \ldots, 0, 1_k, 0, \ldots, 0)$ $(1 \leq k \leq n)$.

- Für $\mathbf{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{B}^n$ bezeichet $\|\mathbf{a}\|$ die Anzahl der Komponenten $a_i = 1$ in $\mathbf{a}$ (HAMMING-Gewicht).
- Für $\mathbf{a}, \mathbf{b} \in \mathbb{B}^n$ bezeichnet

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} \oplus \mathbf{b}\| = \|\mathbf{a}\| + \|\mathbf{b}\| - 2 \cdot \|\mathbf{a} \wedge \mathbf{b}\|$$

  den HAMMING-Abstand.
- Die Funktion

$$d : \mathbb{B}^n \times \mathbb{B}^n \to \{0, 1, 2, \ldots, n\} : (\mathbf{a}, \mathbf{b}) \mapsto d(\mathbf{a}, \mathbf{b})$$

  ist eine *Metrik* auf $\mathbb{B}^n$, d.h. es gilt für $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{B}^n$:

$$d(\mathbf{a}, \mathbf{b}) = 0 \Leftrightarrow \mathbf{a} = \mathbf{b}$$
$$d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$$
$$d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c}) \geq d(\mathbf{a}, \mathbf{c})$$

  Translationsinvarianz: $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{a} \oplus \mathbf{c}, \mathbf{b} \oplus \mathbf{c})$

- $\mathbb{B}^n_k = \{\mathbf{a} \in \mathbb{B}^n ; \|\mathbf{a}\| = k\}$ $(0 \leq k \leq n)$
- $\sharp\mathbb{B}^n_k = \binom{n}{k}$
- $\mathbb{B}^n_{\leq k} = \{\mathbf{a} \in \mathbb{B}^n ; \|\mathbf{a}\| \leq k\}$ $(0 \leq k \leq n)$
  HAMMING-Kugel vom Radius k um den Nullvektor
- $S_k(\mathbf{a}) = \mathbf{a} \oplus \mathbb{B}^n_{\leq k}$
  HAMMING-Kugel vom Radius k um den Vektor $\mathbf{a}$
- $V(n, k) = \sharp\mathbb{B}^n_{\leq k} = \sum_{j=0}^{k} \binom{n}{j}$
  Volumen einer HAMMING-Kugel vom Radius k.
- $d(\mathbf{a}, \mathbf{b}) > k \Leftrightarrow \mathbf{b} \notin S_k(\mathbf{a}) \Leftrightarrow \mathbf{a} \notin S_k(\mathbf{b})$
- $d(\mathbf{a}, \mathbf{b}) > 2k \Leftrightarrow S_k(\mathbf{a}) \cap S_k(\mathbf{b}) = \emptyset$

Abschätzung des Kugelvolumens

Sei $0 \leq \lambda \leq 1/2$, also $\frac{\lambda}{1-\lambda} \leq 1$. Aus

$$\left[\frac{\lambda}{1-\lambda}\right]^k \geq \left[\frac{\lambda}{1-\lambda}\right]^{\lfloor \lambda n \rfloor} \qquad (0 \leq k \leq \lfloor \lambda n \rfloor)$$

und mit Verwendung der Binomialformel folgt

$$V(n, \lambda n) \leq \lambda^{-\lfloor \lambda n \rfloor}(1-\lambda)^{\lfloor \lambda n \rfloor - n} \leq 2^{n \cdot H(\lambda)}$$

Zusammen mit der Abschätzung für $\binom{n}{\lfloor \lambda n \rfloor}$ ergibt sich

$$\lim_{n \to \infty} \frac{1}{n} \log V(n, \lambda n) = H(\lambda)$$

Diskreter gedächtnisfreier stationärer Kanal

- Daten
  - Inputalphabet $A = \{a_1, a_2, \ldots, a_m\}$
  - Outputalphabet $B = \{b_1, b_2, \ldots, b_n\}$
  - Kanalmatrix $P = (p_{i,j}\,;\, 1 \leq i \leq m, 1 \leq j \leq n)$ mit $p_{i,j} \geq 0$ $(1 \leq i \leq m, 1 \leq j \leq n)$ und $\sum_{1 \leq j \leq n} p_{i,j} = 1$ (stochastische Matrix)
- Funktionsweise: wird $a_i \in A$ gesendet, so wird $b_j \in B$ empfangen mit Wahrscheinlichkeit

$$P(b_j | a_i) = p_{i,j} \quad (1 \leq i \leq m, 1 \leq j \leq n)$$

- für $N \geq 1$: wird $\mathbf{u} = (u_1, u_2, \ldots, u_N) \in A^N$ gesendet, so wird $\mathbf{v} = (v_1, v_2, \ldots, v_N) \in B^N$ empfangen mit Wahrscheinlichkeit

$$P(\mathbf{v} \,|\, \mathbf{u}) = \prod_{1 \leq k \leq N} P(v_k \,|\, u_k) = P(v_1 \,|\, u_1) \cdot P(v_2 \,|\, u_2) \cdots P(v_N \,|\, u_N)$$

Beispiele

- Binärer symmetrischer Kanal BSC_p
  $A = B = \mathbb{B} = \{0, 1\}$, $0 < p < 1$

$$P = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

- Binärer Kanal mit Löschung $A = \mathbb{B} = \{0, 1\}$, $B = \mathbb{B} \cup \{*\}$, $0 < \varepsilon < 1$

$$P = \begin{bmatrix} 1-\varepsilon & 0 & \varepsilon \\ 0 & 1-\varepsilon & \varepsilon \end{bmatrix}$$

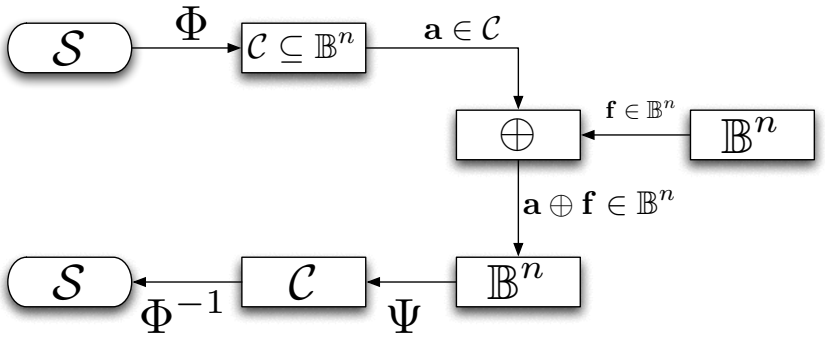


Abbildung: Informationsübertragung über einen gestörten Kanal

Nachrichtenübertragung mit BSC

- Quelle (source): $\mathcal{Q} = (S, \pi)$, wobei S Alphabet und $\pi = (\pi_s)_{s \in S}$ Wahrscheinlichkeitsverteilung auf S.
- Codierung: injektive Abbildung $\Phi : S \to \mathbb{B}^n$
  $\mathcal{C} = \Phi(\mathbb{B}^n)$: der durch Φ definierte Code
- Kanalmodell: die Wahrscheinlichkeit, beim Senden von $\mathbf{a} \in \mathbb{B}^n$ den Vektor $\mathbf{b} \in \mathbb{B}^n$ zu empfangen, d.h. dass der Fehler $\mathbf{f} = \mathbf{a} \oplus \mathbf{b}$ auftritt, ist

$$P(\mathbf{b} \mid \mathbf{a}) = bin_{n,p}(\mathbf{f}) = p^{\|\mathbf{f}\|}(1-p)^{n-\|\mathbf{f}\|}$$

- Der Empfänger ist bestrebt, aus der Kenntnis des empfangenen $\mathbf{b} \in \mathbb{B}^n$ das *mit grösster Wahrscheinlichkeit* gesendete $\mathbf{a} \in \mathcal{C}$ zu ermitteln.

- Unter der Annahme, dass alle $\mathbf{a} \in \mathcal{C}$ mit gleicher Wahrscheinlichkeit gesendet werden, d.h. $\pi(s) = 1/\sharp S$ für alle $s \in S$, muss man $\mathbf{a} \in \mathcal{C}$ bestimmen, für das $P(\mathbf{b} \mid \mathbf{a})$ maximal wird. (*maximum likelihood Decodierung*)
- Für $0 < p < 1/2$ gilt

$$0 \le j \le k \le n \quad \Rightarrow \quad p^j(1-p)^{n-j} \ge p^k(1-p)^{n-k}$$

- Daraus ergibt sich das Prinzip der *minimum distance* Decodierung:

  $\Psi : \mathbb{B}^n \to \mathcal{C} : \mathbf{b} \mapsto \mathbf{a} \in \mathcal{C}$ für das $d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} \oplus \mathbf{b}\|$ minimal ist

  Ein solches $\mathbf{a} \in \mathcal{C}$ muss nicht eindeutig bestimmt sein!

Beispiel (1)

- Quelle $S = \{s_1, s_2, \ldots, s_8\}$, $\pi_s = \frac{1}{8}\,(1 \le s \le 8)$
- Codierung Φ_1

s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
↓	↓	↓	↓	↓	↓	↓	↓
000	100	010	001	110	101	011	111

- Inputalphabet für den BSC_p ist $\Phi_1(S) = \mathcal{C}_1 = \mathbb{B}^3$
- Decodierung $\Psi_1(a_1a_2a_3) = a_1a_2a_3$, d.h. $\Psi_1 = id_{\mathbb{B}^3}$
- Die Wahrscheinlichkeit fehlerfreier Übertragung eines $(a_1a_2a_3) \in \mathbb{C}$ ist $(1-p)^3$. N Nachrichten werden mit Wahrscheinlichkeit $(1-p)^{3N}$ korrekt übertragen und decodiert.
- Es werden keine Fehler erkannt oder korrigiert

Beispiel (2)

- Quelle $S = \{s_1, s_2, \ldots, s_8\}$, $\pi_s = \frac{1}{8}\,(1 \le s \le 8)$
- Codierung Φ_2

s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
↓	↓	↓	↓	↓	↓	↓	↓
0000	1001	0101	0011	1100	1010	0110	1111

- Inputalphabet für den BSC_p ist $\Phi(S) = \mathcal{C}_2 = \{\mathbf{a} \in \mathbb{B}^4 \,;\, \|\mathbf{a}\| \text{ gerade}\}$
- Decodierung

$$\Psi_2(a_1a_2a_3a_4) = \begin{cases} a_1a_2a_3 & \text{falls } \|a_1a_2a_3a_4\| \text{ gerade} \\ \text{error} & \text{falls } \|a_1a_2a_3a_4\| \text{ ungerade} \end{cases}$$

- $a_1a_2a_3a_4 \in \mathbb{C}$ wird mit Wahrscheinlichkeit $(1-p)^4$ fehlerfrei übertragen, mit W.keit $4p(1-p)^3$ tritt ein 1-Bit-Fehler auf.
- 1-Bit-Fehler werden erkannt, aber nicht korrigiert.

Beispiel (3)

- Quelle $S = \{s_1, s_2, \ldots, s_8\}$, $\pi_s = \frac{1}{8}\,(1 \le s \le 8)$
- Codierung Φ_3

s_1	s_2	s_3	s_4	s_5	$\ldots$	s_8
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$
000000	100100	010010	001001	110110	$\ldots$	111111

- Inputalphabet für den BSC_p ist $\Phi_3(S) = \mathcal{C}_3 \subset \mathbb{B}^6$
- Decodierung

$$\Psi_3(a_1a_2a_3a_4a_5a_6) = \begin{cases} a_1a_2a_3 & \text{falls } a_1a_2a_3 = a_4a_5a_6 \\ \text{error} & \text{falls } a_1a_2a_3 \neq a_4a_5a_6 \end{cases}$$

- Es werden viele Fehler erkannt, aber sie können nicht korrigiert werden.

Beispiel (4)

- Quelle $S = \{s_1, s_2, \ldots, s_8\}$, $\pi_s = \frac{1}{8}\,(1 \le s \le 8)$
- Codierung Φ_4

s_1	s_2	s_3	$\ldots$	s_8
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$
$(000)^{2n+1}$	$(100)^{2n+1}$	$(010)^{2n+1}$	$\ldots$	$(111)^{2n+1}$

- Inputalphabet für den BSC_p ist $\Phi_4(S) = \mathcal{C}_4 \subset \mathbb{B}^{6n+3}$
- Decodierung

$$\Psi_4(a_1a_2\ldots a_{6n+3}) = \begin{cases} abc & \text{falls} \ge n+1 \text{ Dreierblöcke } = abc \\ \text{error} & \text{sonst} \end{cases}$$

- Es werden alle Fehler mit Gewicht $\le 2n$ erkannt.
  Alle Fehler mit Gewicht $\le n$ können korrigiert werden.

Codes

- Ein (binärer Block)-Code der Länge n ist eine Teilmenge $\mathcal{C} \subseteq \mathbb{B}^n$.
- Die *Minimaldistanz* von $\mathcal{C}$ ist

$$d_{\min}(\mathcal{C}) = \min_{\substack{\mathbf{a},\mathbf{b}\in\mathcal{C} \\ \mathbf{a}\neq\mathbf{b}}} d(\mathbf{a},\mathbf{b})$$

- Die *Coderate* von $\mathcal{C}$ ist

$$R(\mathcal{C}) = \frac{1}{n} \cdot \log \sharp\mathcal{C}$$

- $\mathcal{C}$ ist ein (n, K, d)-Code, falls $\mathcal{C} \subseteq \mathbb{B}^n$, $\sharp\mathcal{C} = K$ und $d_{min}(\mathcal{C}) = d$.

- Ein (n, K, d)-Code kann e Fehler *erkennen*, falls $e < d$ ist, d.h. falls

$$\forall \mathbf{a} \in \mathcal{C} : S_e(\mathbf{a}) \cap \mathcal{C} = \{\mathbf{a}\}$$

- Ein (n, K, d)-Code kann e Fehler *korrigieren*, falls $2e < d$ ist, d.h. falls

$$\forall \mathbf{a}, \mathbf{b} \in \mathcal{C} \text{ mit } \mathbf{a} \neq \mathbf{b} : S_e(\mathbf{a}) \cap S_e(\mathbf{b}) = \emptyset$$

- Für einen e Fehler korrigierenden (n, K, d)-Code gilt die *Kugelpackungsschranke*:

$$K \cdot V(n, e) \le 2^n, \text{ d.h. } R(\mathcal{C}) + H(\frac{e}{n}) \le 1$$

- Gesucht sind Codes mit hoher Rate und grosser Minimaldistanz: das sind antagonistische Anforderungen!
- $A(n, d)$: maximales K für das ein (n, K, d)-Code existiert.

Beispiele

- $d_{\min}(\mathcal{C}_1) = 1,\ R(\mathcal{C}_1) = 1$
- $d_{\min}(\mathcal{C}_2) = 2,\ R(\mathcal{C}_2) = \frac{3}{4}$
- $d_{\min}(\mathcal{C}_3) = 2,\ R(\mathcal{C}_3) = \frac{1}{2}$
- $d_{\min}(\mathcal{C}_4) = 2n+1,\ R(\mathcal{C}_4) = \frac{1}{2n+1}$

- $A(n,1) = 2^n, A(n,n) = 2$
- $A(n,2) = 2^{n-1}$
- $A(3,3) = 2, A(4,3) = 2, A(5,3) = 5, A(7,3) = 16$
- $A(n,2t+1) \leq \frac{2^n}{V(n,t)}$ (Hamming)
- $A(n,d) \leq 2^{n-d+1}$ (Singleton)
- $A(n,d) \geq \frac{2^n}{V(n,d-1)}$ (Gilbert-Varshamov)

Lineare Codes

- (binärer) *linearer* Code der Länge n = linearer Teilraum $\mathcal{C} \subseteq \mathbb{B}^n$.
  NB: hier gilt
  linearer Teilraum = Untergruppe = $\oplus$-abg. Teilmenge ($\neq \emptyset$)
- Hat $\mathcal{C}$ die Dimension $\dim \mathcal{C} = k$, so gilt $\sharp\mathcal{C} = 2^k$.
  Die Coderate ist dann $R(\mathcal{C}) = \frac{k}{n}$.
- Für lineare Codes $\mathcal{C}$ gilt immer

$$d_{\min}(\mathcal{C}) = \min_{\mathbf{0} \neq \mathbf{a} \in \mathcal{C}} \|\mathbf{a}\|$$

  Man spricht deshalb vom *Minimalgewicht*.
- Ein linearer Code $\mathcal{C} \subseteq \mathbb{B}^n$ mit $\dim \mathcal{C} = k$ und Minimalgewicht $d_{min}(\mathcal{C}) = d$ wird als $[n,k,d]$-Code bezeichnet.

Es gibt zwei wesentliche Möglichkeiten, lineare Codes $\mathcal{C} \subseteq \mathbb{B}^n$ zu beschreiben:

- mittels *Generatoren*, d.h. durch Angabe einer *Basis* $\mathbf{g}^1, \mathbf{g}^2, \ldots, \mathbf{g}^k$ von $\mathcal{C}$, also durch eine *Generatormatrix*

$$G = \begin{bmatrix} \mathbf{g}^1 \\ \mathbf{g}^2 \\ \vdots \\ \mathbf{g}^k \end{bmatrix}$$

  d.h. es gilt $\forall \mathbf{c} \in \mathbb{B}^n$:

$$\mathbf{c} \in \mathcal{C} \iff \exists \mathbf{x} \in \mathbb{B}^k : \mathbf{c} = \mathbf{x} \cdot G$$

- Da die Basis eines Vektorraumes nicht eindeutig bestimmt ist, gibt es auch immer mehrere Generatormatrizen zu einem linearen Code.

- mittels *Akzeptoren*, d.h. durch Angabe einer *Basis* $\mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^{n-k}$ des zu $\mathcal{C}$ orthogonalen Teilraumes

$$\mathcal{C}^\perp = \{\mathbf{b} \in \mathbb{B}^n \ : \ \forall \mathbf{c} \in \mathcal{C} : \mathbf{b} \cdot \mathbf{c}^t = 0\},$$

  also durch eine *Kontrollmatrix*

$$H = \begin{bmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \vdots \\ \mathbf{h}^{n-k} \end{bmatrix}$$

  d.h. es gilt $\forall \mathbf{c} \in \mathbb{B}^n$:

$$\mathbf{c} \in \mathcal{C} \iff H \cdot \mathbf{c}^t = \mathbf{0}^t$$

- Da die Basis eines Vektorraumes nicht eindeutig bestimmt ist, gibt es auch immer mehrere Kontrollmatrizen zu einem linearen Code.

Beispiel

- Ein Code $\mathcal{C} \subseteq \mathbb{B}^4$ der Dimension 2 sei gegeben durch die Generatormatrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

oder – gleichwertig – durch die Kontrollmatrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Der Code enthält die 4 Vektoren

$$0000, 1010, 0111, 1101$$

Sein Minimalgewicht und der Minimalabstand sind $d_{\min}(\mathcal{C}) = 2$.

Beispiel

- Ein Code $\mathcal{C} \subseteq \mathbb{B}^4$ der Dimension 3 sei gegeben durch die Generatormatrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

oder – gleichwertig – durch die Kontrollmatrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Der Code enthält die $2^3 = 8$ Vektoren aus $\mathbb{B}^4$ mit geradem Gewicht.

$$0000, 1001, 0101, 0011, 1100, 1010, 0110, 1111$$

Sein Minimalgewicht und der Minimalabstand sind $d_{\min}(\mathcal{C}) = 2$.

- Systematische Codierung
  - Bezeichnet $E_k \in \mathbb{B}^{k \times k}$ die Einheitsmatrix und ist $A \in \mathbb{B}^{k \times (n-k)}$ eine Matrix, so hat der von

$$G = \left[\begin{array}{c|c} E_k & A \end{array} \right]$$

erzeugte lineare $[n, k]$-Code die Kontrollmatrix

$$H = \left[\begin{array}{c|c} A^t & E_{n-k} \end{array} \right]$$

  - Bei der Codierung

$$\mathbb{B}^k \to \mathcal{C} : \mathbf{x} \mapsto \mathbf{x} \cdot G = [\mathbf{x} \,|\, \mathbf{x} \cdot A]$$

ist die ursprüngliche Nachricht $\mathbf{x}$ im Codewort sichtbar!

Codierung und Decodierung linearer Codes

- $\mathcal{C}$ linearer $[n, k, d]$-Code mit Generatormatrix G und Kontrollmatrix H.
- Quelle $S = \mathbb{B}^k$
- Codierung mittels linearer Transformation

$$\Phi : \mathbb{B}^k \to \mathbb{B}^n : \mathbf{x} \mapsto \mathbf{c} = \mathbf{x} \cdot G$$

- Tritt bei Übertragung Fehler $\mathbf{f} \in \mathbb{B}^n$ auf, d.h. wird $\mathbf{b} = \mathbf{c} \oplus \mathbf{f}$ empfangen, so gilt wegen $H \cdot \mathbf{c}^t = \mathbf{0}^t$ und Linearität:

$$\mathbf{s}^t = H \cdot \mathbf{b}^t = H \cdot \mathbf{c}^t \oplus H \cdot \mathbf{f}^t = H \cdot \mathbf{f}^t$$

d.h. empfangener Vektor $\mathbf{b}$ und Fehlervektor $\mathbf{f}$ haben das gleiche *Syndrom* $\mathbf{s}^t$.
- Codevektoren sind Vektoren mit Syndrom $\mathbf{0}^t$.

Syndromdecodierung

- Mit den genannten Daten $G, H, \mathbf{b}$:
  - berechne das Syndrom $\mathbf{s}^t = H \cdot \mathbf{b}^t$ des empfangenen Vektors $\mathbf{b}$
  - bestimme unter den Vektoren $\mathbf{b} \oplus \mathcal{C}$ mit dem gleichen Syndrom $\mathbf{s}^t$ einen Vektor $\mathbf{a}$ mit minimalem Gewicht (der mutmassliche Fehlervektor)
  - $\Psi(\mathbf{b}) = \mathbf{b} \oplus \mathbf{a} \in \mathcal{C}$
- Algebraisch gesprochen ist die Menge $\mathbf{b} \oplus \mathcal{C}$ eine *Nebenklasse* (coset) der Untergruppe $\mathcal{C}$ von $\mathbf{B}^n$. Einen Vektor von minimalem Gewicht (nicht notwendigerweise eindeutig) in einer solchen Klasse nennt man einen *Führer* der Nebenklasse (coset leader). Man spricht deshalb auch von coset-leader Decodierung. Für "kleine" Codes kann man effizient mit coset-leader Tabellen (Syndromtabellen) arbeiten, d.h. Tabelle mit (Fehler)Vektor minimalen Gewichts zu jedem möglichen Syndrom.

Zur Komplexität der linearen Decodierung

- Die beiden folgenden Probleme sind NP-vollständig:
  - (LD – linear decoding)
    Gegeben: eine Kontrollmatrix $H \in \mathbb{B}^{m \times n}$, ein (Syndrom)Vektor $\mathbf{s} \in \mathbb{B}^m$ und eine Zahl $w \in \mathbb{N}$.
    • Gibt es einen Vektor $\mathbf{x} \in \mathbb{B}^n$ mit $H \cdot \mathbf{x}^t = \mathbf{s}^t$ und $\|\mathbf{x}\| \leq w$?
  - (EMD – exact minimum distance)
    Gegeben: eine Kontrollmatrix $H \in \mathbb{B}^{m \times n}$ und eine Zahl $w \in \mathbb{N}$.
    • Gibt es einen Vektor $\mathbf{x} \in \mathbb{B}^n$ mit $H \cdot \mathbf{x}^t = \mathbf{0}^t$ und $\|\mathbf{x}\| = w$?
- E.R. Berlekamp, R.J. McEliece, H.C.A. van Tilburg, *On the inherent intractability of certain coding problems*, IEEE Transactions on Information Theory 24 (1978).

Der $[7, 4, 3]$-Hamming Code

- In $\mathbb{B}^7$ werden folgende Vektoren ausgezeichnet

$$\begin{array}{ll}
\mathbf{g}^0 = \mathbf{1}^7 = 1111111 & \mathbf{h}^0 = \mathbf{0}^7 = 0000000 \\
\mathbf{g}^1 = \oplus_{i\in\{1,2,4\}}\mathbf{e}^i = 1101000 & \mathbf{h}^1 = \oplus_{i\in\{3,5,6,7\}}\mathbf{e}^i = 0010111 \\
\mathbf{g}^2 = \oplus_{i\in\{2,3,5\}}\mathbf{e}^i = 0110100 & \mathbf{h}^2 = \oplus_{i\in\{4,6,7,1\}}\mathbf{e}^i = 1001011 \\
\mathbf{g}^3 = \oplus_{i\in\{3,4,6\}}\mathbf{e}^i = 0011010 & \mathbf{h}^3 = \oplus_{i\in\{5,7,1,2\}}\mathbf{e}^i = 1100101 \\
\mathbf{g}^4 = \oplus_{i\in\{4,5,7\}}\mathbf{e}^i = 0001101 & \mathbf{h}^4 = \oplus_{i\in\{6,1,2,3\}}\mathbf{e}^i = 1110010 \\
\mathbf{g}^5 = \oplus_{i\in\{5,6,1\}}\mathbf{e}^i = 1000110 & \mathbf{h}^5 = \oplus_{i\in\{7,2,3,4\}}\mathbf{e}^i = 0111001 \\
\mathbf{g}^6 = \oplus_{i\in\{6,7,2\}}\mathbf{e}^i = 0100011 & \mathbf{h}^6 = \oplus_{i\in\{1,3,4,5\}}\mathbf{e}^i = 1011100 \\
\mathbf{g}^7 = \oplus_{i\in\{7,1,3\}}\mathbf{e}^i = 1010001 & \mathbf{h}^7 = \oplus_{i\in\{2,4,5,6\}}\mathbf{e}^i = 0101110
\end{array}$$

Beachte: $\mathbf{h}^i = \mathbf{g}^i \oplus \mathbf{1}^7$ $(0 \leq i \leq k)$

Die Fano-Ebene PG(2,2)

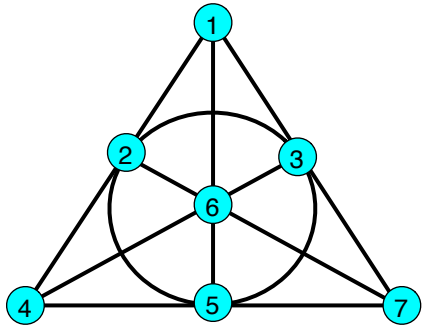


Abbildung: Fano-Ebene

- Eine *endliche projektive Ebene* besteht aus einer endlichen Menge von Punkten und einer endlichen Menge von Geraden und einer Inzidenzrelation ("liegt auf) zwischen Punkten und wobei gilt:
  - Je zwei Punkte liegen auf genau einer gemeinsamen Geraden
  - Je zwei Geraden schneiden sich in genau einem Punkt
  - Es gibt vier Punkte, so dass je drei von ihnen nicht auf einer Geraden liegen
- Zu einer endlichen projektiven Ebene gibt es eine Zahl n, die *Ordnung*, so dass gilt:
  - Jede Gerade enthält genau $n+1$ Punkte
  - Jeder Punkt liegt auf $n+1$ Geraden
  - Es gibt insgesamt n^2+n+1 Punkte
  - Es gibt insgesamt n^2+n+1 Geraden
- Die FANO-Ebene ist eine (sogar die einzige) projektive Ebene der Ordnung 2, also die kleinstmögliche projektive Ebene.

Zur Geometrie der FANO-Ebene

- Die FANO-Ebene enthält sieben Punkte $\{\mathbf{1},\mathbf{2},\ldots,\mathbf{7}\}$ und sieben Geraden $\{\mathbf{g}^1,\mathbf{g}^2,\ldots,\mathbf{g}^7\}$, wobei

$$\mathbf{g}^k = \{\mathbf{k},\mathbf{k}+\mathbf{1},\mathbf{k}+\mathbf{3}\} \pmod 7 \quad (1 \le k \le 7)$$

- Die drei Geraden, die durch den Punkt $\mathbf{k}$ gehen, sind $\mathbf{g}^k,\mathbf{g}^{k+4},\mathbf{g}^{k+6}$ $(1 \le k \le 7)$. Es gilt also

$$\mathbf{g}^k \oplus \mathbf{g}^{k+4} \oplus \mathbf{g}^{k+6} = \mathbf{1}^7$$

- Sind $\mathbf{g}^i$ und $\mathbf{g}^j$ zwei Geraden, so haben diese genau einen Schnittpunkt ℓ und es gibt noch genau eine weitere Gerade $\mathbf{g}^k$, die ℓ enthält. Also ist $\mathbf{g}^i \oplus \mathbf{g}^j \oplus \mathbf{g}^k = \mathbf{1}^7$ und somit

$$\mathbf{g}^i \oplus \mathbf{g}^j = \mathbf{g}^k \oplus \mathbf{1}^7 = \mathbf{h}^k$$

- $\mathcal{H} := \{\mathbf{h}^i\,;\, 0 \le i \le 7\}$, $\mathcal{G} := \mathcal{H} \cup \{\mathbf{g}^i\,;\, 0 \le i \le 7\}$
- Fundamentale Beobachtung (FANO-Ebene!):
  Zu jedem Paar (i,j) mit $0 \le i,j \le 7$ gibt es ein k mit $0 \le k \le 7$ mit : $\mathbf{g}^i \oplus \mathbf{g}^j = \mathbf{h}^k$
- Folgerung: $\mathcal{G}$ und $\mathcal{H}$ sind unter $\oplus$ abgeschlossen, also lineare Codes der Länge 7, denn

$$\begin{aligned}
\mathbf{g}^i \oplus \mathbf{g}^j &= \mathbf{h}^k \\
\mathbf{g}^i \oplus \mathbf{h}^j &= \mathbf{g}^i \oplus \mathbf{g}^j \oplus \mathbf{1}^7 = \mathbf{h}^k \oplus \mathbf{1}^7 = \mathbf{g}^k \\
\mathbf{h}^i \oplus \mathbf{h}^j &= \mathbf{g}^i \oplus \mathbf{1}^7 \oplus \mathbf{g}^j \oplus \mathbf{1}^7 = \mathbf{h}^k
\end{aligned}$$

- $\mathcal{G}$ und $\mathcal{H}$ sind lineare Teilräume von $\mathbb{B}^7$, $\dim \mathcal{G} = 4$, $\dim \mathcal{H} = 3$.
- $\mathcal{G}$ ist der $[7,4,3]$-HAMMING-Code, $\mathcal{H}$ der dazu *duale* $[7,3,4]$-Code.

- Nachweis der Orthogonalität von $\mathcal{G}$ und $\mathcal{H}$

  Mit $\mathbf{g}^i \oplus \mathbf{h}^j = \mathbf{g}^k$ (wie vorher) gilt:
  - wegen $\|\mathbf{g}^i\|, \|\mathbf{g}^k\| \in \{3,7\}$ und $\|\mathbf{h}^j\| \in \{0,4\}$ und

$$\|\mathbf{g}^k\| = \|\mathbf{g}^i \oplus \mathbf{h}^j\| = \|\mathbf{g}^i\| + \|\mathbf{h}^j\| - 2 \cdot \|\mathbf{g}^i \wedge \mathbf{h}^j\|$$

    gilt $\|\mathbf{g}^i \wedge \mathbf{h}^j\| \equiv 0 \mod 2$, also $\mathbf{h}^j \cdot (\mathbf{g}^i)^t = 0$
  - wegen $\|\mathbf{h}^i\|, \|\mathbf{h}^j\|, \|\mathbf{h}^k\| \in \{0,4\}$ und

$$\|\mathbf{h}^k\| = \|\mathbf{h}^i \oplus \mathbf{h}^j\| = \|\mathbf{h}^i\| + \|\mathbf{h}^j\| - 2 \cdot \|\mathbf{h}^i \wedge \mathbf{h}^j\|$$

    gilt $\|\mathbf{h}^i \wedge \mathbf{h}^j\| \equiv 0 \mod 2$, also $\mathbf{h}^j \cdot (\mathbf{h}^i)^t = 0$
  -

- Je 4 linear unabhängige Vektoren aus $\mathcal{G}$ bilden eine Basis und können für eine Generatormatrix genommen werden, z.B.

$$G = \begin{pmatrix} \mathbf{g}^1 \\ \mathbf{g}^2 \\ \mathbf{g}^3 \\ \mathbf{g}^4 \end{pmatrix} = \begin{pmatrix} 1101000 \\ 0110100 \\ 0011010 \\ 0001101 \end{pmatrix}$$

- Je 3 linear-unabhängige Vektoren aus $\mathcal{H}$ bilden eine Basis und können für eine Kontrollmatrix genommen werden, z.B.

$$H = \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = \begin{pmatrix} 0010111 \\ 1001011 \\ 1100101 \end{pmatrix}$$

Beachte: die Spalten von H enthalten jeden Vektor $\neq \mathbf{0}^3$ aus $\mathbb{B}^3$ genau einmal!

Eine interessante Eigenschaft

- Wegen $d_{\min}(\mathcal{G}) = 3$ ist der $[7, 4, 3]$-HAMMING-Code 1-Fehler-korrigierend, die HAMMING-Kugeln vom Radius 1

$$\mathbf{c} \oplus \mathbb{B}^7_{\leq 1} \quad (\mathbf{c} \in \mathcal{G})$$

sind paarweise disjunkt.

- Jede dieser HAMMING-Kugeln enthält genau $V(7, 1) = 8$ Elemente

$$\mathbf{c} \text{ und } \mathbf{c} \oplus \mathbf{e}^i \ (1 \leq i \leq 7)$$

- Wegen

$$\sharp\mathcal{G} \cdot V(7, 1) = 2^4 \cdot 8 = 2^7 = \sharp\mathbb{B}^7$$

bilden diese HAMMING-Kugeln eine *Zerlegung* von $\mathbb{B}^7$. Man sagt: der $[7, 4, 3]$-HAMMING-Code ist ein *perfekter* Code.

- Perfekte Codes sind extrem selten!

- Sei $r > 0$ und $n = 2^r - 1$. Sei H eine $r \times n$-Matrix, die alle Elemente von $\mathbb{B}^r$ $(\neq \mathbf{0}^r)$ als Spaltenvektoren enthält.
- Die Matrix H hat den Rang r, kann also als Kontrollmatrix eines Codes $\mathcal{G}$ der Länge n und der Dimension $\dim \mathcal{G} = n - r = 2^r - r - 1$ aufgefasst werden.
- Da je zwei Spalten von H linear-unabhängig sind, enthält $\mathcal{G}$ keine Vektoren vom Gewicht 1 oder 2. Also gilt $d_{\min}(\mathcal{G}) \geq 3$, d.h. der Code $\mathcal{G}$ ist 1-Fehler-korrigierend.
- Wegen

$$\sharp\mathcal{G} \cdot V(n, 1) = 2^{n-r} \cdot (n + 1) = 2^n$$

bilden die HAMMING-Kugeln vom Radius 1 um die Codevektoren eine Zerlegung von $\mathbb{B}^n$, es gilt also $d_{\min}(\mathcal{G}) = 3$ und der Code ist perfekt.

- Dieser $[2^r - 1, 2^r - r - 1, 3]$-Code heisst (binärer) HAMMING-Code der Ordnung r.

Zur Decodierung der HAMMING Codes

- H Kontrollmatrix des $[2^r - 1, 2^r - r - 1, 3]$-HAMMING Codes der Ordnung r.
- Die Spalten von H sind genau die Vektoren $\neq \mathbf{0}$ aus $\mathbb{B}^r$. Zu jedem $\mathbf{b} \in \mathbb{B}^{2^r-1}$ gibt es also genau ein $1 \leq i \leq 2^r - 1$ mit $H \cdot \mathbf{b}^t = H \cdot \mathbf{e}^i$, d.h. das Syndrom ist die i-te Spalte der Matrix H. Dann ist $\mathbf{e}^i$ der gesuchte coset leader!
- Decodierung
  - empfange $\mathbf{b} \in \mathbb{B}^{2^r-1}$ und berechne das Syndrom $\mathbf{s} = H \cdot \mathbf{b}^t$
  - $\Psi(\mathbf{b}) = \mathbf{b} \oplus \mathbf{e}^i$, wobei $\mathbf{s}$ die i-te Spalte von H ist.
- R.W. HAMMING, *Error detecting and error correcting codes*, Bell Systems Technical Journal, 29 (1950).

Reed-Muller Codes

- Die Abbildung

$$\rho_n : \mathbb{B}^{2n} \equiv \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^{2n} : (\mathbf{a}, \mathbf{b}) \mapsto (\mathbf{a} \oplus \mathbf{b}, \mathbf{b})$$

  ist linear und invertierbar ($\rho_n \circ \rho_n = id$, also $\rho^{-1} = \rho$).
- $A \subseteq \mathbb{B}^{2n}$ linear unabhängig $\Rightarrow$ $\rho_n(A) \subseteq \mathbb{B}^{2n}$ linear unabhängig
- A Basis von $\mathbb{B}^{2n}$ $\Rightarrow$ $\rho_n(A)$ Basis von $\mathbb{B}^{2n}$
- $A, B \subseteq \mathbb{B}^n$ linear unabhängig $\Rightarrow$

  $\rho_n(A \times \mathbf{0}^n \cup \mathbf{0}^n \times B) = \{(\mathbf{a}, \mathbf{0}^n); \mathbf{a} \in A\} \cup \{(\mathbf{b}, \mathbf{b}); \mathbf{b} \in B\} \subseteq \mathbb{B}^{2n}$

  linear unabhängig
- A, B Basen von $\mathbb{B}^n$ $\Rightarrow$ $\rho_n(A \times \mathbf{0}^n \cup \mathbf{0}^n \times B)$ Basis von $\mathbb{B}^{2n}$

- Für $m = 0, 1, 2, \ldots$ werden Basen $\mathcal{G}_m$ von $\mathbb{B}^n$ mit $n = 2^m$ definiert:
  - $\mathcal{G}_0 = \{\mathbf{1}^1\}$
  - $\mathcal{G}_{m+1} = \rho_n(\mathcal{G}_m \times \mathbf{0}^n) \uplus \rho_n(\mathbf{0}^n \times \mathcal{G}_m)$
    $= \{(\mathbf{a}, \mathbf{0}^n); \mathbf{a} \in \mathcal{G}_m\} \cup \{(\mathbf{a}, \mathbf{a}); \mathbf{a} \in \mathcal{G}_m\}$
- Eigenschaften:
  - $\sharp\mathcal{G}_m = 2^m$
  - Für alle $\mathbf{a} \in \mathcal{G}_m$ ist $\|\mathbf{a}\| = 2^r$ für ein $r \in \{0, 1, 2, \ldots, m\}$
  - Sei $\mathcal{G}_{m,r} = \{\mathbf{a} \in \mathcal{G}_m; \|\mathbf{a}\| = 2^{m-r}\}$.
    Dann gilt $\mathcal{G}_{m,0} = \{\mathbf{1}^n\}$ $(m \geq 0)$ und

$$\mathcal{G}_{m+1,r+1} = \rho_n(\mathcal{G}_{m,r} \times \mathbf{0}^n) \uplus \rho_n(\mathbf{0}^n \times \mathcal{G}_{m,r+1}) \quad (0 \leq r \leq m)$$

    also insbesondere

$$\sharp\mathcal{G}_{m+1,r+1} = \sharp\mathcal{G}_{m,r} + \sharp\mathcal{G}_{m,r+1} \quad (0 \leq r \leq m)$$

    und daher (per Induktion)

$$\sharp\mathcal{G}_{m,r} = \binom{m}{r} \quad (0 \leq r \leq m)$$

- Der (binäre) Reed-Muller Code $\mathcal{RM}_{m,r}$ der Länge $n = 2^m$ und der Ordnung r $(0 \leq r \leq m)$ ist der von

$$\mathcal{G}_{m,0} \cup \mathcal{G}_{m,1} \cup \ldots \cup \mathcal{G}_{m,r}$$

  erzeugte lineare Teilraum von $\mathbb{B}^n$.
- Eigenschaften
  - Inklusion als Teilräume

$$\{\mathbf{0}^n, \mathbf{1}^n\} = \mathcal{RM}_{m,0} \subset \mathcal{RM}_{m,1} \subset \mathcal{RM}_{m,2} \subset \ldots \subset \mathcal{RM}_{m,m} = \mathbb{B}^n$$

  - rekursive Darstellung

$$\mathcal{RM}_{m+1,r+1} = \rho_n(\mathcal{RM}_{m,r} \times \mathcal{RM}_{m,r+1})$$

  - Dimension

$$\dim(\mathcal{RM}_{m,r}) = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r} = V(m, r)$$

  - Minimalabstand: $d_{\min}(\mathcal{RM}_{m,r}) = 2^{m-r}$

Beispiele:

- $\mathcal{G}_{0,0} = \{1\}$
- $\mathcal{G}_{1,0} = \{11\}$
  $\mathcal{G}_{1,1} = \{10\}$
- $\mathcal{G}_{2,0} = \{1111\}$
  $\mathcal{G}_{2,1} = \{1100, 1010\}$
  $\mathcal{G}_{2,2} = \{1000\}$
- $\mathcal{G}_{3,0} = \{11111111\}$
  $\mathcal{G}_{3,1} = \{11110000, 11001100, 10101010\}$
  $\mathcal{G}_{3,2} = \{11000000, 10100000, 10001000\}$
  $\mathcal{G}_{3,3} = \{10000000\}$

Alternative Beschreibung der REED-MULLER Basisvektoren
Für $n = 2^m$ werden Vektoren $\mathbf{v}^1, \cdots, \mathbf{v}^m$ definiert

$$\mathbf{v}^1 = \mathbf{1}^{2^{m-1}}\mathbf{0}^{2^{m-1}}$$
$$\mathbf{v}^2 = \mathbf{1}^{2^{m-2}}\mathbf{0}^{2^{m-2}}\mathbf{1}^{2^{m-2}}\mathbf{0}^{2^{m-2}}$$
$$\cdots$$
$$\mathbf{v}^k = \left(\mathbf{1}^{2^{m-k}}\mathbf{0}^{2^{m-k}}\right)^{2^{r-1}}$$
$$\cdots$$
$$\mathbf{v}^m = (\mathbf{10})^{2^{m-1}}$$

Für $1 \leq r \leq m$ besteht $\mathcal{G}_{m,r}$ aus den Vektoren $\bigwedge_{i\in A} \mathbf{v}^i$, wobei A r-elementige Teilmenge von $\{1, 2, \ldots, m\}$ ist.

Andere Formulierung dieser Beschreibung:

- Es seien $X_1, X_2, \ldots, X_m$ boolesche Variable.
  Jede Variable definiert eine (lineare) Funktion
  $$X_k : \mathbb{B}^m \to \mathbb{B} : \mathbf{b} = (b_1, b_2, \ldots, b_m) \mapsto b_k \quad (1 \leq k \leq m)$$
- Für jeden Bitvektor $\mathbf{c} = (c_1, c_2, \ldots, c_m) \in \mathbb{B}^m$ definiert das boolesche Monom $X_\mathbf{c} = \bigwedge_{c_i=1} X_i$ eine Funktion:
  $$X_\mathbf{c} : \mathbb{B}^m \to \mathbb{B} : \mathbf{b} = (b_1, b_2, \ldots, b_m) \mapsto \bigwedge_{c_k=1} b_k$$
  d.h. $X_\mathbf{c}(\mathbf{b}) = 1 \quad \Leftrightarrow \quad \mathbf{c} \leq \mathbf{b}$
- Ein boolesches Polynom ist eine Summe von Monomen, also durch eine Abbildung $p : \mathbb{B}^m \to \mathbb{B}$ gegeben. Dies definiert eine Funktion
  $$X_p : \mathbb{B}^m \to \mathbb{B} : \mathbf{b} \mapsto \bigoplus_{p(\mathbf{c})=1} X_\mathbf{c}(\mathbf{b})$$

- Zu jeder booleschen Funktion $f : \mathbb{B}^m \to \mathbb{B}$ gibt es genau ein boolesches Polynom X_p, das diese Funktion darstellt:
  $$\forall \mathbf{b} \in \mathbb{B} : f(\mathbf{b}) = X_p(\mathbf{b})$$
  (*Ring-Normalform* von booleschen Funktionen).
- Der REED-MULLER Code $\mathcal{RM}_{m,r}$ besteht genau aus den Werte-Vektoren
  $$\mathbf{v}_p = \left(X_p(\mathbf{b})\right)_{\mathbf{b}\in\mathbb{B}^m}$$
  wo p ein boolesches Polynom vom Grad $\leq r$ ist.

Zum Minimalabstand (Beweis durch Induktion)

- $d_{\min}(\mathcal{RM}_{n+1,r+1}) \leq 2^{m-r}$
  - Ist $\mathbf{a} \in \mathcal{RM}_{m,r}$ mit $\|\mathbf{a}\| = 2^{m-r}$, so ist $(\mathbf{a}, \mathbf{0}^n) \in \mathcal{RM}_{m+1,r+1}$ mit $\|(\mathbf{a}, \mathbf{0}^n)\| = 2^{m-r} = 2^{(m+1)-(r+1)}$
  - Ist $\mathbf{a} \in \mathcal{RM}_{m,r+1}$ mit $\|\mathbf{a}\| = 2^{m-r-1}$, so ist $(\mathbf{a}, \mathbf{a}) \in \mathcal{RM}_{m+1,r+1}$ mit $\|(\mathbf{a}, \mathbf{a})\| = 2 \cdot 2^{m-r-1} = 2^{(m+1)-(r+1)}$
- $d_{\min}(\mathcal{RM}_{m+1,r+1}) \geq 2^{m-r}$
  Für $\mathbf{w} = (\mathbf{u} \oplus \mathbf{v}, \mathbf{v}) \in \mathcal{RM}_{m+1,r+1}$ mit mit $\mathbf{u} \in \mathcal{RM}_{m,r}$, $\mathbf{v} \in \mathcal{RM}_{m,r+1}$ und mit $\mathbf{w} \neq \mathbf{0}^{2n}$ gilt
  - $\mathbf{v} = \mathbf{0}^n \Rightarrow \mathbf{u} \neq \mathbf{0}^n$, also $\|\mathbf{w}\| = \|(\mathbf{u}, \mathbf{0}^n)\| = \|u\| \geq 2^{m-r}$
  - $\mathbf{u} = \mathbf{0}^n \Rightarrow \mathbf{v} \neq \mathbf{0}^n$, also $\|\mathbf{w}\| = \|(\mathbf{v}, \mathbf{v})\| = 2 \cdot \|\mathbf{v}\| \geq 2 \cdot 2^{m-r-1} = 2^{m-r}$
  - $\mathbf{u} \neq \mathbf{0}^n \neq \mathbf{v}$ und $\mathbf{u} = \mathbf{v} \Rightarrow \mathbf{w} = (\mathbf{0}^n, \mathbf{v})$ und $\mathbf{v} \in \mathcal{RM}_{m,r}$, also $\|\mathbf{w}\| = \|\mathbf{v}\| \geq 2^{m-r}$
  - $\mathbf{u} \neq \mathbf{0}^n \neq \mathbf{v}$ und $\mathbf{u} \neq \mathbf{v} \Rightarrow \mathbf{u} \oplus \mathbf{v} \in \mathcal{RM}_{m,r+1}$, also $\|\mathbf{w}\| = \|\mathbf{u} \oplus \mathbf{v}\| + \|\mathbf{v}\| \geq 2 \cdot 2^{m-r-1} = 2^{m-r}$

Bemerkungen: Die Reed-Muller Codes

- gehen zurück auf die Arbeiten
  - I.S. Reed, *A Class of Multiple Error Correcting Codes and the Decoding Scheme*, IRE Transactions Inform. Theory (1954).
  - D.E. Muller, *Application of Boolean Algebra to Switching Circuit Design*, IRE Transactions Electronic Computing (1954).
- lassen sich elegant und effizient mit *majority logic* decodieren – dank enger Beziehung zu geometrischen Konfigurationen (diskrete euklidische und projektive Geometrie).
- gehören, neben den Hamming und den Golay Codes, zu frühesten interessanten und nützlichen linearen Codes.
- fanden breite Beachtung dank ihrer Verwendung in Projekten des Nasa (1969–1976), z.b. der $[32, 6, 16]$-Code $\mathcal{RM}_{5,1}$ im Mariner-Projekt. → E.C. Posner, *Combinatorial Structures in Planetary Reconnaissance*, in: H.B. Mann (ed.), *Error Correcting Codes*, Wiley (1968).

Zyklische Codes

- Ein (binärer) linearer $[n, k, d]$ Code $\mathcal{C} \subseteq \mathbb{B}^n$ ist *zyklisch*, wenn er mit jedem Codewort auch dessen zyklische shifts enthält:

$$\mathbf{a} = (a_1, a_2, \ldots, a_n) \in \mathcal{C} \Rightarrow (a_2, a_3, \ldots, a_n, a_1) \in \mathcal{C}$$

- Die meisten der praktisch verwendeten Blockcodes sind zyklische Codes (z.B. Hamming Codes, Golay Codes, verkürzte Reed-Muller Codes, BCH Codes, Reed-Solomon Codes)
- Zyklische Codes lassen sich elegant mit Mitteln der *Polynomarithmetik* beschreiben, untersuchen und implementieren (→ lineare Schieberegister)
- Effiziente Codierungsalgorithmen benutzen lineare Schieberegister; effiziente Decodierung beruht auf Polynomarithmetik (euklidischer Algorithmus!)
- sehr lange zyklische Codes sind nicht besonders gut im Sinne der asymptotische Schranken, bessere Codes (z.B. Goppa Codes) erfordern aber einen viel höheren mathematischen Aufwand!

- $A(n, d)$: maximales K, für das ein (n, K, d)-Code existiert
- obere Schranke (Plotkin-Schranke)

$$A(n, d) \leq \frac{2d}{2d - n} \quad \text{für } 2d > n$$

  - Beweis: für einen (n, K, d)-Code $\mathcal{C}$ gilt

$$d \cdot \frac{K(K-1)}{2} \leq \frac{1}{2} \sum_{\mathbf{a},\mathbf{b} \in \mathcal{C}} d(\mathbf{a}, \mathbf{b}) \leq n \cdot \max_{1 \leq k \leq K} k \cdot (K - k) = n \cdot \frac{K^2}{4}$$

- untere Schranke (Gilbert-Varshamov-Schranke)

$$A(n, d) \geq \frac{2^n}{V(n, d-1)}$$

  - Beweis: Ist $\mathcal{C}$ ein (n, K, d)-Code, so gilt wegen der Maximalität von K:

$$\mathbb{B}^n \subseteq \bigcup_{\mathbf{a} \in \mathcal{C}} S_{d-1}(\mathbf{a})$$

Asymptotische Aussagen

- Sei $0 \leq \delta \leq 1$ und $R(\delta)$ die maximale Rate, die asymptotisch für Codes der Länge n mit relativer Minimaldistanz $\delta = d/n$ erreicht werden kann. D.h.

$$R(\delta) = \limsup_{n \to \infty} \frac{1}{n} \log A(n, \delta \cdot n)$$

Dann gilt

- (Plotkin)

$$R(\delta) \begin{cases} \leq 1 - 2 \cdot \delta & \text{falls } 0 \leq \delta \leq 1/2 \\ = 0 & \text{falls } 1/2 \leq \delta \leq 1 \end{cases}$$

- (Gilbert-Varshamov)

$$R(\delta) \geq 1 - H(\delta) \qquad (0 \leq \delta \leq 1/2)$$

Beweis der asymptotischen PLOTKIN-Schranke

- Ist $\mathcal{C}$ ein (n, K, d)-Code, so kann man daraus für $r = 1, 2, \ldots$ einen $(n - r, K_r, d)$-Code $\mathcal{C}_r$ konstruieren mit $K_r \geq K/2^r$.
- Für $n - r = 2d - 2$ erhält man nach PLOTKIN:

$$\frac{K}{2^r} \leq \sharp\mathcal{C}_r \leq \frac{2d}{2d - (n - r)} = d, \quad \text{also } K \leq d \cdot 2^r$$

- Daraus folgt mit $d = \delta \cdot n$

$$\begin{aligned} R(\delta) &\leq \limsup_{n\to\infty} \frac{1}{n} \cdot \log(\delta \cdot n \cdot 2^{n-2\delta\cdot n+2}) \\ &= \lim_{n\to\infty} \left(\frac{\log \delta}{n} + \frac{\log n}{n} + 1 - 2\delta + \frac{2}{n} \right) \\ &= 1 - 2\,\delta \end{aligned}$$

Beweis der asymptotischen GILBERT-VARSHAMOV-Schranke

- Aus der asymptotischen Abschätzung des Volumens von HAMMING-Kugeln folgt sofort:

$$\begin{aligned} R(\delta) &\geq \limsup_{n\to\infty} \frac{1}{n} \cdot \log \frac{2^n}{V(n, \delta \cdot n)} \\ &= 1 - \lim_{n\to\infty} \frac{1}{n} \cdot \log V(n, \delta \cdot n) \\ &= 1 - H(\delta) \end{aligned}$$

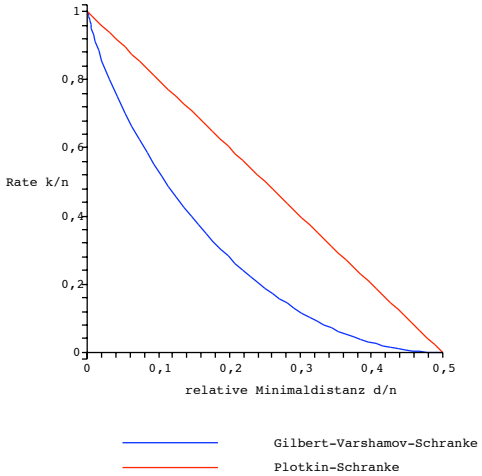


Abbildung: Asymptotische Schranken für Codes

- Es gibt neben der GILBERT-VARSHAMOV-Schranke und der PLOTKIN-Schranke noch viele weitere Schranken, vor allem obere (HAMMING, ELIAS, MCELIECE, ...).
- Die GILBERT-VARSHAMOV-Schranke war von 1952 bis 1982 die beste bekannte untere Schranke! Erst dann gelang es TSFASMAN, VLADUT, ZINK mit Hilfe von GOPPA-Codes (1977 erfunden) Codes zu konstruieren, welche die GILBERT-VARSHAMOV-Schranke übertreffen.
- Literatur
  - D. WELSH, *Codes and Cryptography*, Oxford UP (1988).
  - F.J. MACWILLIAMS, N. J.A. SLOANE, *The Theory of Error-Correcting Codes*, Elsevier (1997).

Lexicografische Ordnung und Grössenvergleich

Notation

- $\mathcal{B} = \{0,1\}$: "bits", $\mathcal{B}^n$: bitstrings der Länge n, induktiv definiert durch

$$\mathcal{B}^0 = \{\epsilon\},\ \mathcal{B}^1 = \mathcal{B},\ \mathcal{B}^{n+1} = \mathcal{B} \cdot \mathcal{B}^n\ (n \geq 1)$$

- bitstrings endlicher Länge

$$\mathcal{B}^* = \biguplus_{n \geq 0} \mathcal{B}^n, \quad \mathcal{B}^+ = \biguplus_{n > 0} \mathcal{B}^n$$

  Längenfunktion $\mathcal{B}^* \to \mathbb{N}$: $|a| = n \Leftrightarrow a \in \mathcal{B}^n\ (n \geq 0)$

- Konstruktoren head und tail

  $\mathtt{h} : \mathcal{B}^+ \to \mathcal{B}$, $\mathtt{t} : \mathcal{B}^+ \to \mathcal{B}^*$, definiert durch: $\mathtt{h}(a) \cdot \mathtt{t}(a) = a\ (a \in \mathcal{B}^+)$

- Interpretation als Binärdarstellung natürlicher Zahlen

  $\mathtt{val} : \mathcal{B}^+ \to \mathbb{N} : a \mapsto \mathtt{if}\ |a| = 1\ \mathtt{then}\ a\ \mathtt{else}\ h(a) \cdot 2^{|\mathtt{t}(a)|} + \mathtt{val}(\mathtt{t}(a))$

- lexicografische Ordnung auf $\mathcal{B}^n\ (n > 0)$

$$\mathtt{lex}_n : \mathcal{B}^n \times \mathcal{B}^n \to \{\text{“}<_{lex}\text{”}, \text{“}=_{lex}\text{”}, \text{“}>_{lex}\text{”}\} :$$

$$(a,b) \mapsto \begin{cases} \mathtt{if}\ \mathtt{h}(a) < \mathtt{h}(b)\ \mathtt{then}\ \text{“}<_{lex}\text{”} \\ \mathtt{elif}\ \mathtt{h}(a) > \mathtt{h}(b)\ \mathtt{then}\ \text{“}>_{lex}\text{”} \\ \mathtt{else} \begin{cases} \mathtt{if}\ n = 1\ \mathtt{then}\ \text{“}=_{lex}\text{”} \\ \mathtt{else}\ \mathtt{lex}_{n-1}(\mathtt{t}(a), \mathtt{t}(b)) \end{cases} \end{cases}$$

- lexicografische Ordnung und "natürliche" Ordnung auf $\mathbb{N}$:
  - die Abbildung $\mathtt{val}$ definiert eine Bijektion zwischen $\mathcal{B}^n$ und $[0 \ldots 2^n - 1]$ mit

$$a <_{lex} b \quad \Leftrightarrow \quad \mathtt{val}(a) < \mathtt{val}(b)$$

  (und entsprechend für $=_{lex}$ und $>_{lex}$)

Kosten des lexicografischen Grössenvergleichs $\mathtt{lex}_n$

$$V_n : \mathcal{B}^n \times \mathcal{B}^n \to \mathbb{N} : (a,b) \mapsto \begin{array}{l}\text{Anzahl der Vergleichsoperationen} \\ \underline{\text{auf bit-Ebene}} \\ \text{für die Berechnung von } \mathtt{lex}_n(a,b)\end{array}$$

- im *best case*

$$\min_{(a,b) \in \mathcal{B}^n \times \mathcal{B}^n} V_n(a,b) = 1$$

- im *worst case*

$$\max_{(a,b) \in \mathcal{B}^n \times \mathcal{B}^n} V_n(a,b) = n$$

- im *average case*

$$\overline{V_n} = \frac{1}{\sharp(\mathcal{B}^n \times \mathcal{B}^n)} \sum_{(a,b) \in \mathcal{B}^n \times \mathcal{B}^n} V_n(a,b) = \quad ?$$

rekursives Verhalten:

$$V_1(a,b) = 1 \qquad (a,b) \in \mathcal{B} \times \mathcal{B}$$

$$V_{n+1}(a,b) = \begin{cases} 1 & \text{falls } \mathtt{h}(a) \neq \mathtt{h}(b) \\ 1 + V_n(\mathtt{t}(a), \mathtt{t}(b)) & \text{falls } \mathtt{h}(a) = \mathtt{h}(b) \end{cases} \qquad (a,b) \in \mathcal{B}^{n+1} \times \mathcal{B}^{n+1}$$

Folgerung: für $\Sigma V_n = \sum_{(a,b) \in \mathcal{B}^n \times \mathcal{B}^n} V_n(a,b)$ gilt

$$\begin{aligned} \Sigma V_{n+1} &= 2 \cdot \sum_{(a,b) \in \mathcal{B}^n \times \mathcal{B}^n} 1 + 2 \cdot \sum_{(a,b) \in \mathcal{B}^n \times \mathcal{B}^n} (1 + V_n(a,b)) \\ &= 2 \cdot (\sharp \mathcal{B}^n)^2 + 2 \cdot (\sharp \mathcal{B}^n)^2 + 2 \cdot \Sigma V_n \\ &= 4^{n+1} + 2 \cdot \Sigma V_n \end{aligned}$$

Daher gilt

$$\overline{V_{n+1}} = \frac{\Sigma V_{n+1}}{4^{n+1}} = 1 + \frac{1}{2} \frac{\Sigma V_n}{4^n} = 1 + \frac{1}{2} \overline{V_n}\ , \quad \overline{V_1} = 1$$

und somit

$$\overline{V_n} = 2 - \frac{1}{2^{n-1}}, \quad \text{d.h. } \overline{V_n} \sim 2$$

Maximumsuche in einer Liste:
Analyse des Algorithmus maxfind

Problem: In Liste $A[1..n]$ der Länge n, mit $A[i] \in \mathbb{Z}$ $(1 \leq i \leq n)$ das maximale Element mittels paarweiser Vergleiche bestimmen.

(Natur der Listenelemente ist irrelevant: es könnte sich genauso gut um Elemente irgendeiner total geordneten Menge handeln.)

Algorithmische Idee: Liste von links nach rechts durchlaufen, jeweils das grösste der bisher gesehenen Elemente mit dem nächsten vergleichen

Maple-Programm

```
maxfind := proc (A:list(integer))
local length, maxim;
length := nops(A);
if length = 0 then RETURN(NIL) fi;
maxim := A[1];
for j from 2 to length do
   if A[j] > maxim then maxim := A[j] fi; od;
RETURN(maxim);
end;
```

relevante Kostenmaße

- Anzahl der Vergleichsoperationen

  `if A[j] > maxim`

- Anzahl der Wertzuweisungen

  `maxim := A[j]`

Beispiel

$A[i]$	33	12	57	61	44	28	61	72	49	93	12	66
`maxim`	33	33	57	61	61	61	61	72	72	93	93	93

Offensichtlich:

- minimale Anzahl der Vergleichsoperationen die auf einem input $A[1..n]$ ist nur von der Länge n von $A[1..n]$ abhängig und beträgt $n-1$.
- Anzahl der Wertzuweisungen ist dagegen auch von der relativen Grösse (der Anordnung) der Listenelemente anhängig.

  Wertzuweisung `maxim := A[j]` wird genau für diejenigen Werte j ausgeführt, für die $A[j]$ grösser als alle $A[i]$ mit $1 \leq i < j$ ist.

Definition: $lrm(A) =$ Anzahl der Links-Rechts-Maxima von A:

$$lrm(A) := \sharp\{1 \leq i \leq length(A)\ ;\ \forall j < i : A[j] < A[i]\}$$

Beispiel

$$lrm([\underline{33}, 12, \underline{57}, \underline{61}, 44, 28, 61, \underline{72}, 49, \underline{93}, 12, 66]) = 5$$

Die (Zeit-)Komplexität von maxfind kann man (uniform) ansetzen als

$$T_{\texttt{maxfind}}(A) = c_0 + c_1 \cdot length(A) + c_2 \cdot lrm(A)$$

mit "Implementierungskonstanten" c_0, c_1, c_2

Das schlechteste, beste und das mittlere Verhalten von maxfind auf Listen gegebener Länge n:

$$\begin{aligned} T^{\max}_{\texttt{maxfind}}(n) &:= \max\{T_{\texttt{maxfind}}(A)\ ;\ length(A) = n\} \\ T^{\min}_{\texttt{maxfind}}(n) &:= \min\{T_{\texttt{maxfind}}(A)\ ;\ length(A) = n\} \\ T^{\text{aver}}_{\texttt{maxfind}}(n) &:= \text{“Mittelwert”}\{T_{\texttt{maxfind}}(A)\ ;\ length(A) = n\} \end{aligned}$$

- worst case

$$A[1] < A[2] < A[3] < \cdots < A[n] \quad \text{mit} \quad n-1 \text{ Wertzuweisungen}$$

- best case

$$A[1] = \max\{A[i]\ ;\ 1 \le i \le n\} \quad \text{mit} \quad 1 \text{ Wertzuweisung}$$

Also

$$\begin{aligned} T^{\max}_{\texttt{maxfind}}(n) &= c_0 + c_1 \cdot n + c_2 \cdot (n-1) \\ T^{\min}_{\texttt{maxfind}}(n) &= c_0 + c_1 \cdot n + c_2 \end{aligned}$$

Aber was ist $T^{\text{aver}}_{\texttt{maxfind}}(n)$?

Antwort auf Frage nach dem mittleren Verhalten bei inputs der Länge n benötigt “stochastisches Modell”, d.h. eine Annahme darüber, mit welcher Wahrscheinlichkeit $p_{n,k}$ das Ereignis $lrm(A) = k$ bei (den unendlich vielen verschiedenen) inputs der Länge n auftritt.

Permutationsmodell:

bei Listen der Länge n sind alle $n!$ verschiedenen relativen linearen Ordnungen (Permutationen) der Listenelemente gleichwahrscheinlich.

$\mathfrak{S}_n$: Menge der Permutationen der n Elemente $[1..n] = \{1, 2, \ldots, n\}$

Jede Permutation $\sigma \in \mathfrak{S}_n$ wird identifiziert mit der Liste

$$[\sigma(1), \sigma(2), \ldots, \sigma(n)] \quad \text{d.h. mit} \quad \sigma = \sigma_1\sigma_2\ldots\sigma_n$$

Es sei nun

$$s_{n,k} := \text{Anzahl der Permutationen } \sigma \in \mathfrak{S}_n \text{ mit } lrm(\sigma) = k \quad (1 \le k \le n)$$

Permutationsmodell quantitativ

$$p_{n,k} = \frac{1}{n!}\, s_{n,k} \quad (1 \le k \le n)$$

Beispiel $n = 3$ und $n = 4$

$\mathfrak{S}_3$		$\mathfrak{S}_4$							
σ	$\texttt{lrm}(\sigma)$	σ	$\texttt{lrm}(\sigma)$	σ	$\texttt{lrm}(\sigma)$	σ	$\texttt{lrm}(\sigma)$	σ	$\texttt{lrm}(\sigma)$
123	3	1234	4	1243	3	1342	3	2341	3
213	2	2134	3	2143	2	3142	2	3241	2
132	2	1324	3	1423	2	1432	2	2431	2
312	1	3124	2	4123	1	4132	1	4231	1
231	2	2314	3	2413	2	3412	2	3421	2
321	1	3214	2	4213	1	4312	1	4321	1

Tabelle der Werte $s_{n,k}$ und $lrm(n) = \sum_k k \cdot s_{n,k}$ für $0 \le k \le n \le 6$

$n\backslash k$	0	1	2	3	4	5	6	$lrm(n)$	$lrm(n)/n!$
0	1							0	0
1	0	1						1	1
2	0	1	1					3	$\frac{3}{2} = 1.5$
3	0	2	3	1				11	$\frac{11}{6} = 1.8333\ldots$
4	0	6	11	6	1			50	$\frac{50}{24} = 2.08333\ldots$
5	0	24	50	35	10	1		274	$\frac{274}{120} = 2.28333\ldots$
6	0	120	274	225	85	15	1	1764	$\frac{1764}{720} = 2.45$

Im Permutationsmodell ist

$$T^{\texttt{aver}}_{\texttt{maxfind}}(n) = c_0 + c_1 \cdot n + c_2 \cdot \frac{lrm(n)}{n!}$$

wobei

$$lrm(n) = \sum_{\sigma \in \mathfrak{S}_n} lrm(\sigma) = \sum_{k=1}^{n} k \cdot s_{n,k}$$

Der Vergleich der Beispiele $n = 3$ und $n = 4$ und die Tabelle suggerieren

$$lrm(n+1) = (n+1) \cdot lrm(n) + n!$$

d.h.

$$\frac{lrm(n+1)}{(n+1)!} = \frac{lrm(n)}{n!} + \frac{1}{n+1}$$

und damit (Induktion)

$$\frac{lrm(n)}{n!} = H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Das gilt in der Tat !

Damit ist der Mittelwert $\mathbf{E}(lrm_n)$ der Wahrscheinlichkeitsverteilung $lrm_n = (p_{n,k})_{1 \le k \le n}$ bestimmt:

$$\mathbf{E}(lrm_n) = \sum_k k \cdot p_{n,k} = \frac{1}{n!} \sum_k k \cdot s_{n,k} = H_n$$

Mit einem systematischen Ansatz lassen sich auch andere Parameter, wie etwa die Varianz, dieser WV leicht bestimmen.

$$\mathbf{E}(lrm_n^2) - \mathbf{E}(lrm_n)^2 = \frac{1}{n!} \sum_k k^2 \cdot s_{n,k} - \left(\frac{1}{n!} \sum_k k \cdot s_{n,k} \right)^2$$

Dazu muss man sich die Zahlen $s_{n,k}$ ansehen.

Diese Zahlen $s_{n,k}$ sind in der mathematischen Literatur seit langem bekannt als die (vorzeichenlosen) STIRLING- Zahlen erster Art — siehe etwa das Buch Concrete Mathematics von GRAHAM/KNUTH/PATASHNIK.

Sie genügen der Rekursion

$$\begin{aligned} s_{0,0} &= 1 \\ s_{n,k} &= 0 \quad \text{für } k \le 0 < n \text{ und } 0 \le n < k \\ s_{n,k} &= s_{n-1,k-1} + (n-1) \cdot s_{n-1,k} \quad \text{für } 0 \le k \le n \end{aligned}$$

Diese folgt leicht aus der kombinatorischen Bedeutung.

Wichtig: die Zahlen $s_{n,k}$ haben auch folgende Bedeutung

> $s_{n,k}$ ist die Anzahl der Permutationen aus $\mathfrak{S}_n$, die genau k Zyklen in ihrer Zyklenzerlegung (Darstellung als Produkt von elementfremden Zyklen) haben.

Zur genauen Analyse betrachtet man

$$s_n(x) := \sum_{k=0}^{n} s_{n,k}\, x^k$$

Beispielsweise ist

$$s_4(x) = 6x + 11x^2 + 6x^3 + x^4$$

Das Interesse an diesen Polynomen rührt her von der Beziehung

$$\sum_{k=0}^{n} k \cdot s_{n,k} = \left[\frac{d}{dx}\, s_n(x)\right]_{x=1}$$

Die Polynome $s_n(x)$ genügen nun einer sehr einfachen rekursiven Beziehung (äquivalent zu der obigen Rekursion für die $s_{n,k}$)

$$\begin{aligned} s_0(x) &= 1 \\ s_n(x) &= (x+n-1)\cdot s_{n-1}(x) \quad (n>0) \end{aligned}$$

Daraus per Induktion

$$s_n(x) = \prod_{k=0}^{n-1}(x+k) \qquad (n \geq 0)$$

Berechnung des Erwartungswertes für lrm_n

$$\begin{aligned} \sum_{k=1}^{n} k \cdot s_{n,k} &= \left[\frac{d}{dx}\, s_n(x)\right]_{x=1} \\ &= \sum_{k=0}^{n-1} \left[\frac{s_n(x)}{x+k}\right]_{x=1} \\ &= \sum_{k=0}^{n-1} \frac{n!}{1+k} = n! \cdot H_n \end{aligned}$$

wobei

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

die n-te harmonische Zahl ist (das Anfangsstück der ersten n Summanden der divergierenden harmonischen Reihe).

Also

$$\mathbf{E}(lrm_n) = H_n$$

Die Varianz kann man ganz entsprechend ermitteln.

Zunächst ist

$$\begin{aligned} \sum_{k} k^2 \cdot s_{n,k} &= \left[\left(\frac{d}{dx}\right)^2 s_n(x)\right]_{x=1} + \left[\frac{d}{dx} s_n(x)\right]_{x=1} \\ &= n! \cdot \left(H_n^2 - H_n^{(2)}\right) + n! \cdot H_n \end{aligned}$$

wobei die $H_n^{(2)}$ die harmonischen Zahlen zweiter Ordnung sind:

$$H_n^{(2)} := \sum_{k=1}^{n} \frac{1}{k^2} = 1 + \frac{1}{4} + \frac{1}{9} + \cdot + \frac{1}{n^2}$$

Mit dieser Bezeichnungsweise ergibt sich also

$$\begin{aligned} \mathbf{E}(lrm_n^2) - \mathbf{E}(lrm_n)^2 &= H_n^2 - H_n^{(2)} + H_n - H_n^2 \\ &= H_n - H_n^{(2)} \end{aligned}$$

Es fehlt nun nur noch eine Information über das Wachstumsverhalten der harmonischen Zahlen, um diese Analyse des (mittleren) Laufzeitverhaltens von maxfind abzuschließen.

Ein Vergleich mit dem Graphen der Logarithmusfunktion zeigt

$$\ln(n) < H_n < \ln(n) + 1 \quad \text{für } n > 1$$

Der Grenzwert

$$\lim_{n\to\infty} (H_n - \ln(n)) = \gamma = 0.5772156649\ldots$$

ist die Eulersche Konstante.

Eine gute Approximation ist

$$H_n = \ln(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{\epsilon_n}{120n^4} \quad \text{mit} \quad 0 < \epsilon_n < 1$$

Während die unendliche Reihe $\sum_n \frac{1}{n}$ divergiert, konvergiert die Folge der $H_n^{(2)}$:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} = 1.644034068\ldots$$

Alles weitere Wissenswerte über die harmonischen Zahlen erfährt man aus Kapitel 6.3 des Buches Concrete Mathematics von Graham/Knuth/Patashnik.

Permutationen und Inversionen

- $L = [L_1, L_2, \ldots, L_n]$ Liste von Elementen einer totalgeordneten Menge
- Indexpaar (i, j) mit $1 \le i < j \le n$ ist *Inversion* von L, falls $L_i > L_j$
- Beispiel: $[101, 115, 30, 63, 47, 20]$ hat 12 Inversionen:

  $$(1,3),(1,4),(1,5),(1,6),(2,3),(2,4),(2,5),(2,6),(3,6),(4,5),(4,6),(5,6)$$

- $\mathtt{inv}(L)$ =Anzahl der Inversionen von L
  ist ein Maß für die "(Un)Sortiertheit" der Liste L
  - $\mathtt{inv}([1, 2, 3, \ldots, n-1, n]) = 0$
  - $\mathtt{inv}([n, n-1, n-2, \ldots, 2, 1]) = \binom{n}{2}$
  - wie groß ist $\mathtt{inv}(\sigma)$ für $\sigma \in \mathfrak{S}_n$ im Mittel?

Inversionsvektoren

$$\underbrace{\sigma = [\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n]}_{\in \mathfrak{S}_n} \quad \mapsto \quad \underbrace{\ell(\sigma) = [\ell_1(\sigma), \ell_2(\sigma), \ell_3(\sigma), \ldots, \ell_n(\sigma)]}_{\ell_j(\sigma) = \sharp\{i < j\,;\, \sigma_i > \sigma_j\}}$$

Beispiel:

$$\ell([5, 3, 2, 4, 6, 1]) = [0, 1, 2, 1, 0, 5]$$

- $\ell(\sigma) = (\ell_1(\sigma), \ldots, \ell_n(\sigma)$ ist der *Inversionsvektor* ("Lehmer-Code") von σ
- es gilt $0 \le \ell_j(\sigma) < j$ $(1 \le j \le n)$
- $\mathcal{L}_n := \{(\ell_1, \ell_2, \ldots, \ell_n)\,;\, 0 \le \ell_j(\sigma) < j \ (1 \le j \le n)\}$
- $\mathcal{L}_n = [0..0] \times [0..1] \times [0..2] \times \cdots \times [0..n-1], \qquad \sharp\mathcal{L}_n = n!$
- $\sigma \mapsto \ell(\sigma)$ ist eine Bijektion zwischen $\mathfrak{S}_n$ und $\mathcal{L}_n$
  Wie sieht die Umkehrabbildung aus?
- $\mathtt{inv}(\sigma) = \|\ell(\sigma)\| = \ell_1(\sigma) + \cdots + \ell_n(\sigma), \quad \mathtt{lrm}(\sigma) = \sharp_0 \ell(\sigma)$

$n = 1$

σ	$\ell(\sigma)$	$\mathtt{inv}(\sigma)$	$\mathtt{lrm}(\sigma)$
1	0	0	1

$n = 2$

σ	$\ell(\sigma)$	$\mathtt{inv}(\sigma)$	$\mathtt{lrm}(\sigma)$
1 2	0 0	0	2
2 1	0 1	1	1

$n = 3$

σ	$\ell(\sigma)$	$\mathtt{inv}(\sigma)$	$\mathtt{lrm}(\sigma)$
1 2 3	0 0 0	0	3
1 3 2	0 0 1	1	2
2 1 3	0 1 0	1	2
2 3 1	0 0 2	2	2
3 1 2	0 1 1	2	1
3 2 1	0 1 2	3	1

$n = 4$

σ	$\ell(\sigma)$	$\mathtt{inv}(\sigma)$	$\mathtt{lrm}(\sigma)$	σ	$\ell(\sigma)$	$\mathtt{inv}(\sigma)$	$\mathtt{lrm}(\sigma)$
1 2 3 4	0 0 0 0	0	4	3 1 2 4	0 1 1 0	2	2
1 2 4 3	0 0 0 1	1	3	3 1 4 2	0 1 0 2	3	2
1 3 2 4	0 0 1 0	1	3	3 2 1 4	0 1 2 0	3	2
1 3 4 2	0 0 0 2	2	3	3 2 4 1	0 1 0 3	4	2
1 4 2 3	0 0 1 1	2	2	3 4 1 2	0 0 2 2	4	2
1 4 3 2	0 0 1 2	3	2	3 4 2 1	0 0 2 3	5	2
2 1 3 4	0 1 0 0	1	3	4 1 2 3	0 1 1 1	3	1
2 1 4 3	0 1 0 1	2	2	4 1 3 2	0 1 1 2	4	1
2 3 1 4	0 0 2 0	2	3	4 2 1 3	0 1 2 1	4	1
2 3 4 1	0 0 0 3	3	3	4 2 3 1	0 1 1 3	5	1
2 4 1 3	0 0 2 2	4	2	4 3 1 2	0 1 2 2	5	1
2 4 3 1	0 0 1 3	4	2	4 3 2 1	0 1 2 3	6	1

Die Statistiken `lrm` und `inv` für Permutationen

– Häufigkeiten

$$s_{n,k} = \sharp\{\sigma \in \mathfrak{S}_n \,;\, \mathtt{lrm}(\sigma) = k\} = \sharp\{\ell \in \mathcal{L}_n \,;\, \sharp_0\ell = k\} \quad (1 \le k \le n)$$

$$i_{n,k} = \sharp\{\sigma \in \mathfrak{S}_n \,;\, \mathtt{inv}(\sigma) = k\} = \sharp\{\ell \in \mathcal{L}_n \,;\, \|\ell\| = k\} \quad (0 \le k \le \binom{n}{2})$$

– "erzeugende Polynome" für die Statistiken

$$s_n(x) = \sum_{\sigma\in\mathfrak{S}_n} x^{\mathtt{lrm}(\sigma)} = \sum_{\ell\in\mathcal{L}_n} x^{\sharp_0\ell} = \sum_{k=1}^{n} s_{n,k}\, x^k$$

$$i_n(x) = \sum_{\sigma\in\mathfrak{S}_n} x^{\mathtt{inv}(\sigma)} = \sum_{\ell\in\mathcal{L}_n} x^{\|\ell\|} = \sum_{k=0}^{\binom{n}{2}} i_{n,k}\, x^k$$

— Beispiele

$$s_4(x) = 6x + 11x^2 + 6x^3 + x^4$$
$$i_4(x) = 1 + 3x + 5x^2 + 6x^3 + 5x^4 + 3x^5 + x^6$$

– rekursive Struktur von $\mathcal{L}_n$: $\mathcal{L}_{n+1} = \mathcal{L}_n \times [0..n]$

– rekursives Verhalten von `lrm` und `inv`

$$\sharp_0(\ell_1,\ell_2,\ldots,\ell_n,\ell_{n+1}) = \sharp_0(\ell_1,\ell_2,\ldots,\ell_n) + \delta_{\ell_{n+1},0} \qquad \sharp_0(\epsilon) = 0$$
$$\|(\ell_1,\ell_2,\ldots,\ell_n,\ell_{n+1})\| = \|(\ell_1,\ell_2,\ldots,\ell_n)\| + \ell_{n+1} \qquad \|\epsilon\| = 0$$

– Rekursion der Polynome

$$s_{n+1}(x) = s_n(x)\cdot(x+n) \qquad s_0(x) = 1$$
$$i_{n+1}(x) = i_n(x)\cdot(1 + x + x^2 + \cdots + x^n) \qquad i_0(x) = 1$$

– Produktdarstellung der Polynome

$$s_n(x) = x(x+1)(x+2)\cdots(x+n-1) \qquad (n \ge 0)$$
$$i_n(x) = (1+x)(1+x+x^2)(1+x+x^2+x^3)\cdots(1+x+\cdots+x^{n-1}) \qquad (n \ge 0)$$

Beweise für die Rekursionsformeln

$$\begin{aligned}
s_{n+1}(x) &= \sum_{(\ell_1,\ldots,\ell_{n+1})\in\mathcal{L}_{n+1}} x^{\sharp_0(\ell_1,\ldots,\ell_{n+1})} \\
&= \sum_{(\ell_1,\ldots,\ell_n)\in\mathcal{L}_n,\ell_{n+1}\in[0..n]} x^{\sharp_0(\ell_1,\ldots,\ell_n)+\delta_{\ell_{n+1},0}} \\
&= \sum_{(\ell_1,\ldots,\ell_n)\in\mathcal{L}_n} x^{\sharp_0(\ell_1,\ldots,\ell_n)} \cdot \sum_{a\in[0..n]} x^{\delta_{a,0}} \\
&= s_n(x)\cdot(x+n)
\end{aligned}$$

$$\begin{aligned}
i_{n+1}(x) &= \sum_{(\ell_1,\ldots,\ell_{n+1})\in\mathcal{L}_{n+1}} x^{\|(\ell_1,\ldots,\ell_{n+1})\|} \\
&= \sum_{(\ell_1,\ldots,\ell_n)\in\mathcal{L}_n,\ell_{n+1}\in[0..n]} x^{\|(\ell_1,\ldots,\ell_n)\|+\ell_{n+1}} \\
&= \sum_{(\ell_1,\ldots,\ell_n)\in\mathcal{L}_n} x^{\|(\ell_1,\ldots,\ell_n)\|} \cdot \sum_{a\in[0..n]} x^{a} \\
&= i_n(x)\cdot(1 + x + x^2 + \cdots + x^n)
\end{aligned}$$

Erwartungswert (Mittelwert) für `lrm` auf $\mathfrak{S}_n$

- gesucht:

$$\frac{1}{n!}\sum_k k \cdot s_{n,k} = \frac{s'(1)}{s(1)}$$

- aus Rekursion: $s'_{n+1}(1) = s'_n(1)\cdot(1+n) + n!$, also

$$\frac{s'_{n+1}(1)}{s_{n+1}(1)} = \frac{s'_n(1)}{s_n(1)} + \frac{1}{n+1}$$

- Folgerung:

$$\frac{s'_n(1)}{s_n(1)} = \sum_{k=1}^{n} \frac{1}{k} = H_n$$

Erwartungswert (Mittelwert) für `inv` auf $\mathfrak{S}_n$

- gesucht:

$$\frac{1}{n!}\sum_k k \cdot i_{n,k} = \frac{i'(1)}{i(1)}$$

- aus Rekursion: $i'_{n+1}(1) = i'_n(1)\cdot(1+n) + n!\cdot(1+2+\cdots+n)$, also

$$\frac{i'_{n+1}(1)}{i_{n+1}(1)} = \frac{i'_n(1)}{i_n(1)} + \frac{1}{n+1}\frac{n(n+1)}{2}$$

- Folgerung:

$$\frac{i'_n(1)}{i_n(1)} = \sum_{k=1}^{n-1} \frac{k}{2} = \frac{n(n-1)}{4}$$

Sortieren

ist eines der am häufigsten auftauchenden und am besten untersuchten Proleme in der Informatik (GA)

Literaturempfehlungen:

- D. E. Knuth
  The Art of Computer Programming vol. 3 Sorting and Searching (2nd. ed.)
  Addison-Wesley 1998
- R. Sedgewick
  Algorithms (diverse Ausgaben)
  Addison-Wesley (ab 1983)
- Jon Bentley
  Programming Pearls (2nd. ed.)
  Addison Wesley 1999

Zitat Knuth (Aus TAOCP vol. 3, Einleitung):

Computer manufacturers in the 1960s estimated that more than 25 percent of the running time of computer were spent on sorting, when all their customer were taken into account. In fact, there were many installations in which the taks of sorting was responsible for more than half of the computing time.

From these statistics we may conclude that either

(i) there are many important applications of sorting, or

(ii) many people sort when they shouldn't, or

(iii) inefficient sorting algorithms have been in common use.

The real truth probably involves all three of these possibilities, but in any event we can see that sorting is worthy of serious study, as a practical matter.

Even if sorting were almost useless, there would be plenty of rewarding reasons for studying it anyway! The ingenious algorithms that have been discovered show that sorting is an extremely interesting topic to explore in its own right. Many fascinating unsolved problems remain in this area, as well as quite a few solved ones.

From a broader perspective we will find that sorting algorithms make a valuable case study on how to attack computer programming problems in general...

Wichtige Anwendungen des Sortierens (Knuth)

- Solving the "togetherness" problem
  (siehe z.B. das von Bentley diskutierte "Anagram-Problem")
- Matching items in two or more files
- Seaching for information by key values

Aber auch

- Sorting techniques provide excellent illustrations of the general ideas involved in the analysis of algorithms — the ideas used to determine performance characteristics of algorithms so that an intelligent choice can be made between competing methods.

Beispiel aus den *Programming Pearls* von Bentley

Vergleich von drei Sortierverfahren in fünf Versionen

Programm	# Zeilen	Laufzeit	Zeit (s) für Filegrösse		
	C-code	(μs)	100	1000	10000
Insertion Sort 1	5	17 n^2	0.17	17.3	1730
Insertion Sort 2	7	6 n^2	0.06	5.7	570
Quicksort 1	11	63 $n \log n$	0.05	0.63	7.8
Quicksort 2	20	32 $n \log n$	0.03	0.32	4.3
System Sort	1	97 $n \log n$	0.06	1.0	13.6

Das Sortierproblem

- gegeben: Liste $[D_1, D_2, \ldots, D_n]$ von Datensätzen
  - jeder Datensatz D_i hat Schlüssel k_i
  - auf der Menge aller möglichen Schlüssel totale Ordnung $\leq$
- gesucht: Permutation $\pi \in \mathfrak{S}_n$
  ($\mathfrak{S}_n$ = Menge der Permutationen von $\{1, 2, \ldots, n\}$) mit

  $$k_{\pi^{-1}(1)} \leq \cdots \leq k_{\pi^{-1}(n)} \quad \text{(aufsteigend sortiert)}$$

  oder

  $$k_{\pi^{-1}(1)} \geq \cdots \geq k_{\pi^{-1}(n)} \quad \text{(absteigend sortiert)}$$

- stabile Sortierung leistet zusätzlich

  $$(k_i = k_j \wedge i < j) \;\Rightarrow\; \pi(i) < \pi(j)$$

Beispiel

$$\begin{array}{rcl} k & = & [\ 55\ \ 41\ \ 59\ \ 26\ \ 53\ \ 58\ \ 97\ \ 93\ ] \\ & & \Downarrow \\ k_{\pi^{-1}} & = & [\ 26\ \ 41\ \ 53\ \ 55\ \ 58\ \ 59\ \ 93\ \ 97\ ] \end{array}$$

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 2 & 6 & 1 & 3 & 5 & 8 & 7 \end{pmatrix}$$

$$\pi^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 2 & 5 & 1 & 6 & 3 & 8 & 7 \end{pmatrix}$$

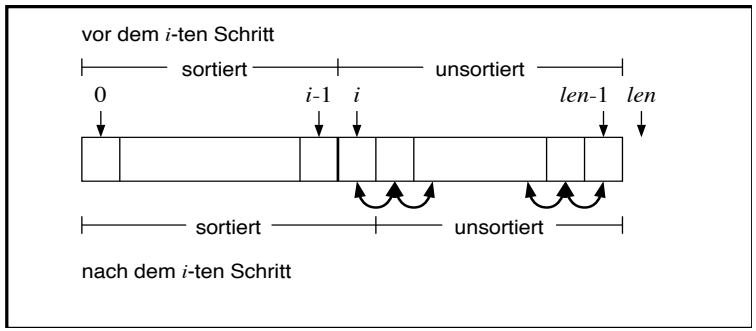
Bemerkungen zum Sortierproblem

- Vergleichsfunktion `compare` auf Schlüsseln mit

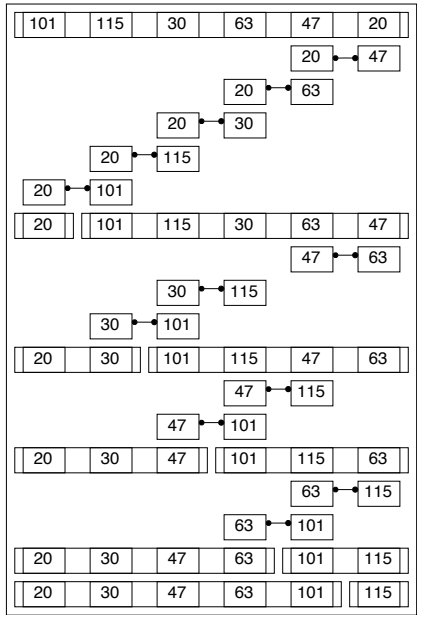
  $$\texttt{compare}(x, y) = \begin{cases} -1 & \text{falls } x < y \\ 0 & \text{falls } x = y \\ 1 & \text{falls } x > y \end{cases}$$

  wird implizit als gegeben angenommen
- unterscheide Zugriffsmodi: sequentiell (lineare Liste) vs. wahlfrei (Feld)
- operiere nur mit Referenzen (Zeigern), nie mit Datensätzen
- Annahme im folgenden: Daten bestehen nur aus Schlüsseln, diese sind natürliche Zahlen
- relevante Grössen für Komplexität sind: Anzahl der Vergleichsoperationen (meist), Anzahl der Zuweisungen, zusätzlicher Speicherplatz
- implizite Annahme: Sortieren ohne externen Speicher

Das Prinzip von "Sortieren durch Auswählen" (Selectionsort):



Iterierte Auswahl des kleinsten Elements im unsortierten Bereich und Anfügen an den schon sortierten Bereich



Sortieren durch Auswählen — Version für Listen (gofer)

```
selectionSort :: [Int] -> [Int]
selectionSort [] = []
selectionSort ul = let (m,l)=select(ul)
                   in m:selectionSort(l)

select :: [Int] -> (Int,[Int])
select [e]    = (e,[])
select (u:ul) = let (m,l)=select ul
                in if (u<=m) then (u,m:l)
                             else (m,u:l)
```


Sortieren durch Auswählen — Version für Felder (C)

```
SelectionSort (int array[], int len)
{
    for (int i = 0; i < len; i++)
        for (int j = len − 1; j > i; j−−)
            if (array[j − 1] > array[j])
                swap(array[j − 1], array[j]);
}

swap(int& a, int& b)
{
    int temp = a;    a = b;    b = temp;
}
```

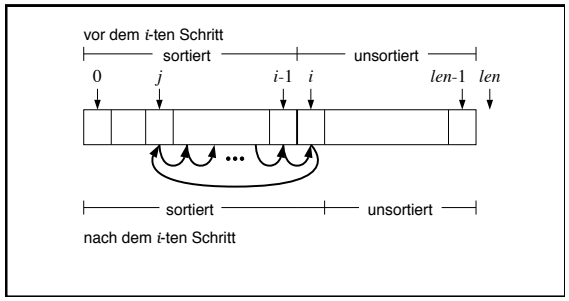
Analyse von Selectionsort

- hier: Komplexität = Anzahl der Vergleichsoperationen $V_{sel}(n)$
- Bestimmung des kleinsten (grössten) Elements in einer Liste der Länge n erfordert $n-1$ Vergleichoperationen
- Bestimmung des Elements in Position i erfordert Minimumsuche in einer Liste von $n-i$ Elementen $\Rightarrow n-i-1$ Vergleiche $(0 \leq i < n)$
- Gesamtaufwand

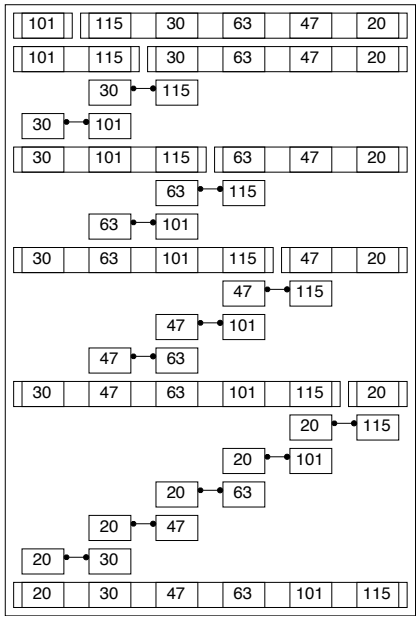
$$V_{sel}(n) = \sum_{i=0}^{n-1} n-i-1 = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = \binom{n}{2} \in O(n^2)$$

- Selectionsort benötigt zum Sortieren einer Liste von n Element maximal $\binom{n}{2}$, also $O(n^2)$ Vergleiche (GA Theorem 2.1)

Das Prinzip von "Sortieren durch Einfügen" (Insertionsort)



Iteriert Element aus dem unsortierten Bereich in den sortierten Bereich an der richtigen Stelle einfügen



Sortieren durch Einfügen — Version für Listen (`gofer`)

```
insertionSort :: [Int]->[Int]
insertionSort []     = []
insertionSort (e:us) = insert e (insertionSort us)

insert :: Int -> [Int] -> [Int]
insert e []     = [e]
insert e (s:sl) | e>s  = s:(insert e sl)
                | e<=s = e:(s:sl)
```

Nächste Folie: Sortieren durch Einfügen — Version für Felder (`C`)

```
InsertionSort (int array[], int len)
{
    for (int i = 1; i < len; i++)
        right_shift(array, LinearSearch(array[], i, array[i]), i);
}

int LinearSearch(int array[], int len, int elt)
{
    int j = 0;
    while (j < len && array[j] ≤ elt) j++;
    return j;
}

right_shift(int array[], int ℓ, int r)
{
    int k = array[r];
    for (int j = r; j > ℓ; j--) array[j] = array[j − 1];
    array[ℓ] = k;
}
```


Analyse von Insertionsort

- hier: Komplexität = Anzahl der Vergleichsoperationen $V_{ins}(n)$
- Einfügen eines Elementes in eine sortierte Liste von m Elementen an der richtigen Position erfordert maximal m Vergleiche
- Für das Einfügen des Elementes in Position i genügen i Vergleiche $(0 \leq i < n)$
- Gesamtaufwand

$$V_{ins}(n) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = \binom{n}{2} \in O(n^2)$$

- Insertionsort benötigt zum Sortieren einer Liste von n Elementen maximal $\binom{n}{2}$, also $O(n^2)$ Vergleiche (GA Theorem 2.2)

Verbesserung von Insertionsort durch binäre Suche

In einem sortierten (!) Feld der Länge n kann man mittels sukzessivem Halbieren des Suchintervalls mit maximal $\lceil \log(n+1) \rceil$ Vergleichen die korrekte Position eines Elementes feststellen (GA Lemma 2.3)

```
BinarySearch (int array[], int len, int elt)
{   /* search in [ℓ : r − 1] */
    int ℓ = 0, r = len;
    for (int pos = (ℓ + r)/2; ℓ < r; pos = (ℓ + r)/2)
        if (elt ≥ array[pos]) ℓ = pos + 1;
        else r = pos;
    return ℓ;
}
```


- Ersetzt man bei Insertionsort den Aufruf von `LinearSearch` durch `BinarySearch`, so erhält man ein Verfahren, das mit maximal $n \cdot \lceil \log(n) \rceil$ Vergleichsoperationen für Listen der Länge n auskommt
- Allerdings, man hat im worst-case und im Mittel $O(n^2)$ Vertauschungs-/Verschiebeoperationen mittels `right_shift` auszuführen!

Genauere Analyse von Selectionsort und Insertionsort:

- Für die Laufzeit spielen nicht nur die Vergleichsoperationen eine Rolle, sondern auch die Vertauschungen!
- Ein Maß für die "(Un)Sortiertheit" einer Liste ist die Anzahl der "Fehlstellungen" (Inversionen), das sind Indexpaare (i, j) mit $1 \leq i < j \leq n$ und $k_i > k_j$
- Im Beispiel: Die Liste

  $$[101, 115, 30, 63, 47, 20]$$

  hat 12 Inversionen:

  $$(1,3),(1,4),(1,5),(1,6),(2,3),(2,4),(2,5),(2,6),(3,6),(4,5),(4,6),(5,6)$$

- Jede Vertauschungsoperation, die in Selectionsort bzw. Insertionsort ausgeführt wird, macht die Anzahl der Inversionen der aktuellen Liste um genau 1 kleiner, d.h.

  Anzahl der Vertauschungen = Anzahl der Inversionen der input-Liste

Folgerung: eine sorgfältige Analyse von Sortieralgorithmen — insbesondere auch dann, wenn es um das average-case-Verhalten geht — muss untersuchen, wie sich die Algorithmen in ihrem Ablauf auf die Anzahl der Inversionen auswirken

Frage: wieviele Inversionen hat eine Liste von n (verschiedenen) Elementen?

- maximal: $\frac{n(n-1)}{2}$
- minimal: 0
- im Mittel (alle $n!$ Permutationen $\sigma \in \mathfrak{S}_n$ gleichwahrscheinlich): $\frac{n(n-1)}{4}$

Selectionsort und Insertionsort sind u.a. auch deshalb ineffiziente Sortieralgorithmen, weil sie im average-case $O(n^2)$ Vertauschungsoperationen erfordern
(was nicht heisst, dass sie gänzlich unbrauchbar wären!)

Ergänzung : Berechnung von Mittelwerte und Varianz der Inversionszahl

- $\mathtt{inv}_n : \mathfrak{S}_n \to [0\,..\,\binom{n}{2}] : \sigma \mapsto$ Anzahl der Inversionen von σ
- $i_{n,k} := \sharp\{\sigma \in \mathfrak{S}_n\,;\, \mathtt{inv}_n(\sigma) = k\} \quad (0 \leq k \leq \binom{n}{2})$
- $I_n(x) := \sum_{\sigma\in\mathfrak{S}_n} x^{\mathtt{inv}_n(\sigma)} = \sum_{k=0}^{\binom{n}{2}} i_{n,k}\, x^k$
- $\mathtt{inv}_n(\sigma_1\sigma_2\ldots\sigma_{n-1}\sigma_n) + \mathtt{inv}_n(\sigma_n\sigma_{n-1}\ldots\sigma_2\sigma_1) = \binom{n}{2}$
- für $\sigma = (\sigma_1\sigma_2\ldots\sigma_n) \in \mathfrak{S}_n$ und $0 \leq k \leq n$:
  $\mathtt{inv}_{n+1}(\sigma_1\sigma_2\ldots\sigma_k\,(n+1)\,\sigma_{k+1}\ldots\sigma_n) = \mathtt{inv}_n(\sigma) + n - k$
- $I_{n+1}(x) = (1 + x + x^2 + \cdots + x^n)\cdot I_n(x)\ (n \geq 1)$, $I_1(x) = 1$
- $I_n(x) = \prod_{j=1}^{n}(1 + x + \cdots + x^{j-1})$

Beispiel $n = 3$ und $n = 4$

$\mathfrak{S}_3$		$\mathfrak{S}_4$							
σ	$\mathtt{inv}(\sigma)$	σ	$\mathtt{inv}(\sigma)$	σ	$\mathtt{inv}(\sigma)$	σ	$\mathtt{inv}(\sigma)$	σ	$\mathtt{inv}(\sigma)$
123	0	1234	0	1243	1	1423	2	4123	3
213	1	2134	1	2143	2	2413	3	4213	4
132	1	1324	1	1342	2	1432	3	4132	4
312	2	3124	2	3142	3	3412	4	4312	5
231	2	2314	2	2341	3	2431	4	4231	5
321	3	3214	3	3241	4	3421	5	4321	6

$$I_3(x) = 1 + 2x + 2x^2 + x^3 = (1+x)(1+x+x^2)$$
$$I_4(x) = 1 + 3x + 5x^2 + 6x^3 + 5x^4 + 3x^5 + x^6 = (1+x+x^2+x^3)\cdot I_3(x)$$
$$= (1+x)(1+x+x^2)(1+x+x^2+x^3)$$

Tabelle der Werte $i_{n,k}$ und $I'_n(1) = \sum_k k \cdot i_{n,k}$ für $0 \le k \le n \le 5$

$n\backslash k$	0	1	2	3	4	5	6	7	8	9	10	$I'_n(1)$	$I'_n(1)/I_n(1)$
0	1											0	$\frac{0}{1} = 0 = \frac{0(0-1)}{4}$
1	1											0	$\frac{0}{1} = 0 = \frac{1(1-0)}{4}$
2	1	1										1	$\frac{1}{2} = 0.5 = \frac{2(2-1)}{4}$
3	1	2	2	1								9	$\frac{9}{6} = 1.5 = \frac{3(3-1)}{4}$
4	1	3	5	6	5	3	1					50	$\frac{72}{24} = 3 = \frac{4(4-1)}{4}$
5	1	4	8	15	20	22	20	15	8	4	1	600	$\frac{600}{120} = 5 = \frac{5(5-1)}{4}$

- Definition: $[k](x) = 1 + x + x^2 + \cdots + x^{k-1} = \dfrac{1-x^k}{1-x}$
- Eigenschaften:
  $[k](1) = k = \binom{k}{1}$
  $[k]'(1) = 1 + 2 + \cdots + (k-1) = \binom{k}{2}$
  $[k]''(1) = \sum_{i=1}^{k} i(i-1) = 2\binom{k}{3}$
- aus $I_n(x) = \prod_{k=1}^{n} [k](x)$ erhält man mittels logarithmischer Ableitung

$$\frac{I'_n(x)}{I_n(x)} = \sum_{k=1}^{n} \frac{[k]'(x)}{[k](x)}$$

$$\frac{I''_n(x)}{I_n(x)} + \frac{I'_n(x)}{I_n(x)} - \left(\frac{I'_n(x)}{I_n(x)}\right)^2 = \sum_{k=1}^{n} \left[\frac{[k]''(x)}{[k](x)} + \frac{[k]'(x)}{[k](x)} - \left(\frac{[k]'(x)}{[k](x)}\right)^2 \right]$$

Folgerungen

- Erwartungswert

$$\begin{aligned}
\boldsymbol{E}(inv_n) &= \frac{1}{n!} \sum_{k=0}^{\binom{n}{2}} k \cdot i_{n,k} \\
&= \frac{I'_n(1)}{I_n(1)} = \sum_{k=1}^{n} \frac{[k]'(1)}{[k](1)} \\
&= \sum_{k=1}^{n} \frac{\binom{k}{2}}{k} = \sum_{k=1}^{n} \frac{k-1}{2} \\
&= \frac{1}{2}\binom{n}{2} = \frac{n(n-1)}{4}
\end{aligned}$$

- Varianz

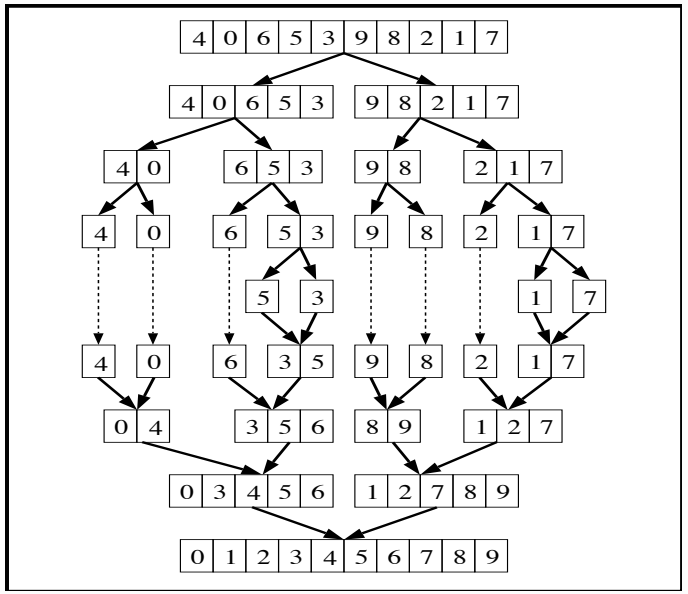
$$\begin{aligned}
\boldsymbol{V}(inv_n) &= \boldsymbol{E}(inv_n^2) - \boldsymbol{E}(inv_n)^2 \\
&= \frac{1}{n!} \sum_{k=0}^{\binom{n}{2}} k^2 \cdot i_{n,k} - \left(\frac{1}{n!} \sum_{k=0}^{\binom{n}{2}} k \cdot i_{n,k} \right)^2 \\
&= \frac{I''_n(1)}{I_n(1)} + \frac{I'_n(1)}{I_n(1)} - \left(\frac{I'_n(1)}{I_n(1)}\right)^2 \\
&= \sum_{k=1}^{n} \left[\frac{[k]''(1)}{[k](1)} + \frac{[k]'(1)}{[k](1)} - \left(\frac{[k]'(1)}{[k](1)}\right)^2 \right] \\
&= \sum_{k=1}^{n} \left[\frac{(k-1)(k-2)}{3} + \frac{k-1}{2} - \left(\frac{k-1}{2}\right)^2 \right] \\
&= \sum_{k=1}^{n} \frac{(k-1)(k+1)}{12} \\
&= \frac{n(n-1)(2n-5)}{72}
\end{aligned}$$

Sortieren durch Mischen (mergesort)

Idee:

- Teile die zu sortierende Liste L in zwei (etwa) gleichgrosse Teile L_1, L_2
- Sortiere diese Teillisten unabhängig voneinander durch rekursive Anwendung der Idee: $L_1 \mapsto \hat{L}_1$, $L_2 \mapsto \hat{L}_2$
  (Verankerung: Listen der Länge 1 sind bereits soriert)
- Konstruiere aus den sortierten Liste $\hat{L}_1$, $\hat{L}_2$ durch Mischen die sortierte Liste $\hat{L}$

Bemerkung: Mischen sortierter Listen braucht zusätzlichen Platz!




```
MergeSort (int a[], int b[], int ℓ, int r)
{
    if (ℓ < r − 1)
    {
        int m = (ℓ + r)/2;
        MergeSort(a, b, ℓ, m);
        MergeSort(a, b, m, r);
        merge(a, b, ℓ, m, r);
        copy(b, a, ℓ, r);
    }
}

merge(int a[], int b[], int ℓ, int m, int r)
{
    for (int i = ℓ, int j = m, int k = ℓ; k < r; k++)
        if ((j ≥ r) || ((i < m) && (a[i] ≤ a[j])))
            b[k] = a[i++];
        else
            b[k] = a[j++];
}

copy(int b[], int a[], int ℓ, int r)
{
    for (int i = ℓ; i < r; i++)
        a[i] = b[i];
}
```


Zur Analyse

- Das Mischen (merging) zweier sortierter Listen der Längen m und n erfordert $n + m - 1$ Vergleichsoperationen
- $V^{\mathsf{rek}}_{\mathsf{merge}}(n)$: maximale Anzahl von Vergleichoperationen für das Sortieren von Listen der Länge n mittels (rekursivem) mergesort
- Rekursionsgleichung

$$V^{\mathsf{rek}}_{\mathsf{merge}}(n) = V^{\mathsf{rek}}_{\mathsf{merge}}(\lceil n/2 \rceil) + V^{\mathsf{rek}}_{\mathsf{merge}}(\lfloor n/2 \rfloor) + n - 1$$
$$V^{\mathsf{rek}}_{\mathsf{merge}}(1) = 0$$

Zur Lösung der Rekursionsgleichung für mergesort (GA Lemma 2.5)

- Definiere $W : [1, \infty) \to \mathbb{R}$ durch

$$W(x) = \begin{cases} 0 & \text{für } x \in [1, 2) \\ 2 \cdot W\left(\frac{x+1}{2}\right) + x - 1 & \text{für } x \in [2, \infty) \end{cases}$$

- Es gilt $V^{\mathsf{rek}}_{\mathsf{merge}}(n) \leq W(n)$
- Es gilt (für $n \in \mathbb{N}$) $W(n) = (n-1)\lceil \log n \rceil$ $(\in \Theta(n \log n))$
- Folgerung (GA, Theorem 2.6): Rekursives mergesort benötigt für das Sortieren eines Feldes der Länge n maximal $n \cdot \lceil \log n \rceil$ Vergleiche
- Man kann die Lösung der Rekursion sogar explizit angeben:

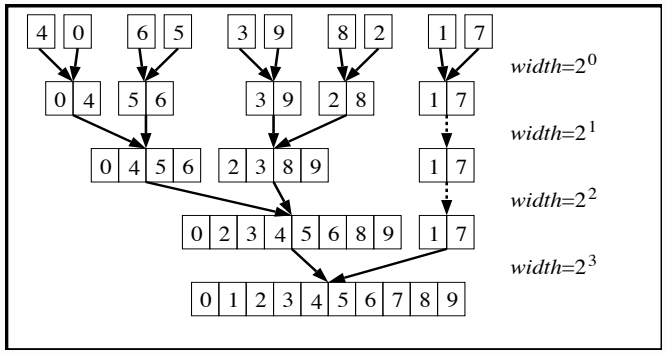
$$V^{\mathsf{rek}}_{\mathsf{merge}}(n) = n \cdot \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1$$

Beweis durch (sorgfältige!) Induktion.

Iterative Variante von mergesort

Idee:

- Bottom-up merging (ohne die top-down rekursive Aufteilung)
- In der i-ten Phase liegen $\lceil n/2^{i-1} \rceil$ sortierte Listen der Länge $width = 2^{i-1}$ (davon eine evtl. mit Länge $< 2^{i-1}$) vor
- In der i-ten Phase werden sortierte Folgen der Länge 2^{i-1} durch merging zu sortierten Listen der Länge $width = 2^i$ verschmolzen
- Nach $\lceil \log n \rceil$ Phasen liegt eine sortierte Liste vor




```
MergeSort (int a[], int b[], int len)
{
    for (int width = 1; width < len; width *= 2)
    {
        for (int i = 0; i < len; i += 2 * width)
        {
            int ℓ = i;
            int m = min(len, i + width);
            int r = min(len, i + 2 * width);
            merge(a, b, ℓ, m, r);
        }
        copy(b, a, 0, len);
    }
}
```

Analyse des iterativen mergesort

- $V^{\mathsf{iter}}_{\mathsf{merge}}(n) =$ maximale Anzahl von Vergleichoperationen für das Sortieren von Listen der Länge n mittels iterativem mergesort
- in der i-ten Phase ($1 \leq i \leq \lceil \log n \rceil$) werden $\lceil n/2^i \rceil$ Paare von sortierten Listen der Länge $(\leq)2^{i-1}$ zu Listen der Länge $(\leq)2^i$ verschmolzen — das erfordert

  $$\lceil \frac{n}{2^i} \rceil \cdot (2^{i-1} + 2^{i-1} - 1) = \lceil \frac{n}{2^i} \rceil \cdot (2^i - 1) \leq 2^i + n$$

  Vergleiche
- Insgesamt (GA Theorem 2.7)

  $$V^{\mathsf{iter}}_{\mathsf{merge}}(n) \leq \sum_{i=1}^{\lceil \log n \rceil} (2^i + n) = n \cdot \lceil \log n \rceil + 2^{\lceil \log n \rceil + 1} - 2 \leq n \cdot \lceil \log n \rceil + 4 \cdot n$$

Rekursion für mergesort

$$V(n) = V(\lceil \frac{n}{2} \rceil) + V(\lfloor \frac{n}{2} \rfloor) + n - 1, \quad V(1) = 0$$

Behauptung

$$V(n) = n \cdot \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1$$

Beweis durch Induktion über n

- $n = 1$

$$0 = 1 \cdot \lceil \log 1 \rceil - 2^{\lceil \log 1 \rceil} + 1$$

- $n > 1$, n gerade, $n = 2m$

$$\begin{aligned} V(n) &= V(m) + V(m) + 2m - 1 \\ &= 2\left(m \cdot \lceil \log m \rceil - 2^{\lceil \log m \rceil} + 1\right) + 2m - 1 \\ &= 2m\left(\lceil \log m \rceil + 1\right) - 2^{\lceil \log m \rceil + 1} + 1 \\ &= n \cdot \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 \end{aligned}$$

- $n > 1$, n ungerade, $n = 2m + 1$

$$V(n) = V(m+1) + V(m) + 2m$$
$$= (m+1)\lceil \log(m+1) \rceil - 2^{\lceil \log(m+1) \rceil} + 1 + m\lceil \log m \rceil - 2^{\lceil \log m \rceil} + 1 + 2m$$

– $2^{k-1} < m < 2^k \Rightarrow \lceil \log(m+1) \rceil = k = \lceil \log m \rceil, \lceil \log n \rceil = k + 1$

$$\begin{aligned} V(n) &= (m+1)k - 2^k + 1 + mk - 2^k + 1 + 2m \\ &= (2m+1)(k+1) - 2^{k+1} + 1 \\ &= n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 \end{aligned}$$

– $m = 2^{k-1} \Rightarrow \lceil \log(m+1) \rceil = k = \lceil \log m \rceil + 1, \lceil \log n \rceil = k + 1$

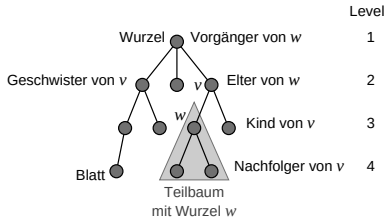
$$\begin{aligned} V(n) &= (m+1)k - 2^k + 1 + m(k-1) - 2^{k-1} + 1 + 2m \\ &= (2m+1)(k+1) - 2^{k+1} + 1 \\ &= n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 \end{aligned}$$

Der Sortieralgorithmus Heapsort

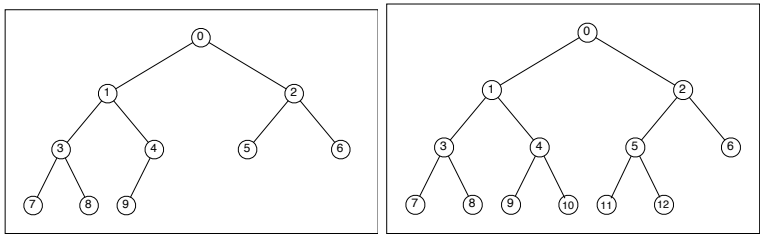
beruht auf einer Idee, wie man Selectionsort verbessern kann:
Information über durchgeführte Vergleiche bei der Minimumsuche aufbewahren

und das noch mit einer effizienten Datenorganisation

Bäume (Terminologie)



Fast vollständige binäre Bäume



haben die Eigenschaften

- alle inneren Knoten — bis auf höchstens einen — haben genau zwei Kinder
- alle Knoten mit weniger als zwei Kindern befinden sich auf den beiden Niveaus maximaler Tiefe
- die Blätter auf maximaler Tiefe sind von links nach rechts aufgefüllt

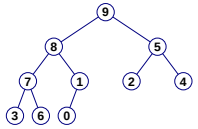
Eigenschaften:

- Zu jeder natürlichen Zahl n gibt es genau einen fast vollständigen binären Baum mit n Knoten
- Numeriert man die Knoten dieses Baumes niveauweise (von der Wurzel ausgehend) und auf jedem Niveau von links nach rechts aufsteigend (kanonische Numerierung) mit den ganzen Zahlen von 0 bis $n-1$, so gilt:
  - der Knoten mit Nummer i hat
    - als linkes Kind (falls vorhanden) den Knoten mit der Nummer $2i+1$
    - als rechtes Kind (falls vorhanden) den Knoten mit der Nummer $2i+2$
    - als Elter den Knoten mit der Nummer $\lfloor \frac{i-1}{2} \rfloor$
  - der Knoten mit Nummer i hat Tiefe $\lfloor \log(i+1) \rfloor$
- Implementierung mit einem Feld der Länge n
  Adressierung mittels kanonischer Numerierung

Heap: Datenstruktur, in der ganze Zahlen (oder Elemente einer total geordneten Menge) gespeichert werden und die folgende Operationen zur Verfügung stellt

- `int size(heap h)` : Anzahl der im Heap `h` gespeicherten Elemente
- `heap create_heap(int array[], int len)` : gibt Heap zurück, der die im `array` der Länge `len` gespeicherten Elemente enthält
- `delete_max(heap h)` : gibt maximales Element aus Heap `h` zurück und löscht dieses

Konzeptuelle Realisierung eines Heaps:



als fast vollständiger binärer Baum, in dessen Knoten Schlüsselwerte abgespeichert sind, wobei für jeden Knoten gilt:

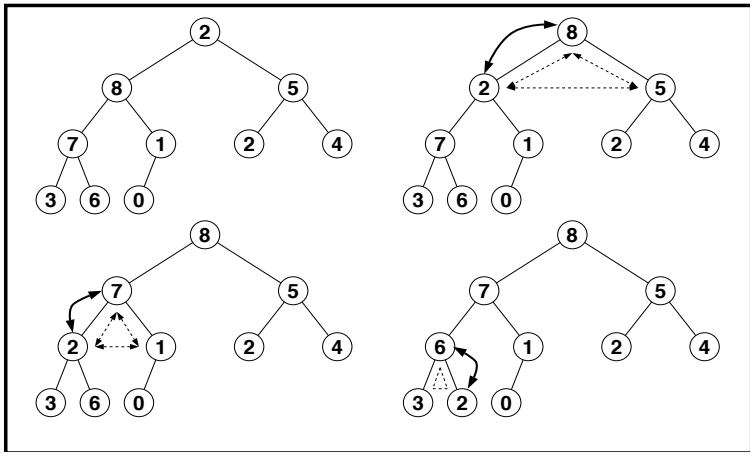
(HE) Der in dem Knoten abgespeicherte Wert ist nicht kleiner als die Werte in seinen Kindern

Sortieren mittels Heaps: Heapsort

```
HEAPSORT (int array[], int len)
{
    heap h = create_heap(array, len);
    while (size(h)> 0) output delete_max(h);
}
```

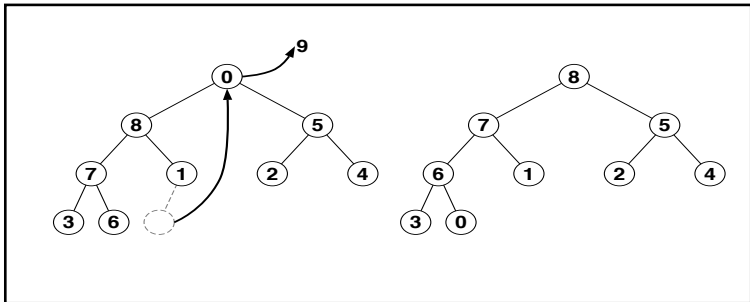

Wiederherstellen der HE, falls diese an der Wurzel verletzt ist

```
REHEAP (heap h)
{
    Sei v die Wurzel des Heaps h;
    while (Heap-Eigenschaft in v nicht erfüllt)
    {
        Sei v* das Kind mit dem größeren Schlüssel;
        Vertausche die Schlüssel in v und v* und setzte v = v*;
    }
}
```

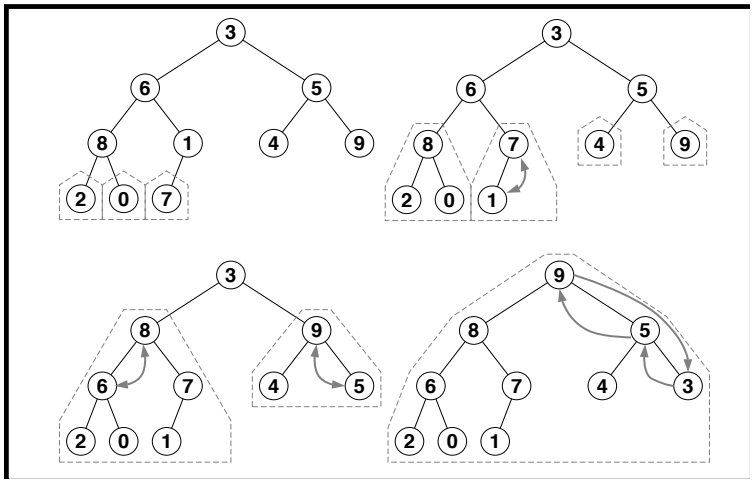
Entfernen des maximalen Elements (Wert an der Wurzel des Heaps) und Wiederherstellung der HE

```
int DELETE_MAX (heap h)
{
    Sei r die Wurzel des Heaps h und sei k der in r gespeicherte Schlüssel;
    Sei ℓ das rechteste Blatt im untersten Level;
    Kopiere den Schlüssel in ℓ in die Wurzel r;
    Lösche das Blatt ℓ und dekrementiere heap_size;
    reheap(h);
    return k;
}
```

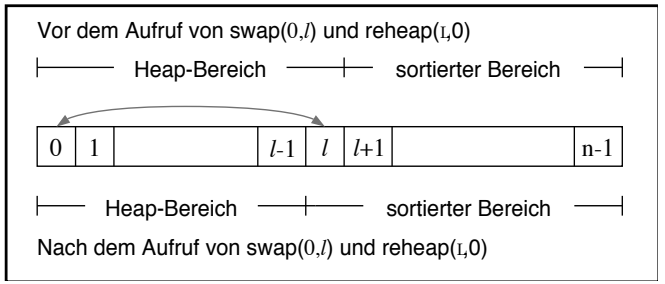


Aufbau eines Heaps aus einer Liste/einem Feld

heap CREATE_HEAP (int $array[]$, int len)
{
  Konstruiere einen fast vollständigen binären Baum H mit len Knoten;
  Setze $heap\_size = len$;
  Fülle jeden Knoten mit einem Schlüssel des Feldes $array$;
  for ($\ell = d(H)$; $\ell \geq 1$; $\ell{-}{-}$)
    for each Knoten v auf Level ℓ
      reheap(Baum H mit Wurzel v);
  return H;
}



Organisation von Heapsort als in-situ-Sortierverfahren:



zu sortierendes Feld → Heap im gleichen Speicherbereich → sukzessives Auslesen des maximalen im (schrumpfenden) Heap verbliebenen Elements mit Aufbau der sortierten Liste in dem vom Heap freigegebenen Bereich

```
HEAPSORT (int array[], int len)
{   /* create_heap */
    for (int i = len − 1; i ≥ 0; i−−)
        reheap(array, len, i);

    /* sort elements */
    for (int ℓ = len − 1; ℓ ≥ 1; ℓ−−)
    {   /* delete_max */
        swap(array[0], array[ℓ]);
        reheap(array, ℓ, 0);
    }
}
```



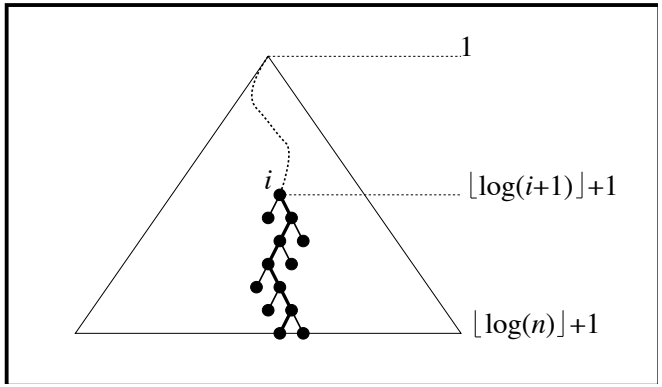
```
REHEAP (int array[], int len, int r)
{
    int i = r;
    int j = 2r + 1;      /* j is always a child of i */
    while (j < len)
    {   /* determine larger child */
        if ((j + 1 < len) && (array[j + 1] > array[j])) j++;
        /* is heap-property not fulfilled? */
        if (array[j] > array[i])
            {
                swap(array[i], array[j]);
                i = j; j = 2 ∗ j + 1;
            }
        else break;
    }
}
```


Analyse der Komplexität von Heapsort

$V_{\texttt{reheap}}(n, i)$ = maximale Anzahl von Vergleichen, die `reheap` auf einem Teilbaum mit Wurzel i und Knoten $< n$ benötigt

$V_{\texttt{create}}(n)$ = maximale Anzahl von Vergleichen, die `create_heap` benötigt um einen Heap mit n Schlüsseln aufzubauen

$V_{\texttt{Heap}}(n)$ = maximale Anzahl von Vergleichen für den Sortiervorgang bei einem Feld von n Elementen



Reheap:

$$V_{\texttt{reheap}}(n,i) = 2(\lfloor \log n \rfloor - \lfloor \log(i+1) \rfloor)$$

Create:

$$V_{\texttt{create}}(n) \leq \sum_{i=0}^{n-1} V_{\texttt{reheap}}(n,i) \leq \ldots \leq 5n$$

Sortieren:

$$V_{\texttt{Heap}}(n) \leq \sum_{\ell=1}^{n-1} V_{\texttt{reheap}}(\ell,0) \leq \ldots \leq 2n \log n$$

Zusammenfassung (GA, Theorem 2.8):

Mittels Heapsort kann ein Feld der Länge n in-situ mit höchstens $2n \log n + 5n$ (genauer sogar: $2n \log n + \frac{7}{2}n$) Vergleichen sortiert werden.

Ergänzende Bemerkungen

- Heapsort benötigt mehr Vergleiche als Mergesort, aber weniger zusätzlichen Speicherplatz (in-situ)
- Es gibt Varianten, die noch besser als standard-Heapsort sind (siehe GA)
  - Carlssons Verfahren: $n \log n + n \log \log n + O(n)$ Vergleiche (mittels binärer Suche)
  - Bottom-Up Heapsort: $\frac{3}{2} n \log n + O(n)$ Vergleiche
  - Vermutung zu Bottom-Up-Heppsort: im Mittel $n \log n + O(n)$ Vergleiche

Quicksort (C.A.R. Hoare, Computer Journal, 1962)

- Klassisches, sehr beliebtes, weil im Mittel sehr effizientes Sortierverfahren
- In vielen Systemen (Unix) das Standardverfahren
- Basiert auf "dynamischem" (datengetriebenem) divide-and-conquer
- Wenig zusätzlicher Speicherplatz ($O(\log^2 n)$ bei Listelänge n)
- Vorsicht: worst-case Verhalten ist ineffizient!
- Paradoxon: worst-case tritt z.B. bei schon sortierten Listen ein
- Viele Varianten und Verbesserungen, um worst-case-Verhalten sehr unwahrscheinlich zu machen

R. Sedgewick in *Algorithms*:

> It is tempting to try to develop ways to improve Quicksort: a faster sorting algorithm is computer science's "better mousetrap".
>
> Almost from the moment Hoare first published the algorithm, "improved" version have been appearing in the literature.
>
> Many ideas have been tried and analyzed, but it is easy to be deceived, because the algorithm is so well balanced that the effects of improvements in one part of the program can be more than offset by the effects of bad performance in another part of the program.
>
> A carefully tuned version of Quicksort is likely to run significantly faster on most computers than any other sorting method.
>
> However, it must be cautioned that tuning any algorithm can make it more fragile, leading to undesirable and unexpected effects for some inputs

Zur Implementierung und detailierten Analyse:
R. Sedgewick, Ouicksort (Diss.), R. Sedgewick, Implementing Quicksort Programs, Communications of the ACM, 1978, J.L. Bentley, M.D. McIlroy, Software Practice and Experience, 1993 ($\rightarrow$ Unix)

Die Idee von Quicksort

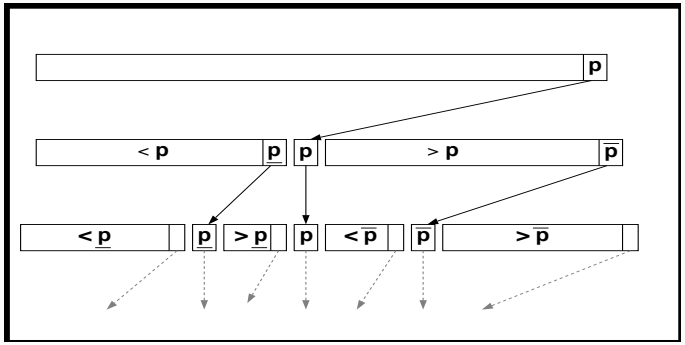
- Ein "splitter" in einer Liste $L[1\,..\,n]$ ist ein Index j $(1 \leq j \leq n)$ mit

$$\forall i < j : L[i] < L[j] \quad \text{und} \quad \forall k > j : L[j] < L[k]$$

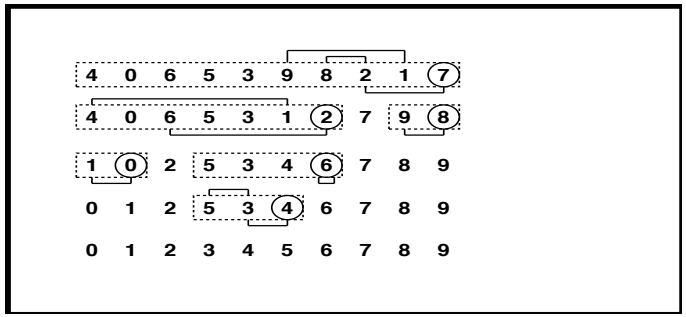
- Soll die Liste $L[1\,..\,n]$ sortiert werden und ist j ein splitter, so steht $L[j]$ bereits an der richtigen Stelle: es genügt also, $L[1\,..\,j-1]$ und $L[j+1\,..\,n]$ zu sortieren! (divide-and-conquer)
- Die Wahrscheinlichkeit, dass eine (zufällige) Liste einen splitter hat, ist sehr klein.
- Man muss sich einen splitter durch geeignete (nicht zu teure!) Umordnung ("partition") beschaffen

- Einfachste Idee:
  - wähle ein Element der Liste $L[1\,..\,n]$, etwa $p := L[n]$, zum "Pivot" (besser: wähle erst $i \in [1\,..\,n]$ zufällig, vertausche $L[i] \leftrightarrow L[n]$)
  - durchlaufe die Liste $L[1\,..\,n-1]$ einmal und erzeuge durch Vergleich mit p die Teillisten
    - $L'[1\,..\,j-1]$ — enthält Elemente von $L[1\,..\,n]$ die $< p$ sind
    - $L'[j+1\,..\,n]$ — enthält Elemente von $L[1\,..\,n]$ die $> p$ sind
  - mit $L'[j] = p$ ist j ein splitter für $L'[1\,..\,n]$
- Jetzt rekursive Anwendung auf $L'[1\,..\,j-1]$ und $L'[j+1\,..\,n]$

divide-and-conquer-Schema für Quicksort



Beispiel



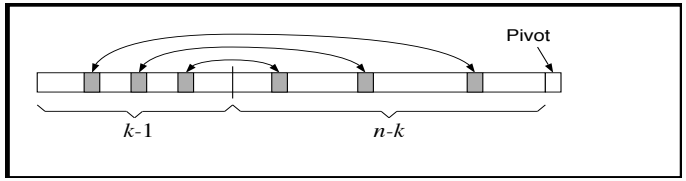

```
QUICKSORT (int array[], int ℓ, int r)
{
    if (ℓ < r)
    {
        int pivot = partition(array, ℓ, r, r);
        if (pivot − ℓ < r − pivot)
        {
            quicksort(array, ℓ, pivot − 1);
            quicksort(array, pivot + 1, r);
        }
        else
        {
            quicksort(array, pivot + 1, r);
            quicksort(array, ℓ, pivot − 1);
        }
    }
}

int partition(int array[], int ℓ, int r, int pivot)
{
    int i = ℓ − 1, j = r;                /* left and right pointer */
    swap(array[pivot], array[r]);        /* move pivot to the right end */
    pivot = r;
    while (i < j)
    {
        do i++; while ((i < j) && (array[i] < array[pivot]));
        do j−−; while ((j > i) && (array[j] > array[pivot]));
        if (i ≥ j)
            swap(array[i], array[pivot]);
        else
            swap(array[i], array[j]);
    }
    return i;
}
```



```
QUICKSORT (int array[], int ℓ, int r)
{
    if (ℓ < r)
    {
        int pivot = partition(array, ℓ, r, r);
        if (pivot − ℓ < r − pivot)
        {
            quicksort(array, ℓ, pivot − 1);
            quicksort(array, pivot + 1, r);
        }
        else
        {
            quicksort(array, pivot + 1, r);
            quicksort(array, ℓ, pivot − 1);
        }
    }
}
```

```
int partition(int array[], int ℓ, int r, int pivot)
{
    int i = ℓ − 1, j = r;                /* left and right pointer */
    swap(array[pivot], array[r]);        /* move pivot to the right end */
    pivot = r;
    while (i < j)
    {
        do i++; while ((i < j) && (array[i] < array[pivot]));
        do j−−; while ((j > i) && (array[j] > array[pivot]));
        if (i ≥ j)
            swap(array[i], array[pivot]);
        else
            swap(array[i], array[j]);
    }
    return i;
}
```


Schema des Partitionierens



Zur Struktur des Partitionierens

Notation:

A : totalgeordnete Menge

für $a \in A$: $A_{<a} = \{x \in A; x < a\}$
$A_{>a} = \{x \in A; x > A\}$
$A_{\neq a} = \{x \in A; x \neq a\}$

für $0 \leq i \leq \sharp A$: $\binom{A}{i}$: i-elementige Teilmengen von A

$\mathfrak{S}(A)$: Permutationen von A (in Listenschreibweise)

Behauptung: für jedes $a \in A$ definiert `partition` eine bijektive Abbildung

$$\mathfrak{S}(A_{\neq a}) \;\leftrightarrow\; \mathfrak{S}(A_{<a}) \times \mathfrak{S}(A_{>a}) \times \bigcup_{i=0}^{\min(\sharp A_{<a}, \sharp A_{>a})} \binom{A_{<a}}{i} \times \binom{A_{>a}}{i}$$

Beispiel

$A = \{0\,..\,9\}$, $a = 4$ (als Pivot)

$$\sigma = [3, 7, 9, 1, 6, 0, 8, 5, 2] \in \mathfrak{S}(A_{\neq 4})$$
$$\downarrow$$
$$[3, \mathbf{2}, \mathbf{0}, 1, 6, \mathbf{9}, 8, 5, \mathbf{7}]$$
$$\downarrow$$
$$\left(\sigma_{<a}, \sigma_{>a}, (A^{\sigma}_{<a}, A^{\sigma}_{>a})\right) = \left(\underbrace{[3,2,0,1]}_{\in \mathfrak{S}(A_{<4})}, \underbrace{[6,9,8,5,7]}_{\in \mathfrak{S}(A_{>4})}, \underbrace{(\{0,2\},\{7,9\})}_{\in \binom{A_{<4}}{2} \times \binom{A_{>4}}{2}}\right)$$

Folgerung:

bei der Abbildung

$$\sigma \mapsto (\sigma_{<a}, \sigma_{>a}, (A^\sigma_{<a}, A^\sigma_{>a}))$$

$$\mathfrak{S}(A_{\neq a}) \;\leftrightarrow\; \mathfrak{S}(A_{<a}) \times \mathfrak{S}(A_{>a}) \times \bigcup_{i=0}^{\min(\sharp A_{<a}, \sharp A_{>a})} \binom{A_{<a}}{i} \times \binom{A_{>a}}{i}$$

- tritt als $\sigma_{<a}$ jedes Element von $\mathfrak{S}(A_{<a})$ gleich oft, nämlich $\frac{(\sharp A_{\neq a})!}{(\sharp A_{<a})!}$-mal auf
- tritt als $\sigma_{>a}$ jedes Element von $\mathfrak{S}(A_{>a})$ gleich oft, nämlich $\frac{(\sharp A_{\neq a})!}{(\sharp A_{>a})!}$-mal auf

Bemerkung

Aus der Bijektivität der Abbildung folgt mit $\sharp A_{<a} = s, \sharp A_{>a} = t$, also $\sharp A_{\neq a} = s + t$, dass

$$(s+t)! = s! \cdot t! \cdot \sum_{i=0}^{\min(s,t)} \binom{s}{i}\binom{t}{i}$$

und daraus schliesst man, dass für beliebige $s, t \in \mathbb{N}$ gilt

$$\binom{s+t}{s} = \sum_{i=0}^{\min(s,t)} \binom{s}{i}\binom{t}{i}$$

Das ist die berühmte Identität Chu-Vandermonde.

Chu Shih-Chieh, 1270-1330, *Precious Book of the Four Elements*.

Alexandre-Théophile Vandermonde, 1735-1796, *Mémoire sur des irrationnelles de différents ordres avec une application au cercle (1772)*.

Folgerung für `partition` in Quicksort:

- Betrachtet man für festes $a \in A$ die Permutationen $\sigma \in \mathfrak{S}(A_{\neq a})$ mit Gleichverteilung und partitioniert man nach dem Pivot a, so treten in den Teilen die Permutationen als $\sigma_{<a}$ bzw. $\sigma_{>a}$ die Elemente von $\mathfrak{S}(A_{<a})$ bzw. $\mathfrak{S}(A_{>a})$ wiederum mit Gleichverteilung auf.

$V(L)$: Anzahl der Vergleichsoperationen zum Sortieren von $L = L[1\,..\,n]$ mittels Quicksort

Erinnerung:

`partition` : $L[1\,..\,n] \mapsto \langle L'[1\,..\,j-1], L'[j+1\,..\,n]\rangle$ falls $j =$ Rang von $L[n]$ in L

Daher

$$V(L) = n - 1 + V(L'[1\,..\,j-1]) + V(L'[j+1\,..\,n]) \text{ falls } j = \text{Rang von } L[n] \text{ in } L$$

- worst case: Pivot ist Maximum oder Minimum
  eine der Listen $L'[1\,..\,j-1]$ bzw. $L'[j+1\,..\,n]$ ist leer
  d.h. $j = 1$ oder $j = n$

  Tritt dies in jedem Rekursionsschritt auf, so erhält man für Listen der Länge n im schlechtesten Fall

$$v_{\texttt{max}}(n) = n - 1 + v_{\texttt{max}}(n-1) \ (n > 1)$$
$$v_{\texttt{max}}(1) = 0$$
$$\text{also} \quad v_{\texttt{max}}(n) = \frac{n(n-1)}{2} = O(n^2)$$

- best case: Pivot ist Median
  die Listen $L'[1\,..\,j-1]$ und $L'[j+1\,..\,n]$ sind (in etwa) gleich lang,
  d.h. $j = \lceil \frac{n+1}{2} \rceil$ oder $j = \lfloor \frac{n+1}{2} \rfloor$

  Tritt dies in jedem Rekursionsschritt auf, so erhält man für Listen der Länge n im besten Fall

$$v_{\texttt{min}}(n) = n - 1 + v_{\texttt{min}}(\lceil \frac{n-1}{2} \rceil) + v_{\texttt{min}}(\lfloor \frac{n-1}{2} \rfloor) \ (n > 1)$$
$$v_{\texttt{min}}(1) = 0$$
$$\text{also} \quad v_{\texttt{min}}(n) = n \log n + O(n)$$

  - Annahme: Permutationsmodell
  - Folgerung: jedes Element der zu sortierenden Menge (n Elemente) hat gleiche Wahrscheinlichkeit $= 1/n$ als Pivot verwendet zu werden
  - Gleichverteilungsannahme vererbt sich auf Teilprobleme (s.o.)
  - Folgerung:
    $\overline{V}_{\texttt{quick}}(n) =$ mittlere Anzahl von Vergleichen für Quicksort auf Feld der Länge n
    genügt der Rekursion

$$\overline{V}_{\texttt{quick}}(1) = 0$$
$$\overline{V}_{\texttt{quick}}(n) = \sum_{k=1}^{n} \frac{1}{n} \left[(n-1) + \overline{V}_{\texttt{quick}}(k-1) + \overline{V}_{\texttt{quick}}(n-k) \right] \quad (n \geq 2)$$

Lösung der Quicksort-Rekursion:

$$\overline{V}_{\texttt{quick}}(n) = \sum_{k=1}^{n} \frac{1}{n} \left[(n-1) + \overline{V}_{\texttt{quick}}(k-1) + \overline{V}_{\texttt{quick}}(n-k) \right] \quad (n \geq 2)$$
$$= (n-1) + \frac{2}{n} \sum_{k=1}^{n-1} \overline{V}_{\texttt{quick}}(k)$$

Umformungen führen zu

$$n\overline{V}_{\texttt{quick}}(n) = (n+1)\overline{V}_{\texttt{quick}}(n-1) + 2(n-1)$$
$$\frac{\overline{V}_{\texttt{quick}}(n)}{n+1} = \frac{\overline{V}_{\texttt{quick}}(n-1)}{n} + \frac{4}{n+1} - \frac{2}{n}$$

und schliesslich

$$\frac{\overline{V}_{\texttt{quick}}(n)}{n+1} = \sum_{j=3}^{n+1} \frac{4}{j} - \sum_{j=2}^{n} \frac{2}{j}$$
$$= 2H_n + \frac{4}{n+1} - 4 \; = 2\ln n + O(1)$$

Satz: Quicksort benötigt zum Sortieren eines Feldes der Länge n im Mittel $2\ln 2 \cdot n \log n + O(n) \sim 1.386\, n \log n$ Vergleiche.
(GA, Theorem 2.13)

Varianten und Verbesserungen

- Beim Aufteilen immer den kleineren Teilbereich zuerst behandeln: die Anzahl der noch nicht sortierten Teilbereiche ($\rightarrow$ zu speichernde Indexgrenzen) bleibt durch $O(\log n)$ beschränkt.
- Kleine Felder (z.B. Länge ≤ 10) nicht rekursiv, sondern mit Insertionsort sortieren.
- Randomisierter Quicksort: zufällige Pivot-Wahl verhindert schlechte Instanzen, aber nicht worst-case-Verhalten $O(n^2)$, d.h. erwartete Zahl der Vergleiche ist $2\ln 2 \cdot n \log n + O(n) \sim 1.386\, n \log n$ für jeden input.
- Median-of-Three-Quicksort: Median von drei Elementen an festen Position als Pivot wählen: mittlere Anzahl der Vergleiche ist
$\sim \frac{12}{7} \ln 2 \cdot n \log n + O(n) \sim 1.188 n \log n$
- Randomisierter Median-of-Three-Quicksort: Median von zufällig gwählzen Elementen als Pivot wählen: erwartete Anzahl der Vergleiche (für jeden input) ist $\sim \frac{12}{7} \ln 2 \cdot n \log n + O(n) \sim 1.188 n \log n$

Quicksort

1 Vorbemerkung

Der um 1960 von C.A.R. Hoare vorgeschlagene Sortieralgorithmus *quicksort* gehört zweifellos zu den populärsten, am meisten verwendeten und am besten untersuchten Algorithmen überhaupt. Sortieren ist ohnehin eine recht häufige Aufgabe, und mit einigem Recht kann man *quicksort* als einen der besten "general-purpose" Sortieralgorithmen ansprechen. Solche Qualifizierungen bedürfen natürlich einer genaueren Begründung, die ihrerseits nur auf der Basis umfangreicher Erfahrung und Vergleiche möglich ist. Darüber ist soviel und kompetent geschrieben worden, daß es keinen Sinn macht, das hier in wenigen Sätzen darstellen zu wollen. Ein Verweis auf Autoritäten wie Knuth, Sedgewick und Bentley muß genügen. Insbesondere die Dissertation von *Sedgewick* ist ein studierenswertes Beispiel skrupulöser Auseinandersetzung mit Implementierung und Analyse der algorithmischen Idee von *quicksort*. So bestechend einfach die ursprüngliche Idee von *Hoare* ist, ihre Realsierung ist durchaus nicht ohne Tücken! Die Zeigerbewegungen in der *splitting*-Phase muß man sehr sorgfältig behandeln, spezielle Vorkehrungen sind zu treffen, um das *worst-case* Verhalten[1] zu verbessern, und durch iterative Implementierung kann man zwar die Effizienz steigen, die Programmierung wird aber delikater.

Die Analyse des *average-case* Verhaltens von *quicksort* ist ein "Klassiker" der Algorithmenanalyse, und darum geht es hier vor allem. Wie schon in den früher behandelten *divide-and-conquer* Situationen führt die rekursive Struktur des Algorithmus auf eine Rekursionsgleichung für die Kosten. Hier allerdings kann die Problemgrösse der jeweils induzierten Teilprobleme stark schwanken — was sich in einem anderen Typ von Rekursionsgleichung äussert.

Es wird sich zeigen, daß *quicksort* im Mittel einen Aufwand (gemessen in der Anzahl der Vergleichsoperationen) benötigt, der sich bei Listenlänge n wie $2n \log n$ verhält, also asymptotisch von optimaler Wachstumsordnung ist. Dies korrespondiert mit den guten praktischen Erfahrungen, die ja auch noch den overhead des Verfahrens berücksichtigen, der bei der Zählung der Vergleichsoperationen unterschlagen wird. *quicksort* kann aus dieser Sicht sehr gut mit Algorithmen wie *heapsort* oder *mergesort* konkurrieren, die ja auch im *worst case* ein $\Theta(n \log n)$ verhalten zeigen.

2 Die Idee — rekursives splitting

quicksort basiert auf einer bestechend einfachen Idee (man muß nur darauf kommen):

> In einer zu sortierenden Liste $A[1..n]$ (von ganzen Zahlen etwa) ist das Element an der k-ten Position, $A[k]$ also, ein *splitter*, wenn es an der "richtigen" Position in Bezug auf die Liste steht, die aus $A[1..n]$ durch Sortieren entsteht. D.h. es gilt
>
> $$\forall i < k \ : \ A[i] \leq A[k] \quad \wedge \quad \forall j > k \ : \ A[k] \leq A[j]$$
>
> Hat man eine solche Information, kann man $A[1..n]$ dadurch sortieren, daß man die Teillisten $A[1..k-1]$ und $A[k+1..n]$ separat sortiert. Hat man eine solche Information nicht, wird man durch eine spezielle Prozedur *split* (mit möglichst wenig Aufwand) $A[1..n]$ so umordnen, daß dabei ein *splitter* in einer bekannten Position (!) entsteht — dann kann man zu den Teillisten wie beschrieben übergehen und dieses Verfahren rekursiv anwenden.

Eine *quicksort*-Implementierung hat also die Struktur

```
qksort := proc(A:array(integer),left:integer,right:integer)
if left < right then
   split(A,left,right,'k');
   qksort(A,left,k-1);
   qksort(A,k+1,right)
   fi;
end;
```

wobei der Aufruf zum Sortieren von `A[left..right]` so behandelt wird, daß durch den Aufruf `split(A,left,right,'k')` eine Umordnung `A'[left..right]` von `A[left..right]` (*in place*) erzeugt wird, bei der `A'[k]` ein *splitter* ist. Auf `A'[left..k-1]` und `A'[k+1..right]` wird das Verfahren rekursiv angewandt. Der ganze Witz von *quicksort* liegt also im *splitting* !

3 Splitting

Die ursprüngliche Idee von Hoare besteht darin, zum Splitten einer Liste $A[1..n]$ so vorzugehen, daß man zwei Zeiger, i und j, und zwar i von 1 her aufsteigend, j von n her absteigend, aufeinander zubewegt. Jedesmal, wenn man $A[i] > A[j]$ feststellt, werden diese beiden Elemente vertauscht — danach setzen die Zeiger ihren Weg analog fort. Treffen sich die beiden Zeiger, hat man die Position eines *splitters* gefunden. Die Details der Zeigerbewegungen (Randfälle!) sind etwas subtil. Man kann sie in der Literatur nachlesen.

Konzeptuell und für die Implementierung etwas einfacher ist eine elegante Idee von N. Lomuto, die von Bentley propagiert wird. Hier bewegen sich auch zwei Zeiger i und j, aber in der gleichen Richtung! Das Listenelement $A[1]$ soll *splitter* werden. Man startet mit j von $j = 1$ aus und läuft so lange, bis man einen Index j mit $A[j] < A[1]$ findet - dann erhöht man i (das auch bei $i = 1$ startet) um 1 und vertauscht $A[i]$ und $A[j]$. Dieses Spiel (j erhöhen bis wieder $A[j] < A[1]$ gefunden wird, dann mit i um eine Stelle nachziehen und $A[i]$ mit $A[j]$ vertauschen) setzt sich fort, bis Zeiger j das Listenende erreicht. Dann werden $A[i]$ und $A[1]$ vertauscht — mit dem Erfolg, daß das nunmehr an der Stelle i stehende Element (das anfängliche A[1]) ein *splitter* der Liste ist.

[1] das sich bei naiver Implementierung paradoxerweise gerade dann zeigt, wenn man *quicksort* auf schon weitgehend sortierte Listen anwendet

Es geht kein Weg dran vorbei: wenn man verstehen will, wie und warum das funktioniert, muß man sich — etwa anhand des weiter unten reproduzierten Programmtextes — Schritt für Schritt durch Beispiele arbeiten. Das bleibt der Eigeninitiative überlassen. Vorher aber noch ein Hinweis auf eine sinnvolle Modifikation der Splittings.

4 Eine Verbesserung: Splitting mit "Randomisieren"

Die eben beschriebene Idee des Splittings hat noch einen Nachteil: es gibt (sehr einfache!) Listen, bei denen die Aufteilung in Teillisten vor und nach dem *splitter* sehr extrem ausfällt, wenn man das erste Listenelement als Vergleichselement nimmt: die linke Teilliste ist leer und die Länge der rechten Teilliste ist gerade mal um 1 kleiner als die Länge des ursprünglichen inputs. Wenn sich diese Erscheinung durch alle Phasen des Algorithmus durchzieht, wird er ein *quadratisches* Laufzeitverhalten im *worst case* haben! Ärgerlich daran ist, daß dieses "schlechte" Verhalten immer bei denselben inputs auftritt. Was man zumindest erreichen möchte, ist dies: es gibt keine inputs bei denen sich der Algorithmus *immer* schlecht verhält. Das kann man bei einem seiner Natur nach *deterministischen* Algorithmus nur dadurch erreichen, indem man ein *zufälliges* Element mit einbaut. Wenn man das geschickt macht, kann man dafür sorgen, daß es keine *schlechten Instanzen* für den Algorithmus gibt, sondern nur noch *schlechte Exekutionen*.

Beim Splitting kann man eine solche Verbesserung dadurch erreichen, daß man zu Beginn ein zufälliges Element $A[r]$ mit $1 \leq r \leq n$ auswählt und mit $A[1]$ vertauscht. Damit haben alle Listenelemente die gleiche Chance, *splitter* zu sein, unabhängig von der ursprünglichen Form der Liste. Klar ist dann auch, daß die *Position* k des *splitters* jede der Positionen $1 \leq k \leq n$ mit gleicher Wahrscheinlichkeit $1/n$ sein wird.

Die hier angesprochene Technik des "Randomisierens" ist in den letzten Jahren sehr populär geworden. "Randomisierte Algorithmen", auch für Problemstellungen, wo man ein determiniertes Resultat erwartet, können u.a. verhindern, daß lange Laufzeiten immer bei den gleichen inputs auftreten (wenn sie denn unvermeidbar sind). Für viele Fragestellungen gibt es randomisierte Algorithmen, die weitaus besser sind als die besten deterministischen Algorithmen, wenn man noch geringe "Versagerquoten" in Kauf zu nehmen bereit ist.

5 Das Programm

In diesem Abschnitt wird ein MAPLE Programm *quicksort* präsentiert. Dabei kommt es nur auf die korrekte und klare Realisierung der dargestellten Ideen an, nicht auf Effizienzsteigerung durch Detailverbesserung. Interessenten (insbesondere auch solche, die sich mit den Interna von MAPLE etwas auskennen), sind eingeladen, dieses Programm zu "tunen" oder gleich eine noch Maschinen-näheres Programm zu schreiben. Es wurde ebenfalls darauf verzichtet, Konsistenz-checks der Eingabedaten mit in die einzelnen Prozeduren aufzunehmen.

Die Prozedur `swap` dient lediglich dazu, zwei Elemente eines arrays zu vertauschen, nämlich die Elemente in Position `u` und Position `v`.

```
swap := proc(A:array(integer),u:integer,v:integer)
local tmp;
if not (u=v) then
   tmp := A[u];
   A[u] := A[v];
   A[v] := tmp;
   fi;
RETURN(op(A));
end;
```

Die Prozedur `split` ist das Herzstück von *quicksort*. Hierbei wird der Abschnitt `A[left..right]` eines arrays `A[lower..upper]` dem *splitting* unterworfen — es wird unterstellt, daß `left` und `right` mit den array-Grenzen verträglich sind. Aus dem Intervall `[left..right]` wird per Zufallsgenerator ein `r` ausgewählt, und `A[r]` wird als *splitting*-Element bestimmt (und mit `A[left]` vertauscht). Der Index `I` als Resultat der Prozedur gibt an, in welcher Position dieses Element *nach* dem *splitting* steht. Als Seiteneffekt wird `A` entsprechend dem *splitting* verändert.

```
split := proc(A:array(integer),left:integer,right:integer,I:string)
local i,j,r;
r := rand(left..right)();
swap(A,left,r);
i := left;
for j from left+1 to right do
   if A[j] < A[left] then
      i := i+1;
      swap(A,i,j)
      fi;
   od;
swap(A,left,i);
I:=i;
end;
```

Die rekursive Struktur von *quicksort* wurde schon oben angesprochen. `qksort` sortiert den Bereich `A[left..right]` der arrays `A`, indem durch `split(A,left,right,'k')` ein *splitter* an der Stelle `k` erzeugt wird. Für Teilarrays links und rechts davon wird `qksort` rekursiv aufgerufen. Bei arrays der Länge ≤ 1 ist nichts zu tun.[2]

```
qksort := proc(A:array(integer),left:integer,right:integer)
local k;
if left < right then
   split(A,left,right,'k');
   qksort(A,left,k-1);
   qksort(A,k+1,right)
   fi;
end;
```

`quicksort(L)` schließlich sortiert Listen `L` durch Aufruf von `qksort` mit der Listenlänge als Parameter.

```
quicksort := proc(L:list(integer))
local length,A;
length := nops(L);
A := convert(L,array);
qksort(A,1,length);
convert(op(A),list);
end
```

6 Die *quicksort*-Rekursion

Es bezeichne nun $F(n)$ die *mittlere Anzahl* von Vergleichsoperationen, bei Ausführung von *quicksort* auf Listen der Länge n. Wie schon früher wollen wir annehmen,

[2]Bei "richtigen" Implementierungen von *quicksort* wird man die Rekursion nicht so weit treiben: man schaltet auf ein schnelles, nichtrekursives Verfahren (wie z.B. *insertionsort* um, sobald die Länge der arrays 10-15 (etwa) unterschreitet.

daß alle $n!$ relativen Anordnungen (Permutationen) von n Elementen mit der gleichen Wahrscheinlichkeit auftreten. Aus der Struktur des Algorithmus ergibt sich somit:

$$F(n) = (n-1) + \frac{1}{n}\sum_{k=1}^{n}\{F(k-1) + F(n-k)\}$$

wobei $n-1$ die Anzahl der Vergleichsoperationen in *split* bei Listenlänge n ist, und k die Position des *splitters* angibt: jede dieser Positionen hat die gleiche Wahrscheinlichkeit $1/n$. Zu Sortieren sind dann Teillisten der Längen $k-1$ und $n-k$, und man überlegt sich leicht, daß hierbei wiederum alle $(k-1)!$ bzw $(n-k)!$ relativen Anordnungen mit gleicher Wahrscheinlichkeit auftreten.

Eine kleine Vereinfachung ergibt sich aus der Beobachtung, daß auf der rechten Seite der Rekursion zweimal die gleiche Summe (mit unterschiedlicher Summationsrichtung) steht, also:

$$F(0) = 0\ ,\ F(n) = n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} F(i)$$

Die Frage ist nun, wie sich die Lösung $F(n)$ dieser Rekursion für $n \to \infty$ verhält. Experimentell wird rasch klar, daß $F(n)$ stärker als linear wächst, aber auch nicht sehr viel stärker:

$$\begin{aligned}
F(1) &= 0\\
F(2) &= 1\\
F(3) &= 8/3 = 2.66...\\
F(4) &= 29/6 = 4.83...\\
F(5) &= 37/5 = 7.4\\
F(6) &= 10.3\\
F(7) &= 13.48...\\
F(8) &= 16.92...\\
F(9) &= 20.57...\\
F(10) &= 30791/1260 = 24.43...\\
F(100) &= 647.85...\\
F(1000) &= 10985.9...
\end{aligned}$$

Interessant ist, daß sich die Lösung der *quicksort*-Rekursion *explizit* angeben läßt:

$$F(n) = 2\,(n+1)\,H_n - 4\,n$$

und damit kann man leicht auch den Funktionsverlauf für größere n verfolgen:

$$\begin{aligned}
F(10^4) &= 155771.6...\\
F(10^5) &= 0.2018...\,10^7\\
F(10^6) &= 0.2478...\,10^8\\
F(10^7) &= 0.2939...\,10^9\\
F(10^8) &= 0.3399...\,10^{10}
\end{aligned}$$

$$\begin{aligned} F(10^9) &= 0.3860...\,10^{11} \\ F(10^{10}) &= 0.4320...\,10^{12} \end{aligned}$$

Insgesamt zeigt sich also:

$$F(n) \sim 2\,n\,\log n$$

7 Zur Analyse der Quicksort-Rekursion

In einem ersten Abschnitt wird die Quicksort-Rekursion behandelt, indem sie in eine Differenzengleichung umgeformt wird, die einer expliziten Lösung zugänglich ist. Allgemeiner gilt, daß sich Rekursionen dieses Typs mit Hilfe von Differentialgleichungen für die "erzeugenden Funktionen" der Lösung beschreiben lassen. Diese Technik wird in einem zweiten Abschnitt vorgestellt.

7.1 Behandlung mittels Differenzengleichung

Wir schreiben die Quicksort-Rekursion

$$F(0) = 0\ ,\ F(n) = n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} F(i)$$

in der Form

$$n\,F(n) = n(n-1) + 2\sum_{i=1}^{n} F(i-1) \quad (n > 0)$$

und ziehen diese Gleichung, aber mit n durch $n-1$ ersetzt, also

$$(n-1)\,F(n-1) = (n-1)(n-2) + 2\sum_{i=1}^{n-1} F(i-1) \quad (n > 0)$$

von der vorigen Gleichung ab:

$$n\,F(n) - (n-1)\,F(n-1) = 2\,(n-1) + 2\,F(n-1)$$

Das ergibt

$$n\,F(n) - (n+1)\,F(n-1) = 2\,(n-1)$$

Dies ist eine lineare Differenzengleichung 1. Ordnung mit *polynomialen* Koeffizienten. Ersetzt man $F(n)$ durch $(n+1)\,G(n)$, so genügt diese neue Funktion $G(n)$ einer linearen Differenzengleichung 1. Ordnung mit *konstanten* Koeffizienten:

$$G(n) - G(n-1) = \frac{2\,(n-1)}{n\,(n+1)} \quad (n > 0)\ ,\quad G(0) = 0$$

Hier kann man zweimal den "Teleskop-Trick" spielen:

$$\begin{aligned} G(n) &= \sum_{i=1}^{n} \{G(i) - G(i-1)\} \\ &= \sum_{i=1}^{n} \frac{2\,(i-1)}{i\,(i+1)} \\ &= 2\sum_{i=1}^{n} \left\{\frac{2}{i+1} - \frac{1}{i}\right\} \\ &= 2\sum_{i=1}^{n} \left\{\frac{2}{i+1} - \frac{2}{i}\right\} + 2\sum_{i=1}^{n} \frac{1}{i} \\ &= 2\left\{\frac{2}{n+1} - 2\right\} + 2\,H_n \\ &= -\frac{4\,n}{n+1} + 2\,H_n \end{aligned}$$

und somit

$$F(n) = (n+1)\,G(n) = 2\,(n+1)\,H_n - 4\,n \sim 2\,n\,\log n$$

7.2 Behandlung mittels Differentialgleichung

Wir betrachten wieder

$$F(0) = 0\ ,\ F(n) = n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} F(i)$$

und definieren uns die "erzeugende Funktion"

$$\phi(z) := \sum_{n\geq 0} F(n)\,z^n$$

Multiplikation der Rekursionsgleichung mit n liefert

$$n\,F(n) = n(n-1) + 2\sum_{i=0}^{n-1} F(i) \quad (n \geq 0)$$

Multiplikation mit z^n und Summation über $n \geq 0$ ergibt

$$\sum_{n\geq 1} n\,F(n)\,z^n = \sum_{n\geq 1} n(n-1)\,z^n + 2\sum_{n\geq 1}\sum_{i=0}^{n-1} F(i)\,z^n$$

und das ist äquivalent zu

$$\phi'(z) = \frac{2z}{(1-z)^3} + \frac{2}{1-z}\,\phi(z)$$

Diese lineare Differentialgleichung kann man mittels der Methode der *Variation der Konstanten* lösen. Dazu betrachtet man erst einmal die zugehörige *homogene* Gleichung

$$\phi_h'(z) = \frac{2}{1-z}\,\phi_h(z)$$

die man leicht lösen kann.

$$\frac{d}{dz}\,\log \phi_h(z) = \frac{\phi_h'(z)}{\phi_h(z)} = \frac{2}{1-z}$$

liefert

$$\log \phi_h(z) = -2\log(1-z) + \gamma$$

mit einer Integrationskonstanten γ, also

$$\phi_h(z) = \frac{\lambda}{(1-z)^2}$$

mit einer Konstanten λ. Nun der Ansatz

$$\phi(z) = \frac{\lambda(z)}{(1-z)^2}$$

d.h. für die Behandlung der inhomogenen Gleichung wird $\phi(z)$ so angesetzt, daß an Stelle der Konstanten λ eine Funktion $\lambda(z)$ erscheint. Dies in die urspüngliche Differentialgleichung eingesetzt liefert eine Differentialgleichung für die zu bestimmende Funktion $\lambda(z)$:

$$\frac{\lambda'(z)}{(1-z)^2} = \frac{2z}{(1-z)^3}$$

oder

$$\lambda'(z) = \frac{2z}{1-z} = \frac{2}{1-z} - 2$$

was offenbar durch

$$\lambda(z) = 2\,(-\log(1-z) - z) + c$$

gelöst wird. Hierbei ist c eine passende Konstante. Damit hat man die Lösung:

$$\phi(z) = 2\left(\frac{-1}{(1-z)^2}\log(1-z) - \frac{z}{(1-z)^2}\right) + c$$

Davon interessiert uns die Potenzreihenentwicklung. Es gilt

$$\begin{aligned}
\frac{-1}{(1-z)^2}\log(1-z) &= \sum_{i\geq 0}(i+1)\,z^i \cdot \sum_{j>0}\frac{z^j}{j} \\
&= \sum_{n>0}\left(\sum_{j=1}^{n}(n-j+1)\,\frac{1}{j}\right)\,z^n \\
&= \sum_{n>0}\left((n+1)H_n - n\right)\,z^n
\end{aligned}$$

und daher kann man schreiben:

$$\phi(z) = \sum_{n\geq 0} F(n)\,z^n = 2\left(\sum_{n>0}\left((n+1)\,H_n - n\right)\,z^n - \sum_{n>0} n\,z^n\right) + c$$

was per Koeffizientenvergleich zu

$$\Rightarrow F(n) = 2\,(n+1)H_n - 4\,n \quad (n>0)$$

führt.

8 Empfohlene Literatur

1. C.A.R. Hoare, Quicksort,
   Computer Journal 5 (1962), 10-15.

2. R. Sedgewick, *Quicksort*,
   ACM Outstanding Dissertations in Computer Science (Stanford, 1975),
   Garland, 1980.
   14GI/mat 8.7-50[492

3. R. Sedgewick, Implementing quicksort programs,
   Communications of the ACM 21 (1978), 847-857.

4. R. Sedgewick, Quicksort,
   Kapitel 9 in *Algorithms*.

5. Cormen/Leiserson/Rivest, Quicksort,
   Kapitel 8. in *Introduction to Algorithms*.

6. J. Bentley, Kolumne 10 in *Communications of the ACM*, 27 (1984).
   ist abgedruckt in:

7. J. Bentley, *Programming Pearls*,
   Addison-Wesley, 1986/89.
   14GI/mat 12.2-83

The average number of splitters in a random permutation

If $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a permutation of $[n] := \{1, 2, \ldots, n\}$, then we say that "$\sigma$ splits at j" or "j is a splitter in σ" if $\sigma_i < \sigma_j$ for all $i < j$ and $\sigma_j < \sigma_k$ for all $j < k$ (so that necessarily $s_j = j$). Splitting properties of random permutations are of interest because of the appearance of that concept in the *quicksort* algorithm (see e.g. section 2.2 in H. Wilfs book *Algorithms and Complexity*, Prentice-Hall, 1986).

In this note I give a very short proof of the fact that the average number of splitters in a random permutation of n elements behaves as $2/n$ for large n.

- Counting splitters is easy: j appears as a splitter in precisely $(j-1)!\,(n-j)!$ permutations of $[n]$, so that there is a total of

$$s_n := \sum_{j=0}^{n} (j-1)!\,(n-j)! = (n-1)! \sum_{j=0}^{n-1} \binom{n-1}{j}^{-1}$$

  splitters in permutations of $[n]$.

- In order to deal with the reciprocals of binomial coefficients it is useful to remember (the β-integral)

$$\frac{1}{a+b+1}\binom{a+b}{a}^{-1} = \int_0^1 t^a\,(1-t)^b dt$$

  for integers $a, b \geq 0$, which can be proved by simple induction using partial integration.

- Let us now consider

$$b_n := \sum_{k=0}^{n} \binom{n}{k}^{-1} = (n+1) \int_0^1 \sum_{k=0}^{n} t^k (1-t)^{n-k} dt = (n+1)\, a_n$$

  Note that the a_n satisfy a very simple recursion:

$$a_{n+1} = \frac{a_n}{2} + \frac{1}{n+2} \quad , \quad a_0 = 1$$

  This can be seen as follows:

$$\begin{aligned} a_{n+1} - \frac{a_n}{2} &= \int_0^1 t^{n+1} dt + \int_0^1 \sum_{k=0}^{n} \left[t^k (1-t)^{n+1-k} - \frac{1}{2} t^k (1-t)^{n-k} \right] dt \\ &= \frac{1}{n+2} + \int_0^1 \left[\sum_{k=0}^{n} t^k (1-t)^{n-k} \right] \left(\frac{1}{2} - t \right) dt \end{aligned}$$

  where the last integral vanishes for reasons of symmetry!

- Now $\overline{s}_n = s_n/n! = b_{n-1}/n = a_{n-1}$, the average number of splitters in permutations of $[n]$, satisfies the recursion

$$\overline{s}_{n+1} = \frac{\overline{s}_n}{2} + \frac{1}{n+1} \quad , \quad \overline{s}_1 = 1$$

  from which it follows immediately that $n \cdot \overline{s}_n \to_{n\to\infty} 2$.

V. Strehl, june 15, 1994.

Splitter

- $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ Permutation von $\{1, 2, \ldots, n\}$
- σ *splittet an Position* j oder j ist ein *Splitter* in σ, wenn

$$\sigma_i < \sigma_j \text{ für } i < j \text{ und } \sigma_j < \sigma_k \text{ für } j < k$$

  Splitter sind (spezielle) Fixpunkte, d.h. $s_j = j$.
- Beispiele
  - $(3, 1, 2, 4, 7, 5, 6)$ hat 4 als Splitter
  - $(5, 1, 2, 4, 7, 3, 6)$ hat keinen Splitter
  - $(1, 2, 3, 4, 5, 6, 7)$ hat alle 7 Positionen als Splitter
- Problem: wieviele Splitter hat eine Permutation *im Mittel*?

Die Gesamtzahl der Splitter in $\mathcal{S}_n$

- Die Position j ist ein Splitter in $(j-1)! \cdot (n-j)!$ Permutationen aus $\mathcal{S}_n$.
- Die Gesamtzahl s_n der Splitter in $\mathcal{S}_n$

$$s_n := \sum_{j=1}^{n} (j-1)!\,(n-j)! = (n-1)! \cdot \sum_{j=0}^{n-1} \binom{n-1}{j}^{-1}$$

- Wie geht man mit folgender Summe um?

$$\frac{1}{\binom{n}{0}} + \frac{1}{\binom{n}{1}} + \cdots \frac{1}{\binom{n}{n}}$$

- Hier hilft das *Beta-Integral*

$$\frac{1}{a+b+1} \binom{a+b}{a}^{-1} = \int_0^1 t^a (1-t)^b dt$$

- Mit Beta-Integral ergibt sich

$$b_n := \sum_{k=0}^{n} \binom{n}{k}^{-1} = (n+1) \underbrace{\int_0^1 \sum_{k=0}^{n} t^k (1-t)^{n-k} dt}_{a_n}$$

- Die Zahlen n_n genügen einer einfachen Rekursion:

$$a_{n+1} = \frac{a_n}{2} + \frac{1}{n+2} \quad , \quad a_0 = 1$$

  Beweis:

$$\begin{aligned} a_{n+1} - \frac{a_n}{2} &= \int_0^1 t^{n+1} dt + \int_0^1 \sum_{k=0}^{n} \left[t^k (1-t)^{n+1-k} - \frac{1}{2} t^k (1-t)^{n-k} \right] dt \\ &= \frac{1}{n+2} + \int_0^1 \left[\sum_{k=0}^{n} t^k (1-t)^{n-k} \right] \left(\frac{1}{2} - t \right) dt \end{aligned}$$

Folgerung

- mittlere Anzahl von Splittern in $\mathcal{S}_n$

$$\overline{s}_n = \frac{s_n}{n!} = \frac{b_{n-1}}{n} = a_{n-1}$$

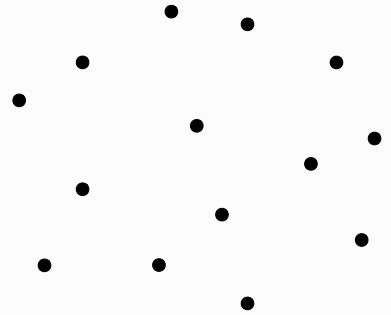
- Rekursion für $\overline{s}_n$

$$\overline{s}_{n+1} = \frac{\overline{s}_n}{2} + \frac{1}{n+1} \quad , \quad \overline{s}_1 = 1$$

- Daraus ergibt sich

$$\lim_{n \to \infty} n \cdot \overline{s}_n = 2, \quad \text{also} \quad \overline{s}_n \sim \frac{2}{n}$$

Das Closest-Pair-Problem



Literaturhinweis: Th. Ottmann, *Das Divide-and-Conquer-Prinzip*,
in *Prinzipien des Algorithmenentwurfs*, Spektrum Akademischer Verlag 1997.

P endliche Menge von Punkten $p = (p_x, p_y) \in \mathbb{R}^2$

$\texttt{dist}(p, q)$: euklidischer Abstand von $p, q \in \mathbb{R}^2$

Gesucht

$$\delta(P) = \min \{\, \texttt{dist}(p, q) \,;\, p, q \in P,\, p \neq q \,\}$$

allgemeiner für endliche $P, Q \subseteq \mathbb{R}^2$

$$\delta(P, Q) = \min \{\, \texttt{dist}(p, q) \,;\, p \in P,\, q \in Q,\, p \neq q \,\}$$

also $\delta(P) = \delta(P, P)$

Algorithmisches Problem

- berechne $\delta(P)$ möglichst effizient
  (evtl. auch: bestimme ein Paar $p, q \in P$ mit $\texttt{dist}(p, q) = \delta(P)$)

Naive Lösung der Aufgabe

- berechne alle paarweisen Distanzen

$$\{\, \texttt{dist}(p, q) \,;\, p, q \in P,\, p \neq q \,\}$$

- berechne das Minimum dieser Menge

Komplexität

- für $\sharp P = n$: $\binom{n}{2}$ Distanzen berechnen: $\mathcal{O}(n^2)$
- Minimum in $\mathcal{O}(n^2)$ Zahlen suchen: $\mathcal{O}(n^2)$ Vergleiche
- insgesamt ein $\mathcal{O}(n^2)$-Verfahren

Divide-and-Conquer Ansatz

- Zerlege P in zwei (ungefähr) gleich grosse Teile P_ℓ, P_r
- berechne (rekursiv) $\delta(P_\ell), \delta(P_r)$
- berechne $\delta(P_\ell, P_r)$
- berechne $\delta(P) = \min(\delta(P_\ell), \delta(P_r), \delta(P_\ell, P_r))$

Dieser Ansatz führt — bei naivem Vorgehen zur Berechnung von $\delta(P_\ell, P_r)$ — auf eine Rekursion für den Aufwand $t(n)$ an Operationen (bei $\sharp P = n$)

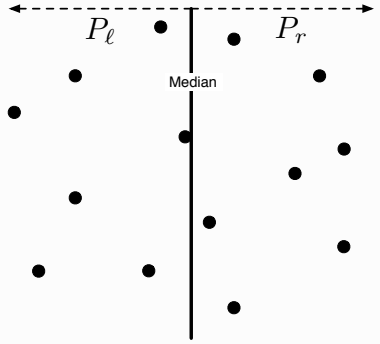
$$t(n) = 2 \cdot t(n/2) + 2 \cdot \left(\frac{n}{2}\right)^2 + 1$$
$$t(2) = 1$$

mit einer Lösung $t(n) \in \mathcal{O}(n^2)$

Geometrisches Divide-and-Conquer

- Zerlege P mit Hilfe des Medians m der x-Koordinaten der $p \in P$ in zwei (ungefähr) gleich grosse Teile

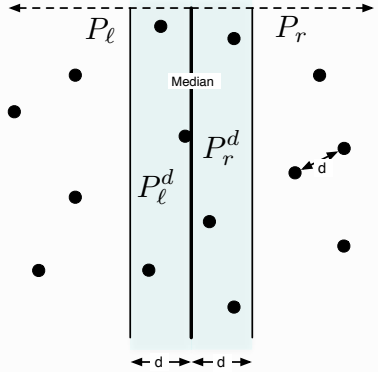
$$P_\ell = \{\, p \in P \,;\, p_x \le m \,\} \qquad P_r = \{\, p \in P \,;\, m < p_x \,\}$$



- Bemerkung: Median-Berechnung für n Elemente geht in $\mathcal{O}(n)$
- berechne (rekursiv) $\delta(P_\ell), \delta(P_r)$ und $d = \min(\delta(P_\ell), \delta(P_r))$

- bestimme

$$P_\ell^d = \{\, p \in P \,;\, m - d \le p_x \le m \,\} \qquad P_r^d = \{\, p \in P \,;\, m < p_x \le m + d \,\}$$

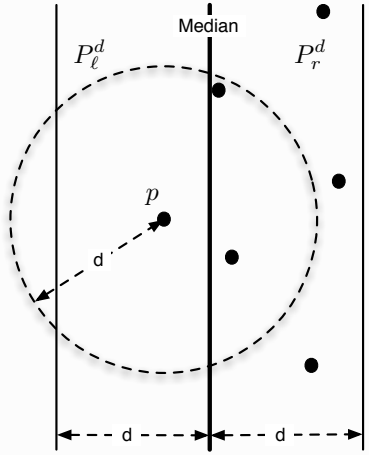


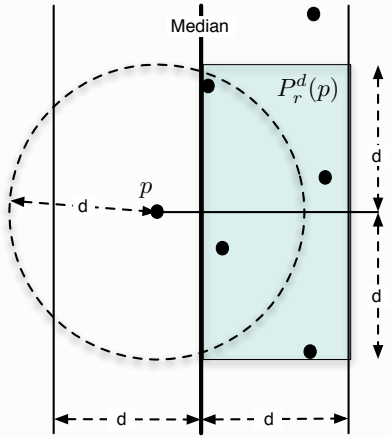
- berechne $\delta(P_\ell^d, P_r^d)$ und somit

$$\delta(P) = \min(d, \delta(P_\ell, P_r)) = \min(d, \delta(P_\ell^d, P_r^d))$$

Vorsicht: auch P_ℓ^d und P_r^d können noch $\mathcal{O}(n)$ Elemente enthalten!

ABER: zu jedem $p \in P_\ell^d$ kann es nur sehr wenige $q \in P_r^d$ geben, die "nahe" ($\approx d$) von p liegen:





$$\texttt{dist}(p,q) < d \;\Rightarrow\; p_y - d < q_y < p_y + d$$

$$\Rightarrow\; q \in \underbrace{P_r^d \cap \{\, q \,;\, p_y - d < q_y < p_y + d \,\}}_{P_r^d(p)}$$

$$\delta(P_\ell, P_r) = \delta(P_\ell^d, P_r^d)$$
$$= \min_{p \in P_\ell^d} \delta(p, P_r^d) = \min_{p \in P_\ell^d} \delta(p, P_r^d(p))$$

Aus der Geometrie folgt (grosszügig abgeschätzt!)

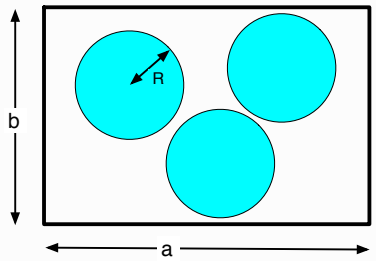
$$\sharp P_r^d(p) \leq 7$$

d.h. zu jedem $p \in P_\ell^d$ gibt es höchstens sieben $q \in P_r^d$ mit $\texttt{dist}(p,q) < d$,

d.h. man muss bei geschickter (!!) Organisation der Daten nur $\mathcal{O}(n)$ Operationen durchführen, um $\delta(P_\ell, P_r)$ zu berechnen!

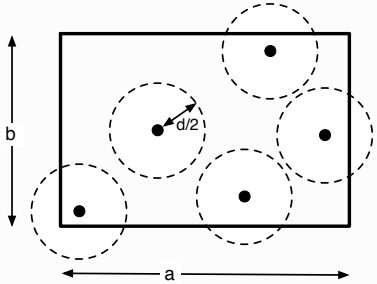
Eine geometrische Überlegung

- Wieviele disjunkte, offene Kreisscheiben mit Radius R kann man in einem $(a \times b)$-Rechteck unterbringen?



Anzahl $k(a,b,R) \leq \dfrac{a \cdot b}{\pi \cdot R^2}$

- Wieviele verschiedene Punkte, die paarweise einen Abstand $\geq d$ haben, kann man in einem $(a \times b)$-Rechteck unterbringen?



Anzahl $\leq k(a+d, b+d, d/2) \leq \dfrac{(a+d)(b+d)}{\pi \cdot (d/2)^2}$

P_ℓ^d nach y-Koordinaten sortiert

$$P_\ell^d \;:\; p^{(1)} \quad p^{(2)} \quad p^{(3)} \quad \ldots \quad p^{(s)}$$
$$p_y^{(1)} \leq \; p_y^{(2)} \leq \; p_y^{(3)} \leq \; \ldots \; < p_y^{(s)}$$

P_r^d nach y-Koordinaten sortiert

$$P_r^d \;:\; q^{(1)} \quad q^{(2)} \quad q^{(3)} \quad \ldots \quad q^{(t)}$$
$$q_y^{(1)} \leq \; q_y^{(2)} \leq \; q_y^{(3)} \leq \; \ldots \; < q_y^{(t)}$$

Kandidaten für $\texttt{dist}(p,q)$ in Blöcken der Länge ≤ 7

$$P_\ell^d \;:\; p^{(1)} \quad p^{(2)} \quad p^{(3)} \quad \ldots \quad \boxed{p} \quad \ldots \quad p^{(s)}$$
$$P_r^d \;:\; q^{(1)} \quad q^{(2)} \quad q^{(3)} \quad \ldots \quad \underbrace{\boxed{q^{(i)} \; q^{(i+1)} \; \ldots \; q^{(i+j)}}}_{P_\ell^d(p)} \quad \ldots \quad q^{(s)}$$

cp-Algorithmus

- Zerlege P mit Hilfe des Medians m der x-Koordinaten der $p \in P$ in zwei (ungefähr) gleich grosse Teile

$$P_\ell = \{\, p \in P \,;\, p_x \leq m \,\} \qquad P_r = \{\, p \in P \,;\, m < p_x \,\}$$

- berechne (rekursiv) $\delta(P_\ell), \delta(P_r)$ und $d = \min(\delta(P_\ell), \delta(P_r))$
- bestimme

$$P_\ell^d = \{\, p \in P \,;\, m - d \leq p_x \leq m \,\} \quad P_r^d = \{\, p \in P \,;\, m < p_x \leq m + d \,\}$$

- sortiere die Mengen P_ℓ^d und P_r^d nach den y-Koordinaten ihrer Elemente
- berechne $\delta(P_\ell^d, P_r^d)$ mit $\mathcal{O}(n)$ Operationen
- $\delta(P) = \min(d, \delta(P_\ell^d, P_r^d))$

Komplexität $t(n)$ des cp-Algorithmus

- Medianberechnung und Zerlegung $P = P_\ell \uplus P_r$: $\mathcal{O}(n)$
- Herausziehen von P_ℓ^d und P_r^d : $\mathcal{O}(n)$
- Sortieren von P_ℓ^d und P_r^d : $\mathcal{O}(n \log n)$
- Berechnen von $\delta(P_\ell^d, P_r^d)$: $\mathcal{O}(n)$
- divide-and-conquer Rekursionsgleichung

$$t(n) = 2\, t(n/2) + \mathcal{O}(n \log n)$$
$$t(2) = 1$$

- Verhalten der Lösung der Rekursionsgleichung

$$t(n) \in \mathcal{O}(n \log^2 n)$$

- Man kann das Sortieren auf jeder Rekursionsstufe durch einmaliges Sortieren zu Beginn und geschickte Verwaltung dieser Information ersetzen, dadurch sogar Gesamtkomplexität $t(n) \in \mathcal{O}(n \log n)$ erreichen

Zwei *divide-and-conquer* Algorithmen der Arithmetik

- KARATSUBA-Multiplikation von Polynomen (und von ganzen Zahlen)

Multiplikation zweier Polynome vom Grad $\deg f, \deg g < 2n$

$$f(x) = \sum_{i=0}^{2n-1} f_i\, x^i$$

$$g(x) = \sum_{j=0}^{2n-1} g_j\, x^j$$

$$(f * g)(x) = \sum_{k=0}^{2n-2} \left(\sum_{0 \le i \le k} f_i \cdot g_{k-i} \right) x^k$$

wird zurückgeführt auf drei Multiplikationen von Polynomen vom Grad $< n$

$$\begin{aligned} f(x) &= a(x) + x^n\, b(x) \\ g(x) &= c(x) + x^n\, d(x) \\ f(x)g(x) &= a(x)c(x) + x^n\,(a(x)d(x) + b(x)c(x)) + x^{2n} b(x)d(x) \\ &= u(x) + x^n\,(w(x) - u(x) - v(x)) + x^{2n} v(x) \end{aligned}$$

wobei

$$\begin{aligned} u(x) &:= a(x)c(x) \\ v(x) &:= b(x)d(x) \\ w(x) &:= (a(x) + b(x))(c(x) + d(x)) \end{aligned}$$

Diese Idee ist rekursiv anzuwenden, bis man Polynome vom Grad 0 (Konstanten) oder Polynome eines kleinen Grades zu multiplizieren hat.

Die Idee überträgt sich wörtlich auf die Multiplikation ganzer Zahlen (Übertrag beachten!)

```
karatsuba := proc(f,g,x,m)
local a,b,c,d,u,v,w,deg;
if m=0 then RETURN(f*g) fi;
deg := 2^(m-1);
a := sum('coeff(f,x,i)*x^i','i'=0..deg-1);
b := sum('coeff(f,x,i+deg)*x^i','i'=0..deg-1);
c := sum('coeff(g,x,i)*x^i','i'=0..deg-1);
d := sum('coeff(g,x,i+deg)*x^i','i'=0..deg-1);
u := karatsuba(a,c,x,m-1);
v := karatsuba(b,d,x,m-1);
w := karatsuba(a+b,c+d,x,m-1);
RETURN(expand(u+(w-u-v)*x^deg+v*x^(2*deg)));
end;
kara_mult := proc(f,g,var)
local df,dg,bound;
df := degree(f,var);
dg := degree(g,var);
bound := ceil(simplify(log[2](max(df,dg)+1)));
karatsuba(f,g,var,bound);
end;
```

Komplexitätsanalyse (für $n = 2^m$):

$$t_K(n) := \begin{cases} \text{(worst-case) Anzahl der Additionen und Multiplikationen} \\ \text{im Koeffizientenbereich} \\ \text{bei input-Grad} < n \end{cases}$$

$$t_K(2n) \le 3 \cdot t_K(n) + 8n \quad , \quad t(1) = 1$$

Lösung der divide-and-conquer-Rekursion

$$t(2n) = 3 \cdot t(n) + \Theta(n) \quad , \quad t(1) = \Theta(1)$$

führt zu

$$\Rightarrow \boxed{t_K(n) \in \Theta(n^{\log_2 3})}$$

Beachte: $\log_2 3 = 1.584962501\ldots$

- Siehe Heun, GA, Abschnitt 7.6. für eine genaue Analyse im Fall der Multiplikation ganzer Zahlen (Theorem 7.45)

$$t_K(n) \leq 41 \cdot n^{\log 3}$$

und Optimierung des Rekursionsabbruchs (Theorem 7.46)

$$t_K(n) \leq 11 \cdot n^{\log 3}$$

- Sei der Arbeit von

A. A. Karatsuba, Y. P. Ofman, Multiplication of multidigit numbers on automata, *Dokl. Acad. Nauk SSR* (145 (1962), 293–294

ist es gelungen, für die Komplexität der Multiplikation ganzer Zahlen eine obere Schranke von $\mathcal{O}(n \log n \log\log n)$ zu finden:

A. Schönhage, V. Strassen, Schnelle Multiplikation grosser Zahlen, *Computing* 7 (1971), 281–292,

basierend auf der Technik der Schnellen Fourier-Transformation (FFT).

- Siehe D. E. Knuth, TAOCP vol. 2 für Details.

- Strassens Matrix-Multiplikation

Ausgangspunkt: die Multiplikation von zwei (2×2)-Matrizen läßt sich mit 7 Multiplikationen im Koeffizientenbereich ausführen — statt mit 8 Multiplikationen nach "Schulmethode":

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$
$$= \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix}$$

Wichtig: das gilt über jedem Ring!

Man berechnet zunächst 7 Produkte

$$\begin{aligned} c_1 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\ c_2 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ c_3 &= (a_{11} - a_{21})(b_{11} + b_{12}) \\ c_4 &= (a_{11} + a_{12})\, b_{22} \\ c_5 &= a_{11}\, (b_{12} - b_{22}) \\ c_6 &= a_{22}\, (b_{21} - b_{11}) \\ c_7 &= (a_{21} + a_{22})\, b_{11} \end{aligned}$$

und danach die $d_{11}, \ldots, d_{22}$ durch

$$d_{11} = c_1 + c_2 - c_4 + c_6 \quad d_{12} = c_4 + c_5$$
$$d_{21} = c_6 + c_7 \quad d_{22} = c_2 - c_3 + c_5 - c_7$$

Der Witz der (rekursiven) Angelegenheit: die Koeffizienten $a_{ij}, b_{i,j}, \ldots$ können selbst wieder Matrizen sein, d.h.

Mit Strassens Idee erreicht man:

> Reduktion einer Matrixmultiplikation für zwei $(2n \times 2n)$-Matrizen auf 7 Matrixmultiplikationen von $(n \times n)$-Matrizen, dazu 18 Additionen/Subtraktionen von $(n \times n)$-Matrizen

```
strassen := proc(A::matrix,B::matrix,m)
local dim,A11,A12,A21,A22,B11,B12,B21,B22,
      C1,C2,C3,C4,C5,C6,C7,D11,D12,D21,D22;
if m=0 then RETURN(evalm(A&*B)) fi;
dim := 2^(m-1);
A11 := submatrix(A,1..dim,1..dim);
B22 := submatrix(B,dim+1..2*dim,dim+1..2*dim); ...(etc.)
C1 := strassen(evalm(A12-A22),evalm(B21+B22),m-1);
C2 := strassen(evalm(A11+A22),evalm(B11+B22),m-1);
C3 := strassen(evalm(A11-A21),evalm(B11+B12),m-1);
C4 := strassen(evalm(A11+A12),B22,m-1);
C5 := strassen(A11,evalm(B12-B22),m-1);
C6 := strassen(A22,evalm(B21-B11),m-1);
C7 := strassen(evalm(A21+A22),B11,m-1);
D11 := evalm(C1+C2-C4+C6);
D12 := evalm(C4+C5);
D21 := evalm(C6+C7);
D22 := evalm(C2-C3+C5-C7);
RETURN(stackmatrix(concat(D11,D12),concat(D21,D22)));
end;
```

Komplexitätsanalyse (für $n = 2^m$):

$$t_S(n) := \begin{cases}\text{(maximale) Anzahl der arithmetischen Operationen im Koeffizi-}\\ \text{entenbereich für die Multiplikation von zwei } (n \times n)\text{-Matrizen}\end{cases}$$

$$t_S(2n) = 7 \cdot t(n) + 18 \cdot n^2 \quad , \quad t_S(1) = 1$$

$$\Rightarrow \boxed{t_S(n) \in \Theta(n^{\log_2 7})}$$

Beachte: $\log_2 7 = 2.81\ldots$!

Traditionelle Matrixmultiplikation benötigt für die Multiplikation von zwei $(n \times n)$-Matrizen

$n^2 \cdot n$ Muliplikationen und $n^2 \cdot (n-1)$ Additionen von Koeffizienten

ist also ein $\mathcal{O}(n^3)$-Verfahren.

Seit

V. Strassen, Gaussian elimination is not optimal,
Numerische Mathematik 13 (1969), 354–356

ist es gelungen, Verfahren bis zu $\mathcal{O}(n^{2.38\ldots})$ zu finden (Coppersmith, Winograd, 1990)

- Siehe Abschnitt 7.8 in Heun, GA, für eine genaue Analyse (Theorem 7.50)

  $$t_S(n) \leq \frac{141}{5} \cdot n^{\log 7}$$

  und Optimierung des Abbruchpunktes für die Rekursion (Theorem 7.51)

  $$t_S(n) \leq 4.62 \cdot n^{\log 7}$$

- Siehe Kapitel 31.2 in Cormen/Leiserson/Rivest für einen Rekonstruktionsversuch der Strassenschen Idee.

Divide-and-Conquer-Algorithmen

Fundamentales Prinzip des Problemlösens

- *Divide* : Zerlege das zu lösende Problem in (ein oder) mehrere kleinere Teilprobleme gleichen Typs
- *Recur* : Falls Problem "elementar": löse dieses mit spezieller Methode
  Falls Problem nicht "elementar": wende *Divide* rekursiv an
- *Conquer* : Konstruiere Lösung des Problems aus den Lösungen der Teilprobleme
- Allgemein: sowohl die Kosten von *Divide* wie von *Conquer* sind relevant
- wichtige Parameter: Anzahl und Grösse der jeweils erzeugten Teilprobleme

Beispiele: Mergesort, Quicksort, Karatsuba-Multiplikation, Strassen-Multiplikation, closest-pair Algorithmus, ...

Bemerkung:

- Mergesort: *Divide* ist simpel, die eigentliche Arbeit liegt im *Conquer* (`merge`)
- Quicksort: *Divide* (`partition`) ist komplex, *Conquer* ist simpel
- Karatsuba: *Divide* ist simpel, *Conquer* erfordet Arbeit
- Strassen: *Divide* ist simpel, *Conquer* erfordet Arbeit
- closest pair: *Divide* und *Conquer* erfordern Arbeit

Kostenbilanz bei divide-and-conquer-Algorithmen

- Mergesort: Vergleichsoperationen

$$V^{\text{rek}}_{\text{merge}}(n) = V^{\text{rek}}_{\text{merge}}(\lceil n/2 \rceil) + V^{\text{rek}}_{\text{merge}}(\lfloor n/2 \rfloor) + n - 1 \quad (n > 1), \quad V^{\text{rek}}_{\text{merge}}(1) = 0$$

- Quicksort: Vergleichsoperationen (im Mittel)

$$\overline{V}_{\texttt{quick}}(n) = (n-1) + \frac{1}{n} \sum_{k=0}^{n-1} \overline{V}_{\texttt{quick}}(k) \quad (n \geq 2), \quad \overline{V}_{\texttt{quick}}(1) = 0$$

- Karatsuba: Multiplikationen und Additionen von Koeffizienten

$$t_K(2n) \leq 3 \cdot t_K(n) + 8 \cdot n \quad (n \geq 1), \quad t_K(1) = 1$$

- Strassen: arithmetische Operationen mit Koeffizienten

$$t_S(2n) = 7 \cdot t(n) + 18 \cdot n^2 \quad (n \geq 1), \quad t_S(1) = 1$$

- closest pair: Vergleichsoperationen und arithmetische Operationen

$$t_{cp}(n) = 2 \cdot t_{cp}(n/2) + \mathcal{O}(n \log n) \quad (n > 2), \quad t_{cp}(2) = 1$$

Kostenbilanz bei (statischen) divide-and-conquer-Algorithmen

- bei Problemgrösse n sei
  - $Div(n)$: Kosten für *Divide*,
    $Con(n)$: Kosten für *Conquer*
  - a_n : Anzahl der entstandenen Teilprobleme,
    $(n_i)_{1 \leq i \leq a_n}$ Grössen der Teilprobleme
  - $\mathcal{C}(n)$: Gesamtkosten
- Bilanz:

$$\mathcal{C}(n) = O(1) \qquad (n \leq n_0)$$

$$\mathcal{C}(n) = Div(n) + \sum_{i=1}^{a_n} \mathcal{C}(n_i) + Con(n) \qquad (n > n_0)$$

Explizite Lösung bei dieser Allgemeinheit schwierig.

Betrachten hier nur eingeschränkte Fälle.

Vereinfachende Annahmen:

- aus Problemen der Grösse n entstehen bei *Divide* jeweils a Teilprobleme der Größe n/b, wobei $b > 1$ konstant
- auftretende Argumente n sind Potenzen von b, d.h. $n = b^m$ für $m \geq 0$
- Zerlegung wird nicht vorzeitig abgebrochen: $n_0 = 1$

Zu lösende Rekursion

$$\begin{aligned} \mathcal{C}(1) &= d \\ \mathcal{C}(n) &= a \cdot \mathcal{C}(n/b) + \underbrace{Div(n) + Con(n)}_{f(n)} \qquad (n > 1) \end{aligned}$$

Mit den Abkürzungen $c_m = \mathcal{C}(b^m)$, $f_m = f(b^m)$ $(m \geq 0)$, wobei

$$\begin{aligned} c_0 &= f_0 = d \\ c_m &= a \cdot c_{m-1} + f_m \quad (m > 0) \end{aligned}$$

(inhomogene lineare Rekursion 1. Ordnung)

Lösung (durch Iteration/Induktion)

$$\begin{aligned} c_m &= a \cdot c_{m-1} + f_m \\ &= a \cdot (a \cdot c_{m-2} + f_{m-1}) + f_m \\ &= a \cdot (a \cdot (a \cdot c_{m-3} + f_{m-2}) + f_{m-1}) + f_m \\ &\vdots \\ &= a^j \cdot c_{m-j} + \sum_{k=0}^{j-1} a^k f_{m-k} \quad (j = 1, 2, 3, \ldots, m) \\ &= d \cdot a^m + \sum_{k=0}^{m-1} a^k f_{m-k} \quad (m \geq 0) \end{aligned}$$

in der ursprünglichen Notation

$$\mathcal{C}(n) = d \cdot n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} a^k \cdot f(\frac{n}{b^k})$$

Spezialfall: lineare overhead-Funktion $f(n) = c \cdot n$

Für $n = b^m$ gilt dann

(beachte $m = \log_b n \quad \Rightarrow \quad a^m = n^{\log_b a}, \ \left(\frac{a}{b}\right)^m = n^{\log_b a - 1}$)

$$\begin{aligned} \mathcal{C}(n) = c_m &= d \cdot a^m + \sum_{k=0}^{m-1} a^k \cdot f_{m-k} \\ &= d \cdot a^m + \sum_{k=0}^{m-1} a^k \cdot c \cdot b^{m-k} \\ &= \underbrace{d \cdot n^{\log_b a}}_{(I)} + \underbrace{c \cdot n \sum_{k=0}^{m-1} \left(\frac{a}{b}\right)^k}_{(II)} \end{aligned}$$

Lösungstypen für

$$\mathcal{C}(n) = \underbrace{d \cdot n^{\log_b a}}_{(I)} + \underbrace{c \cdot n \sum_{k=0}^{m-1} \left(\frac{a}{b}\right)^k}_{(II)}$$

$$\begin{aligned} a < b &\ \Rightarrow\ (I) \in o(n) && (II) \in \Theta(n) && \Rightarrow\ \mathcal{C}(n) \in \Theta(n) \\ a = b &\ \Rightarrow\ (I) \in \Theta(n) && (II) \in \Theta(n \log n) && \Rightarrow\ \mathcal{C}(n) \in \Theta(n \log n) \\ a > b &\ \Rightarrow\ (I) \in \Theta(n^{\log_b a}) && (II) \in \Theta(n^{\log_b a)}) && \Rightarrow\ \mathcal{C}(n) \in \Theta(n^{\log_b a}) \end{aligned}$$

Also:

$$\mathcal{C}(n) \in \begin{cases} \Theta(n) & \text{falls } a < b \\ \Theta(n \log n) & \text{falls } a = b \\ \Theta(n^{\log_b a}) & \text{falls } a > b \end{cases}$$

Ganz analog beweist man im Fall $f(n) = c \cdot n^\ell$ mit $c, \ell > 0$

$$\mathcal{C}(n) \in \begin{cases} \Theta(n^\ell) & \text{falls } a < b^\ell \\ \Theta(n^\ell \log n) & \text{falls } a = b^\ell \\ \Theta(n^{\log_b a}) & \text{falls } a > b^\ell \end{cases}$$

Diskussion weiterer spezieller Fälle

$$\mathcal{C}(n) = a \cdot \mathcal{C}(n/b) + f(n) \quad (n > 1), \quad \mathcal{C}(1) = f(1) > 0,$$

wobei b ganz und ≥ 2, $a > 0$, $f(n) \geq 0$.

Vereinfachung (wie vorher): für n nur Potenzen von b betrachten. (Genaue Begründung: CLR, Abschnitt 4.4.2)

Notation (wie vorher)

$$c_m = \mathcal{C}(b^m), f_m = f(b^m) \quad (m \geq 0)$$

Lösung der Rekursion

$$c_m = \sum_{k=0}^{m} f_k \cdot a^{m-k} \quad (m > 0), \quad c_0 = f_0 = f(1)$$

Fall $f(n) \in \Theta(n^\ell)$ und $a > b^\ell$

$$c_m = \sum_{k=0}^{m} f_k \cdot a^{m-k} = a^m \sum_{k=0}^{m} f_k \cdot \frac{1}{a^k}$$

Mit $f(n) \leq A \cdot n^\ell$, also $f_m \leq A \cdot b^{m\ell}$, ist

$$f_0 \leq \sum_{k=0}^{m} f_k \cdot \frac{1}{a^k} \leq A \cdot \sum_{k=0}^{m} \frac{b^{k\ell}}{a^k} \leq A \cdot \frac{a}{a - b^\ell}$$

und daher

$$c_m \in \Theta(a^m) \quad \text{d.h. } \mathcal{C}(n) = c_m \in \Theta(a^m) = \Theta(n^{\log_b a})$$

Fall $f(n) \in \Theta(n^\ell)$ und $a < b^\ell$

$$c_m = \sum_{k=0}^{m} f_k \cdot a^{m-k} = a^m \sum_{k=0}^{m} f_k \cdot \frac{1}{a^k}$$

Mit $f(n) \leq A \cdot n^\ell$, also $f_m \leq A \cdot b^{m\ell}$, ist

$$c_m \leq A \cdot b^{m\ell} \cdot \sum_{k=0}^{m} \left(\frac{a}{b^\ell}\right)^k < a \cdot b^{m\ell} \cdot \frac{b^\ell}{b^\ell - a}$$

also

$$c_m \in O((b^m)^\ell), \quad \text{d.h. } \mathcal{C}(n) \in O(n^\ell)$$

Wegen $c_m \geq f_m$ gilt sogar

$$\mathcal{C}(n) = c_m \in \Theta(n^\ell)$$

Fall $f(n) = B \cdot n^\ell \cdot (\log_b n)^q$ $(n > 1)$ und $a = b^\ell$

$$c_m = \sum_{k=0}^{m} f_k \cdot a^{m-k} = a^m \cdot \big(f(1) + B \sum_{k=1}^{m} k^q\big)$$

Unterfall $q < -1$

$\sum_{k=1}^{m} k^q$ konvergiert für $m \to \infty$, daher

$$c_m \in \Theta\big(a^m\big), \quad \text{d.h. } \mathcal{C}(n) \in \Theta\big(n^{\log_b a}\big)$$

Unterfall $q = -1$

$\sum_{k=1}^{m} k^q = \sum_{k=1}^{m} \frac{1}{k} = H_m \sim \log m$, daher

$$c_m \in \Theta\big(a^m \log m\big), \quad \text{d.h. } \mathcal{C}(n) \in \Theta\big(n^{\log_b a} \log\log_b n\big)$$

Unterfall $q > -1$

$\sum_{k=1}^{m} k^q \in \Theta\big(m^{q+1}\big)$, daher

$$c_m \in \Theta\big(a^m m^{q+1}\big), \quad \text{d.h. } \mathcal{C}(n) \in \Theta\big(n^{\log_b a} (\log_b n)^{1+q}\big)$$

Beispiele

$$\begin{aligned}
t(n) = t(n/2) + c &\Rightarrow t(n) \in \Theta(\log n)\\
t(n) = 2\,t(n/2) + c\,n &\Rightarrow t(n) \in \Theta(n \log n)\\
t(n) = 2\,t(n/2) + c\,n^2 &\Rightarrow t(n) \in \Theta(n^2)\\
t(n) = 4\,t(n/2) + c\,n^2 &\Rightarrow t(n) \in \Theta(n^2 \log n)\\
t(n) = 7\,t(n/2) + c\,n^2 &\Rightarrow t(n) \in \Theta(n^{\log_2 7})\\
t(n) = 2\,t(n/2) + \log n &\Rightarrow t(n) \in \Theta(n)\\
t(n) = 3\,t(n/2) + n \log n &\Rightarrow t(n) \in \Theta(n^{\log_2 3})\\
t(n) = 2\,t(n/2) + n \log n &\Rightarrow t(n) \in \Theta(n \log^2 n)\\
t(n) = 5\,t(n/2) + (n \log n)^2 &\Rightarrow t(n) \in \Theta(n^{\log_2 5})
\end{aligned}$$

Allgemeinerer Fall ("Master Theorem")
(Heun, GA Theorem 2.17,
Cormen, Leiserson, Rivest, Abschnitte 4.3 und 4.4, "master method")

$$\mathcal{C}(n) \in \begin{cases} \Theta(f(n)) & \text{falls } f(n) \in \Omega(n^{\log_b(a)+\epsilon}) \text{ und } a \cdot f(n/b) \le c \cdot f(n) \\ \Theta(n^{\log_b a} \log n) & \text{falls } f(n) \in \Theta(n^{\log_b a}) \\ \Theta(n^{\log_b a}) & \text{falls } f(n) \in O(n^{\log_b(a)-\epsilon}) \end{cases}$$

(wobei $\epsilon > 0$ und $c < 1$ konstant)

Analytischer Kommentar

- Aufgabe: Untersuchung des Wachstumsverhaltens einer Folge von Zahlen

$$(g_n)_{n \ge 0} = (g_0, g_1, g_2, \ldots)$$

- Transformation in analytisches Objekt (Potenzreihe)

$$g(z) = g_0 + g_1\,z + g_2\,z^2 + \cdots = \sum_{n=0}^{\infty} g_n z^n$$

- Konvergenzradius ρ_g:

$$g(z) = \sum_{n=0}^{\infty} g_n\,z^n \quad \begin{cases} \text{konvergiert für } |z| < \rho_g \\ \text{divergiert für } |z| > \rho_g \end{cases}$$

- Kriterium von Cauchy-Hadamard

$$\rho_g^{-1} = \limsup_{n \to \infty} \sqrt[n]{|g_n|}$$

- Äquivalente Formulierung: für alle $\epsilon > 0$

$$\left(\rho_g^{-1} - \epsilon\right)^n \overset{i.o.}{<} |g_n| \overset{a.e.}{<} \left(\rho_g^{-1} + \epsilon\right)^n$$

  i.o. = für unendlich viele n, *a.e.* = für fast alle n (endlich-viele Ausnahmen)

- Interpretation:

> das asymptotische Wachstum der Koeffizienten einer Potenzreihe $g(z)$ wird in seinem exponentiellen Verhalten bestimmt durch deren Konvergenzradius ρ_g.

- Für genauere Aussagen: es kommt auf den Funktionsverlauf von $g(z)$ in der Nähe der kleinsten "Singularität" an, die immer auf dem Konvergenzkreis $|z| = \rho_g$ liegt (in den uns interessierenden Fällen auch immer auf der positiven reellen Achse)

Beispiele

- Geometrische Reihe

$$(1, a, a^2, a^3, \ldots) \leftrightarrow g_a(z) = \sum_{n\geq 0} a^n\, z^n = \frac{1}{1 - a \cdot z}$$

  $\rightarrow \rho_{g_a} = 1/|a|$

- Quadrat der geometrischen Reihe

$$(1, 2a, 3a^2, 4a^3, \ldots) \leftrightarrow \sum_{n\geq 0} (n+1) a^n\, z^n = g_a(z)^2 = \frac{1}{(1 - a \cdot z)^2}$$

  $\rightarrow \rho_{g_a^2} = 1/|a|$

- Logarithmische Reihe

$$(0, a, a^2/2, a^3/3, a^4/4, \ldots) \leftrightarrow \sum_{n\geq 1} a^n\, z^n/n = -\log(1 - a \cdot z)$$

  $\rightarrow \rho_{\log} = 1/|a|$

- harmonische Zahlen

$$(0, 1, 3/2, 7/4, \ldots, H_n, \ldots) \leftrightarrow H(z) = \sum_{n\geq 1} H_n\, z^n = \frac{1}{1-z} \cdot \log \frac{1}{1-z}$$

  $\rightarrow \rho_H = 1$

- Fibonacci

$$(0, 1, 1, 2, 3, 5, 8, \ldots) \leftrightarrow F(z) = \sum_{n\geq 0} f_n\, z^n = \frac{z}{1 - z - z^2} = \frac{z}{(1 - \phi \cdot z)(1 - \widehat{\phi} \cdot z)}$$

  $\rightarrow \rho_F = 1/\phi = \phi - 1 = 0.62803\ldots$

- Mittlere Binomialkoeffizienten

$$(1, 2, 6, 20, \ldots) \leftrightarrow B(z) = \sum_{n\geq 0} \binom{2n}{n} z^n = \sqrt{1 - 4 \cdot z}$$

  $\rightarrow \rho_B = 1/4$, also

$$(4 - \epsilon)^n \overset{i.o.}{<} \binom{2n}{n} \overset{a.e.}{<} (4 + \epsilon)^n$$

  (genauere Aussage mittels Stirlings Formel)

Besonders wichtiges und instruktives Beispiel: binäre Bäume

- c_n = Anzahl der binären Bäume mit n inneren Knoten
- $C(z) = \sum_{n\geq 0} c_n\, z^n = 1 + z + 2z^2 + 5z^3 + 14z^4 + 42z^5 + \cdots$
- Wie schnell wächst c_n für $n \rightarrow \infty$?
- Rekursionsgleichung (folgt direkt aus der Definition)

$$c_{n+1} = \sum_{i=0}^{n} c_i \cdot c_{n-i} \quad (n \geq 0), c_0 = 1$$

- Äquivalente Funktionalgleichung

$$C(z) = 1 + z \cdot C(z)^2$$

- Lösung der quadratischen Gleichung

$$C(z) = \frac{1 - \sqrt{1 - 4z}}{2z}$$

– Konvergenzradius

$$\rho_C = 1/4$$

also

$$(4-\epsilon)^n \overset{i.o.}{<} c_n \overset{a.e.}{<} (4+\epsilon)^n$$

– genauere Aussage durch Reihenentwicklung mittels Newtons Binomialreihe (für $x \in \mathbb{R}$)

$$(1+z)^x = \sum_{n\geq 0} \binom{x}{n} z^n \qquad \text{wobei} \quad \binom{x}{n} = \frac{x(x-1)\cdots(x-n+1)}{n!}$$

– Catalans Formel (vgl. Heun, GA, Abschnitt 7.7.2)

$$C(z) = \frac{1-\sqrt{1-4z}}{2z} = \sum_{n\geq 0} \frac{1}{n+1}\binom{2n}{n} z^n \qquad \text{also} \quad c_n = \frac{1}{n+1}\binom{2n}{n}$$

$\Rightarrow$ Aufgabe 11.2 für asymptotische Aussage

Variation des Beispiels: unäre-binäre Bäume

– sind definiert wie binäre Bäume, aber innere Knoten können auch nur einen Nachfolger haben

– m_n = Anzahl der unären-binäre Bäume mit $n+1$ Knoten

– $(m_n)_{n\geq 0} = (1, 1, 2, 4, 9, 21, 51, \ldots)$

– es gibt keine "einfache Formel" für m_n

– für $M(z) = \sum_{n\geq 0} m_n\, z^n$ gilt wegen $M(z) = 1 + z \cdot M(z) + (z \cdot M(z))^2$

$$M(z) = \frac{1-z-\sqrt{1-2z-3z^2}}{2z^2}$$

– wegen $1-2z-3z^2 = (1+z)(1-3z)$ ist $\rho_M = 1/3$, also

$$(3-\epsilon)^n \overset{i.o.}{<} c_n \overset{a.e.}{<} (3+\epsilon)^n$$

– genauer gilt

$$m_n \sim 3^n \sqrt{\frac{3}{4\,\pi\, n^3}}$$

Die Betrachtung von (einfachen) divide-and-conquer-Situationen führt auf Rekursionen vom Typ

$$c_m = a \cdot c_{m-1} + f_m$$

Für die Potenzreihen

$$c(z) = \sum_{n\geq 0} c_n\, z^n \quad \text{und} \quad f(z) = \sum_{n\geq 0} f_n\, z^n$$

bedeutet das

$$c(z) = a \cdot z \cdot c(z) + f(z)$$

also

$$c(z) = \frac{1}{1-a\cdot z} \cdot f(z)$$

und Koeffizientenvergleich liefert die bekannte Formel

$$c_m = \sum_{k=0}^{m} a^{m-k} \cdot f_k$$

Für das Produkt

$$\sum_{\ell\geq 0} h_\ell\, z^\ell = h(z) = f(z)\cdot g(z) = \sum_{m\geq 0} f_m\, z^m \cdot \sum_{n\geq 0} g_n\, z^n$$

von Potenzreihen gilt

$$\rho_h = \min\{\rho_f, \rho_g\}$$

Also wird das exponentielle Wachstum von $(h_\ell)_{\ell\geq 0}$ bestimmt von

- ρ_f, falls $\rho_f < \rho_g$
- ρ_g, falls $\rho_f > \rho_g$
- $\rho_f (= \rho_g)$, falls $\rho_f = \rho_g$

Für die divide-and-conquer-Situation $c(z) = \frac{1}{1-a\,z} \cdot f(z)$ sind also prinzipiell drei Situationen möglich

- $\rho_f > 1/a$, also
$$\rho_c = 1/a \qquad \text{und somit } c_m \in \Theta(a^m)$$
Die overhead-Kosten haben gegenüber den Kosten für die a^m Elementarprobleme einen vernachlässigbaren Anteil
- $\rho_f < 1/a$
$$\rho_c = \rho_f \qquad \text{und somit } c_m \in \Theta(\rho_f^{-m})$$
Die overhead-Kosten haben gegenüber den Kosten für die a^m Elementarprobleme einen dominierenden Anteil
- $\rho_f = 1/a$
Dann ist $\rho_c = 1/a$, aber die overhead-Kosten sind erheblich und machen sich in einem subexponentiellen Faktor zu a^m bemerkbar.

Literaturhinweise

P. Flajolet, B. Salvy, P. Zimmermann, "Automatic average case analysis of algorithms", *Theoretical Computer Science*, Band 79 (1991), 37-109.

P. Flajolet, "Mathematical methods in the analysis of algorithms and data structures", in: E. Börger (Hrsg.), *Trends in Theoretical Computer Science*, Computer Science Press, 1988, 225-304.
Gruppenbibliothek Informatik: 14GI/mat 7.2-520.

J.S. Vitter, P. Flajolet, "Average case analysis of algorithms and data structures", Kapitel 9 im *Handbook of Theoretical Computer Science*, herausgegeben von J.v. Leeuwen, Elsevier, 1990, Band A, 432-524.
Gruppenbibliothek Informatik: 14GI/mat 7-31.

R. Sedgewick, P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.
Gruppenbibliothek Informatik: 14GI/mat 12.2.3–1004

1 Lösungstypen für die *divide-and-conquer*-Rekursion $t(n) = a \cdot t(n/b) + f(n)$

1.1 Vorbemerkung

Rekursiongleichungen dieses Typs werden in vielen Büchern über Komplexitätsanalyse behandelt. Besonders herauszuheben ist die sehr gründliche Abhandlung in den Abschnitten 4.3/4.4 von Cormen/Leiserson/Rivest, wo dies als "Master Method" bezeichnet wird. Dort wird allerdings der explizite Bezug auf den Konvergenzradius von Potenzreihen vermieden.

1.2 Die anschauliche Bedeutung solcher Rekursionen

In vielen Fällen führt die Analyse von *divide-and-conquer*-Algorithmen auf Rekursionsgleichungen des folgenden Typs:

$$t(n) = a \cdot t(n/b) + f(n) \quad , \quad t(1) = f(1) > 0 \tag{1}$$

Hierbei soll b ganz und ≥ 2 sein, $a > 0$ und $f(n) \geq 0$.

Die anschauliche Bedeutung dieser Rekursion: $t(n)$ beschreibt die Kosten für die Ausführung eines Algorithmus bei Problemgröße n. Probleminstanzen dieser Größe werden bearbeitet, indem sie in a Teilprobleme (gleichen Typs) der Größe n/b zerlegt werden (und dies Vorgehen wird rekursiv weitergeführt, bis Probleme der Größe 1 erreicht werden).[1] Mit $f(n)$ wird der *overhead* beschrieben, der bei dieser Zerlegung in Teilprobleme und der Synthese des Ergebnisses aus den Lösungen für die Teilprobleme benötigt wird (vgl. etwa die Prozedur *merge* bei *mergesort*).

Das Ziel der Untersuchung von Rekursionen wie (1) wird es sein, Informationen über das Wachstumsverhalten der Lösung $t(n)$ in Abhängigkeit von a, b und der *overhead*-Funktion $f(n)$ zu erhalten. Die Erfahrung lehrt, daß man oft über das Verhalten von $f(n)$ nur ungenaue Kenntnis hat, daß aber auch bei expliziter Kenntnis von $f(n)$ eine explizite Lösung für $t(n)$ nicht möglich ist, aus der man deren Wachstumsverhalten direkt ablesen könnte. Daher muß es das Ziel sein, auch aus ungefährer Kenntnis der Funktion

[1] In der Praxis ist es oft so, daß man diese rekursive Bearbeitung nicht bis zur kleinstmöglichen Problemgröße treibt, sondern vorher schon auf (nicht-rekursive) Algorithmen umschaltet, die für kleine Problemgrößen besser sind als das rekursive Verfahren. Für die Resultate, die hier interessieren, hat das keine Bedeutung.

$f(n)$, also beispielsweise über deren Wachstumsverhalten im Sinne einer Θ-Aussage, entsprechende Aussagen über die Lösungsfunktion $t(n)$ zu gewinnen.

Es macht zunächst nur Sinn, diese Rekursion (1) für Argumente zu betrachten, die Potenzen von b sind — allerdings bleiben alle Aussagen richtig, wenn $n \mapsto t(n)$ *monoton steigend* ist. Man setzt also

$$t_m := t(b^m) \quad , \quad f_m := f(b^m) \quad , \quad m = 0, 1, 2, 3, \ldots$$

und faßt diese Zahlen zusammen, indem man sie als Koeffizienten in Potenzreihen steckt :

$$\tau(z) := \sum_{m \geq 0} t_m \, z^m \quad , \quad \phi(z) := \sum_{m \geq 0} f_m \, z^m$$

1.3 Warum Potenzreihen?

Es ist eine fundamentale Einsicht der Analysis, daß ein enger Zusammenhang zwischen dem Konvergenzradius K_g einer Potenzreihe $g(z) = \sum_{n \geq 0} g_n \, z^n$ und dem Wachstumsverhalten der Koeffizientenfolge $(g_n)_{n \geq 0}$ besteht:

$$K_g^{-1} = \overline{\lim_{n \to \infty}} |g_n|^{1/n}$$

was man für unsere Zwecke besser so ausdrückt[2]:

> für alle $\epsilon > 0$ gilt
>
> $$\left(K_g^{-1} - \epsilon\right)^n <_{i.o.} |g_n| <_{a.e.} \left(K_g^{-1} + \epsilon\right)^n$$
>
> wobei die linke Ungleichung für unendlich viele n gilt (i.o. = *infinitely often*), während die rechte Ungleichung für fast alle n gilt (a.e. = *almost everywhere*).

Das heißt also: der reziproke Konvergenzradius K_g^{-1} beschreibt den *exponentiellen* Anteil am Wachstumsverhalten von $(g_n)_{n \geq 0}$. (N.B. Dies gilt auch, falls

[2] Die Bezeichnung $\overline{\lim}$ liest der Mathematiker als *limes superior* und schreibt dafür oft auch *lim sup*. Jede Folge $(x_n)_{n \geq 0}$ von reellen Zahlen hat einen (eindeutig bestimmten) *limes superior*, der allerdings auch unendlich sein kann. In unserer Situation entspricht das einem Konvergenzradius $K = 0$.

$K_g = 0$ oder $K_g = \infty$ ist: dann wächst $(g_n)_{n\geq 0}$ stärker bzw. schwächer als jede Exponentialfunktion.) Noch anders ausgedrückt:

$$\forall n \ : \ |g_n| = K_g^{-n} \cdot \tilde{g}_n$$

wobei $n \mapsto \tilde{g}_n$ eine Folge ist, die den *subexponentiellen* Anteil am Wachstumsverhalten von $(g_n)_{n\geq 0}$ beschreibt:

$$\forall \epsilon > 0 \ : \ (1-\epsilon)^n <_{i.o.} |\tilde{g}_n| <_{a.e} (1+\epsilon)^n$$

(d.h. die Potenzreihe $\tilde{g}(z) = \sum_n \tilde{g}_n z^n$ hat Konvergenzradius $K_{\tilde{g}} = 1$).

Eine leichte Rechnung zeigt, daß die Rekursionsbeziehung (1) äquivalent ist zu der funktionalen Beziehung

$$\tau(z) = \frac{1}{1-az} \cdot \phi(z) \tag{2}$$

und dies wiederum ist äquivalent zu der Summendarstellung

$$t_m = \sum_{i=0}^{m} f_i \, a^{m-i} \quad (m \geq 0) \tag{3}$$

1.4 Fallunterscheidung und Lösungstypen

Die funktionale Darstellung (2) sagt uns sofort, daß drei verschiedene Situationen auftreten können — beachte dabei, daß $1/a$ der Konvergenzradius der geometrischen Reihe $1/(1-az) = \sum_{n\geq 0}(az)^n$ ist:

- $K_\phi > 1/a$, d.h. es wird $K_\tau = \min(K_\phi, 1/a) = 1/a$ sein und das exponentielle Wachstum von $(t_m)_{m\geq 0}$ wird durch a^m bestimmt. Heuristisch bedeutet dies, daß die overhead-Kosten, ausgedrückt durch die Funktion $\phi(z)$, am Wachstum der Gesamtkosten nur einen subexponentiellen Anteil haben, also für große Probleminstanzen zu vernachlässigen sind. Beachte: a^m gibt die Anzahl der schließlich zu bearbeitenden Elementarprobleme der Größe 1 an, und die Gesamtkosten verhalten sich im wesentlichen proportional zu dieser Zahl.

- $K_\phi = 1/a$, d.h. es wird auch hier $K_\tau = \min(K_\phi, 1/a) = 1/a$ sein, und das exponentielle Wachstum von $(t_m)_{m\geq 0}$ wird also durch a^m bestimmt — allerdings mit dem Unterschied, daß hier auch die overhead-Kosten zu den Gesamtkosten wesentlich beitragen.

- $K_\phi < 1/a$, d.h. es wird $K_\tau = \min(K_\phi, 1/a) = K_\phi$ sein und das exponentielle Wachstum von $(t_m)_{m\geq 0}$ wird durch K_ϕ^{-m} bestimmt. Hier gehen die Gesamtkosten fast ausschließlich zu Lasten des overheads.

Nach diesem qualitativen Überblick nun die genauere Diskussion der Lösungstypen:

1. Der Fall $K_\phi > 1/a$:
   Dieser Fall tritt beispielsweise ein, wenn $f(n) \in \Theta(n^k)$ ist und $a > b^k$. In jedem Fall gilt:
   $$t_m = \sum_{i=0}^{m} f_i \, a^{m-i} = a^m \cdot \sum_{i=0}^{m} f_i \, \frac{1}{a^i}$$
   und es ist
   $$f_0 \leq \sum_{i=0}^{m} f_i \, \frac{1}{a^i} \leq \phi(\frac{1}{a}) \in \mathbf{R}$$
   da ja $\phi(z)$ für $z = 1/a$ konvergiert. Also gilt
   $$t_m \in \Theta(a^m)$$
   und, wenn man dies durch $n = b^m$ ausdrückt:
   $$t(n) = t(b^m) = t_m \in \Theta(a^m) = \Theta(n^{\log_b a})$$

2. Der Fall $K_\phi < 1/a$:
   Betrachten wir hier den "typischen" Fall, daß $f(n) \in \Theta(n^k)$ ist mit $a < b^k$. Aus einer Abschätzung $f(n) \leq c \cdot n^k$, d.h. $f_m \leq c \cdot b^{mk}$, wird mit Hilfe von (3):
   $$t_m \leq c \cdot b^{mk} \sum_{i=0}^{m} \left(\frac{a}{b^k}\right)^i$$
   und die rechts stehende geometrische Reihe konvergiert für $m \to \infty$. Daher also
   $$t_m \in O((b^m)^k) \ \text{ , also } \ t(n) \in O(n^k)$$
   Andererseits ist $t_m \geq f_m$, also insgesamt, wenn man es durch $n = b^m$ ausdrückt:
   $$t(n) \in \Theta(n^k)$$
   Ist f nicht von der angegebenen Form, so muss man eventuell genauer untersuchen, so wie das bei dem nächsten Fall geschieht:

3. Der Fall $K_\phi = 1/a$:
   Dies ist der "kritische" Fall. Schreiben wir hier zunächst einmal

$$f_m = a^m \cdot \tilde{f}_m \quad , \quad \text{wobei} \quad \overline{\lim_{m\to\infty}} \tilde{f}_m^{1/m} = 1$$

   d.h. $\tilde{f}_m$ beschreibt den subexponentiellen Anteil des Verhaltens von f_m. Dann ist also

$$t_m = \sum_{i=0}^{m} f_i \, a^{m-i} = a^m \sum_{i=0}^{m} \tilde{f}_i$$

   und der subexponentielle Anteil des Verhaltens von t_m wird durch die rechts stehende Summe beschrieben. Betrachten wir den Fall, wo $b^k = a$ ist und

$$f(n) = c \cdot n^k \cdot (\log_b n)^q \quad \text{für } n > 1, q \text{ reell}$$

   Dann ist also

$$f_m = c \cdot b^{mk} \cdot m^q \quad \text{d.h.} \quad \tilde{f}_m = c \cdot m^q \quad (m > 0)$$

   Hier sind wiederum drei Fälle zu unterscheiden:

   (a) Fall $q < -1$:
   In diesem Fall konvergiert $\sum_{i=0}^{m} \tilde{f}_i = c\,(f(1) + \sum_{i=1}^{m} i^q)$ für $m \to \infty$ und daher

$$t(n) = t(b^m) \in \Theta(a^m) = \Theta(n^{\log_b a})$$

   (b) Fall $q = -1$:
   In diesem Fall hat man es mit $f(1) + \sum_{i=1}^{m} \tilde{f}_i = f(1) + c\,H_m \approx c' \log m$ zu tun, und deshalb:

$$t(n) \in \Theta(a^m \log m) = \Theta\left(n^{\log_b a} \log\log_b n\right)$$

   (c) Fall $q > -1$:
   In diesem Fall verhält sich $\sum_{i=0}^{m} \tilde{f}_m = f(1) + c \sum_{i=1}^{m} i^q$ wie ein $\Theta(m^{q+1})$ und daher

$$t(n) \in \Theta(a^m\, m^{q+1}) = \Theta\left(n^{\log_b a}\, (\log_b n)^{1+q}\right)$$

1.5 Beispiele

Hier nun einige Beispiele:

$$\begin{aligned}
t(n) = t(n/2) + c &\Rightarrow t(n) \in \Theta(\log n)\\
t(n) = 2\,t(n/2) + c\,n &\Rightarrow t(n) \in \Theta(n \log n)\\
t(n) = 2\,t(n/2) + c\,n^2 &\Rightarrow t(n) \in \Theta(n^2)\\
t(n) = 4\,t(n/2) + c\,n^2 &\Rightarrow t(n) \in \Theta(n^2 \log n)\\
t(n) = 7\,t(n/2) + c\,n^2 &\Rightarrow t(n) \in \Theta(n^{\log_2 7})\\
t(n) = 2\,t(n/2) + \log n &\Rightarrow t(n) \in \Theta(n)\\
t(n) = 3\,t(n/2) + n \log n &\Rightarrow t(n) \in \Theta(n^{\log_2 3})\\
t(n) = 2\,t(n/2) + n \log n &\Rightarrow t(n) \in \Theta(n \log^2 n)\\
t(n) = 5\,t(n/2) + (n \log n)^2 &\Rightarrow t(n) \in \Theta(n^{\log_2 5})
\end{aligned}$$

2 Einige weitere Beispiele und Bemerkungen zu Potenzreihen und dem Wachstumsverhalten der Koeffizienten

2.1 Beispiele

1. Geometrische Reihe:

$$g(z) = 1/(1 - a\,z) = \sum_{n\geq 0} (a\,z)^n \text{ mit } a > 0$$

   Hier ist $K_g = 1/a$ und der subexponentielle Anteil $\tilde{g}_n$ an der Koeffizienten $g_n = a^n$ ist konstant $= 1$.

   Betrachtet man beispielweise das Quadrat von $g(z)$:

$$g(z)^2 = \sum_{n\geq 0} g_n^{(2)}\, z^n = \sum_{n\geq 0} (n+1)\, a^n z^n$$

   so sieht man, daß auch $g(z)^2$ den Konvergenzradius $K_{g^2} = 1/a$ hat, denn $g_n^{(2)} = (n+1)\, a^n$ wächst exponentiell wie a^n. Der subexponentielle Anteil ist hier $\tilde{g}_n^{(2)} = 1 + n$.

2. Logarithmische Reihe:

$$h(z) = -\log(1 - a\,z) = \sum_{n\geq 1} (a\,z)^n/n \text{ mit } a > 0.$$

Hier sind die Reihenkoeffizienten $h_n = a^n/n$, der Konvergenzradius ist, wie bekannt, $K_h = 1/a$, und der subexponentielle Anteil beträgt $\tilde{h}_n = 1/n$.

3. Catalan-Zahlen (sehr wichtiges Beispiel):

$$c(z) = \sum_{n\geq 0} c_n\, z^n = \left(1 - \sqrt{1 - 4\,z}\right)/2z$$

Die Koeffizienten von $c(z)$, die sogenannten Catalan-Zahlen $c_n, n \geq 0$, verdanken ihre Bedeutung für die Informatik der Tatsache, daß sie die Anzahl der vollständigen binären Bäume mit n inneren Knoten (also $n+1$ äußeren Knoten) angeben. Eine intime Kenntnis von Eigenschaften dieser Zahlen c_n ist für Komplexitätsanalysen von Algorithmen, die sich mit Bäumen befassen, unerlässlich. Unter den vielen anderen Bedeutungen dieser Zahlen, die deren Wichtigkeit unterstreichen, findet man die Anzahl der korrekten Klammerausdrücke über einem Alphabet, bestehend aus einem Klammerpaar. Anders ausgedrückt: c_n ist die Anzahl der Wörter der Länge $2n$ der sogenannten Dyck-Sprache, also der wichtigsten kontextfreien Sprache überhaupt.

Es ist nicht schwer, aus einer dieser "kombinatorischen" Bedeutungen von c_n eine rekursive Beziehung herzuleiten:

$$c_n = \sum_{i=0}^{n-1} c_i \cdot c_{n-i-1} \quad (n > 0), \quad c_0 = 1$$

die äquivalent ist zu der Funktionalgleichung

$$c(z) = 1 + z \cdot c(z)^2$$

Da dies eine harmlose quadratische Gleichung ist, kann man sie explizit lösen:

$$c(z) = \frac{1 - \sqrt{1 - 4\,z}}{2\,z}$$

und die Reihentwicklung der Wurzel liefert eine explizite Formel

$$c_n = \frac{1}{n+1}\binom{2\,n}{n}$$

Daraus (wie auch aus der obigen Rekursion) kann man die ersten Werte schnell berechnen:

$$(c_n)_{n\geq 0} = (1, 1, 2, 5, 14, 42, 132, 429, \ldots)$$

Die Frage, wie schnell diese Folge wächst, lässt sich dank der expliziten Formel und der Stirling-Formel für $n!$

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

leicht beantworten:

$$c_n \approx \frac{1}{n+1}\frac{4^n}{\sqrt{\pi n}}$$

und dies zeigt, daß das exponentielle Wachstum durch 4^n gegeben ist — es ist also $K_c = 1/4$ — und daß der subexponentielle Anteil am Wachstum

$$\tilde{c}_n = \frac{1}{(n+1)\sqrt{\pi n}} \in \Theta(n^{-3/2})$$

beträgt.

2.2 Zur Bedeutung des Konvergenzradius

Ist $\alpha(z) = \sum_{n\geq 0} a_n\, z^n$ eine Potenzreihe und K_α ihr Konvergenzradius, so bedeutet das, daß auf dem Kreisrand

$$\{\ z \in \mathbf{C}\ ;\ |z| = K_\alpha\ \}$$

in der komplexen Ebene etwas vorkommt, was den Konvergenzradius der Reihe daran hindert, größer zu sein: (mindestens) eine *Singularität* z_0 der Funktion $\alpha(z)$. Anders formuliert: der Konvergenzradius K_α ist der Radius der größten Kreises um den Nullpunkt, der in seinem Innern keine Singularität der Funktion enthält. Der Begriff *Singularität* kann hier nicht eingehender erörtert werden. Dies präzise zu behandeln ist eine Aufgabe der komplexen Analysis, der *Funktionentheorie*. Hier soll es genügen, dies anhand der vorigen Beispiele zu illustrieren:

1. Geometrische Reihe:

   An der Stelle $z_0 = 1/a$ ist der Ausdruck $1/(1 - a\,z)$ nicht definiert; es handelt sich um eine sogenannte *Polstelle* der Funktion. Polstellen sind Singularitäten eines einfachen Typs.

   Allgemein gilt folgende Aussage: ist $f(z) = p(z)/q(z)$ eine *rationale Funktion*, d.h. sind $p(z), q(z)$ Polynome mit (komplexen) Koeffizienten, so sind die Singularitäten von $f(z)$ genau die Nullstellen des Nennerpolynoms $q(z)$ (vorausgesetzt, daß die beiden Polynome $p(z)$ und $q(z)$ keinen gemeinsamen Teiler (= keine gemeinsame Nullstelle) haben. Der Konvergenzradius von $f(z)$ ist der kleinste Absolutbetrag einer Nullstelle von $q(z)$. Ist also

   $$f(z) = \sum_{n\geq 0} f_n\, z^n = \frac{p(z)}{q(z)}$$

   eine rationale Funktionen, so kennt man das exponentielle Wachstumsverhalten der Folge $(f_n)_{n\geq 0}$, wenn man die betragsmäßig kleinste Nullstelle von $q(z)$ kennt.

2. Logarithmische Reihe:

   An der Stelle $z_0 = 1/a$ ist auch $-\log(1-a\,z)$ nicht definiert. Es handelt sich hier um eine sog. *logarithmische Singularität*.

3. Catalan-Zahlen:

   Hier ist die Situation etwas subtiler: an der Stelle $z_0 = 1/4$ wäre der funktionale Ausdruck $c(z) = (1 - \sqrt{1-4z})/(2z)$ durchaus sinnvoll zu definieren. Um zu sehen, daß hier etwas "passiert", muss man sich vergegenwärtigen, daß $\sqrt{\ }$ eine *mehrdeutige* Funktion ist: $\sqrt{y}$ ist ja definiert als die Lösung(en) der Gleichung $z^2 = y$, und davon gibt es zwei verschiedene, außer wenn $y = 0$ ist. Eine algebraische Gleichung (= Polynomgleichung) n-ten Grades (in z) $p(z, y) = 0$ hat (für festes y) in der Regel n *verschiedene* Lösungen. Werte y, wo dies nicht der Fall ist, nennt man *algebraische Singularitäten*. Auch diese kommen als "Verursacher" eines Konvergenzradius in Frage, so wie man das bei $c(z)$ sieht, wo $y = 1/4$ algebraische Singularität ist.

Bei all diesen Beispielen sieht man, daß sich die "Singularität kleinsten Absolutbetrages", die also den Konvergenzradius bestimmt, auf der positiven

reellen Achse befindet. Das ist kein Zufall! Es gilt allgemein der Satz: hat eine Potenzreihe nichtnegative reelle Koeffizienten, so befindet sich eine Singularität kleinsten Absolutbetrages auf der positiven reellen Achse (falls nicht der Konvergenzradius unendlich ist, d.h. überhaupt keine Singularitäten im endlichen Bereich vorhanden sind). Das ist für die Komplexitätsanalyse eine gute Nachricht, denn da sind die Koeffizienten von Natur aus nichtnegativ! Um das exponentielle Wachstumsverhalten einer Folge $(a_n)_{n\geq 0}$ von "Kosten" zu kennen, muß man also nur $\alpha(z) = \sum_{n\geq 0} a_n\, z^n$ auf der positiven reellen Achse untersuchen und dort die kleinste "verdächtige" Stelle ausfindig machen.

Ein instruktives Beispiel zu diese Aussage: Bezeichne m_n die Anzahl der unär-binären Bäume mit genau $n + 1$ Knoten. Das sind also Bäume, deren Verzweigungsgrad bei den inneren Knoten entweder 1 oder 2 beträgt. Die treten natürlicherweise da auf, wo man es mit arithmetische Ausdrücken zu tun hat, bei denen sowohl einstellige wie zweistellige Operationssymbole auftreten. Die ersten Werte sind

$$(m_n)_{n\geq 0} = (1, 1, 2, 4, 9, 21, 51, \ldots)$$

Im Gegensatz zu der Situation bei den Catalan-Zahlen gibt es hier keine einfache Formel für die m_n. Das einfachste, was man sagen kann ist

$$m_n = \sum_{k=0}^{\lfloor n/2\rfloor} c_k \binom{n}{2k}$$

und wie man daraus etwas über das Verhalten für $n \to \infty$ erfahren soll, ist nicht so klar. Man kann aber (entweder mit Hilfe dieser Summenformel und dem, was man über die Catalan-Zahlen weiß, oder aber direkt aus dem Modell der unär-binären Bäume) eine Darstellung der zugehörigen Potenzreihe herleiten:

$$m(z) = \sum_{n\geq 0} m_n\, z^n = \frac{1 - z - \sqrt{1 - 2z - 3z^2}}{2z^2}$$

Die kritischen Punkt ist dort, wo der Term unter der Wurzel 0 wird. Nun gilt aber

$$1 - 2z - 3z^2 = (1 + z)(1 - 3z)$$

d.h. kritisch sind die Werte $z_0 = -1$ und $z_1 = 1/3$. Dabei ist $z_1 = 1/3$ der betragsmässig kleinste Wert, als ist der Konvergenzradius von $m(z)$ der Wert $K_m = 1/3$ und man weiß ohne weitere Rechnerei, daß das exponentielle Verhalten von $(m_n)_n \geq 0$ durch 3^n gegeben ist.

Durch stärkere Methoden, die im nächsten Abschnitt nur angedeutet werden, erhält man genauere Auskunft auch über den subexponentiellen Anteil:

$$m_n \approx 3^n \sqrt{\frac{3}{4\,\pi\,n^3}}$$

2.3 Stärkere Methoden

Die obigen Ausführungen sollen zeigen, daß man mit Hilfe des Konvergenzradius einer Potenzreihe oft mit recht elementaren Mitteln zu Aussagen über das Wachstumsverhalten einer Folge von reellen Zahlen kommen kann — zumindest was den exponentiellen Anteil am Wachstum betrifft. Diese erfreuliche Feststellung lässt aber gleichzeitig zwei (miteinander verwandte) Fragen unbeantwortet:

- Wie erfährt man etwas genaueres, quantitatives über das Verhalten des "subexponentiellen" Anteils?
- Was macht man eigentlich, wenn der Konvergenzradius unendlich ist — bei Funktionen also, deren Potenzreihenentwicklung in des gesamten komplexen Ebene konvergiert? (Solche Funktionen, zu denen so bekannte Beispiele wie Polynome, Exponentialfunktion, Sinus usw. gehören, nennt man auch *ganze* Funktionen).

Wiederum kann es hier nur um eine Andeutung gehen. Praktisch alle Methoden zur Beantwortung dieser beiden Fragen ("Sattelpunktmethode", "Methode von Heyman/Richman", "Transfermethode", ...) bedienen sich eines ganz zentralen Resultates der Funktionentheorie, der sog. *Integralformel von* Cauchy: Ist $f(z)$ eine "analytische" Funktion, d.h. hat $f(z)$ eine Potenzreihenentwicklung

$$f(z) = \sum_{n\geq 0} f_n\, z^n$$

die in einer Umgebung des Nullpunktes konvergiert, so gilt

$$f_n = \frac{1}{2\pi i} \oint \frac{f(z)}{z^{n+1}}\, dz \quad (n \geq 0)$$

wobei $\oint$ für ein komplexes Kurvenintegral steht, bei dem sich der Integrationsweg einmal im Gegenuhrzeigersinn um den Nullpunkt herumwindet, dabei

aber natürlich im Definitionsbereich von $f(z)$ verbleibt. Um den Zusammenhang mit vorher Gesehenem herzustellen, und um zu motivieren, warum es Sinn machen könnte, solche Dinge zu betrachten, hier eine ganz simple Anwendung:

Wählt man als Weg um den Nullpunkt einfach den Kreis mit Radius $r > 0$, also

$$\{ z \in \mathbf{C}\ ;\ |z| = r\ \} = \{ r \cdot e^{i\phi}\ ;\ 0 \leq \phi \leq 2\pi\ \}$$

wobei $r < K_f$ sein soll, so gilt

$$f_n = \frac{1}{2\pi i} \oint \frac{f(z)}{z^{n+1}}\, dz = \frac{1}{2\pi r^n} \int_0^{2\pi} \frac{f(r\cdot e^{i\phi})}{e^{in\phi}}\, d\phi \quad (n \geq 0)$$

Bezeichnet nun $M_r(f) := \max_{|z|=r} |f(z)|$ das Maximum von $f(z)$ entlang dieses Integrationswegs, so erhält man

$$|f_n| \leq \frac{1}{2\pi r^n} \int_0^{2\pi} \left| \frac{M_r(f)}{e^{in\phi}} \right| d\phi = \frac{M_r(f)}{r^n}$$

und das besagt ja gerade

$$|f_n| \in O(r^{-n})$$

für jedes r mit $0 < r < K_f$. Dies ist also nichts anderes als die eine Seite der Abschätzung aus dem Wachstumskriterium mittels Konvergenzradius.

Speziell der Fall $f(z) = e^z$ liefert wegen $f_n = 1/n!$

$$\frac{1}{n!} \leq \frac{e^r}{r^n}$$

und dies gilt für alle $r > 0$, da ja der Konvergenzradius von e^z unendlich ist. Also

$$\frac{1}{n!} \leq \min_{r>0} \frac{e^r}{r^n} = \frac{e^n}{n^n}$$

und das ist ein Schritt in Richtung auf die Formel von Stirling.

Um diesem ersten Schritt noch einen zweiten folgen zu lassen: hier ist ein "Rezept", wie man in vielen Fällen bei Reihen $f(z) = \sum_{n\geq 0} f_n\, z^n$ mit unendlichem Konvergenzradius ($K_f = \infty$) etwas über das Wachstum der Koeffizientenfolge $(f_n)_{n\geq 0}$ erfahren kann:

- sei $a(z) := z \cdot f'(z)/f(z)$ und $b(z) := z \cdot a'(z)$

- berechne $r_n > 0$ mit $a(r_n) = n$ (unter vernünftigen Bedingungen ist r_n eindeutig bestimmt)
- dann gilt

$$f_n \approx \frac{f(r_n)}{r_n^n \sqrt{2\pi b(r_n)}}$$

Also kleine Anwendung und Illustration: mit $f(z) = e^z$ ist $a(z) = z$ und $b(z) = z$, $r_n = n$ $(n \geq 0)$ und somit

$$\frac{1}{n!} \approx \frac{e^n}{n^n \sqrt{2\pi n}}$$

und das kommt der Wahrheit der STIRLING-Formel schon ein gutes Stück näher.

2.4 Literaturhinweise

Wer einmal sehen will, wie die hier nur angedeuteten Methoden funktionieren, wofür man sie einsetzt, bis hin zu Systemen der automatischen Komplexitätsanalyse von Programmen, kann sich in dem folgenden Artikel informieren, der seinerseits reichhaltige Information über grundlegende mathematische und weiterführende Literatur enthält:

> P. Flajolet, B. Salvy, P. Zimmermann : "Automatic average case analysis of algorithms", *Theoretical Computer Science*, Band 79 (1991), 37-109.

Ein Tutorial zu diesem Thema (ohne Berücksichtigung der neuesten Methoden der automatischen Analyse) enthält der Beitrag

> P. Flajolet : "Mathematical methods in the analysis of algorithms and data structures", in: E. Börger (Hrsg.), *Trends in Theoretical Computer Science*, Computer Science Press, 1988, 225-304.
> Gruppenbibliothek Informatik: 14GI/mat 7.2-520.

Schließlich ein enzyklopädischer Handbuchartikel zum Thema:

> J.S. Vitter und P. Flajolet : "Average case analysis of algorithms and data structures", Kapitel 9 im *Handbook of Theoretical Computer Science*, herausgegeben von J.v. Leeuwen, Elsevier, 1990, Band A, 432-524.
> Gruppenbibliothek Informatik: 14GI/mat 7-31.

Das Lehrbuch zum Thema:

> R. SEDGEWICK, P. FLAJOLET, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.
> Gruppenbibliothek Informatik: 14GI/mat 12.2.3–1004

Binäre Bäume

$\mathcal{B}$: Menge der binären Bäume, rekursiv definiert durch die Regeln:

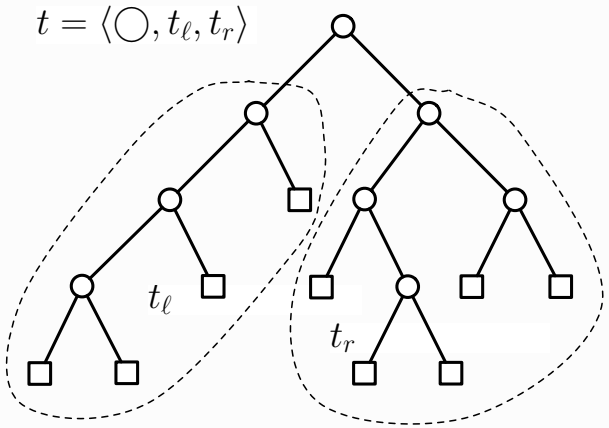
- $\square$ ist ein binärer Baum
- sind t_ℓ, t_r binäre Bäume, so ist auch $t = \langle \bigcirc, t_\ell, t_r \rangle$ ein binärer Baum
- nur das, was durch die beiden vorigen Regeln erzeugt werden kann, ist ein binärer Baum

Übliche Darstellung von Binärbäumen:
Bäume mit Wurzel, wobei jeder Knoten zwei oder keinen Knoten als (geordnete) Nachfolger hat.

- zwei Nachfolger: mit $\bigcirc$ bezeichnete inneren Knoten,
- kein Nachfolger: mit $\square$ bezeichneten äusseren Knoten (Blätter).

- $\square$ ist ein binärer Baum, der nur aus einem einzigen (äusseren) Knoten besteht;
- $t = \langle \bigcirc, t_\ell, t_r \rangle$ ist ein Baum mit Wurzel $\bigcirc$, an die zwei binäre Bäume t_ℓ und t_r als linker bzw. rechter Teilbaum angehängt sind.

Beispiel



Lineare Codierung von Binärbbäumen
(Wörter über dem Alphabet $\{\bigcirc, \square\}$)

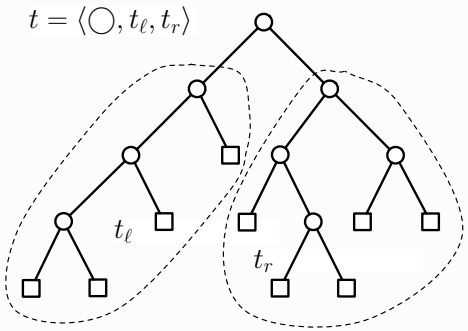
$$\text{code}(\square) = \square$$

$$\text{code}(\langle \bigcirc, t_\ell, t_r \rangle) = \bigcirc \, \text{code}(t_\ell) \, \text{code}(t_r)$$

Entsprechend:
Generierung mittels einer kontextfreien Grammatik in BNF $G_{\mathcal{B}}$ mit:

- Variablensymbol B (auch Startsymbol)
- Terminalsymbolen $\square$ und $\bigcirc$
- Produktionen

$$B \to \square \mid \bigcirc BB$$



$\text{code}(t_\ell) = \bigcirc\bigcirc\bigcirc\square\square\square\square,$

$\text{code}(t_r) = \bigcirc\bigcirc\square\bigcirc\square\square\bigcirc\square\square$

$\text{code}(t) = \bigcirc\,\text{code}(t_\ell)\,\text{code}(t_r)$

$= \bigcirc\bigcirc\bigcirc\bigcirc\square\square\square\square\bigcirc\bigcirc\square\bigcirc\square\square\bigcirc\square\square$

Bemerkungen:

- streicht man in jedem von $G_{\mathcal{B}}$ erzeugten Wort das letzte Symbol und identifiziert man

  $\bigcirc \sim$ "linke Klammer" $\quad \square \sim$ "rechte Klammer",

  so entsprechen diese Wörter genau den wohlgeformten Klammerausdrücken entsprechender Länge.
- die von $G_{\mathcal{B}}$ erzeugte Sprache ist kontextfrei, aber nicht regulär

Notation

- $I(t)$: innere Knoten von $t \in \mathcal{B}$ (internal)
  - $\square$ hat keine inneren Knoten
  - $\langle \bigcirc, t_\ell, t_r \rangle$ hat als innere Knoten die Wurzel $\bigcirc$, die inneren Knoten von t_ℓ und die inneren Knoten von t_r
- $E(t)$: äussere Knoten (Blätter) von $t \in \mathcal{B}$ (external)
  - $\square$ hat $\square$ als äusseren Knoten
  - $\langle \bigcirc, t_\ell, t_r \rangle$ hat als äussere Knoten die äusseren Knoten von t_ℓ und die äusseren Knoten von t_r
- $\mathcal{B}_n$: Menge der Binärbäume mit n inneren und $n+1$ äusseren Knoten

Struktur und Anzahl

- Strukturelle Aussage

$$\mathcal{B}_0 = \{\square\}$$

$$\mathcal{B}_{n+1} \leftrightarrow (\mathcal{B}_0 \times \mathcal{B}_n) \cup (\mathcal{B}_1 \times \mathcal{B}_{n-1}) \cup (\mathcal{B}_2 \times \mathcal{B}_{n-2}) \cup \cdots \cup (\mathcal{B}_n \times \mathcal{B}_0)$$

$$= \biguplus_{0 \le k \le n} \mathcal{B}_k \times \mathcal{B}_{n-k}$$

- Segners Formel

$$c_0 = 1$$
$$c_{n+1} = c_0 \cdot c_n + c_1 \cdot c_{n-1} + c_2 \cdot c_{n-2} + \cdots + c_n \cdot c_0$$

- Catalans Formel

$$c_n = \frac{1}{n+1}\binom{2n}{n} \in \Theta\left(\frac{4^n}{n^{3/2}}\right)$$

Parameter für Binärbäume

- $i(t)$ = Anzahl der inneren ("internal") Knoten von t:

$$i(t) = \begin{cases} 0 & \text{falls } t = \square \\ 1 + i(t_\ell) + i(t_r) & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$

- $e(t)$ = Anzahl der äusseren ("external") Knoten von t:

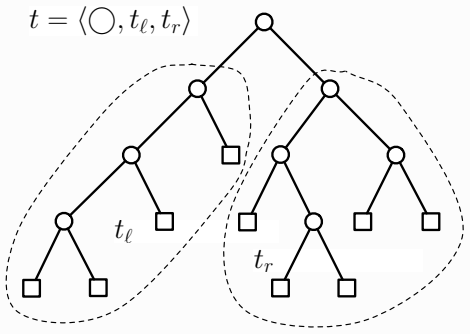
$$e(t) = \begin{cases} 1 & \text{falls } t = \square \\ e(t_\ell) + e(t_r) & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$

- $s(t) = i(t) + e(t)$ = Grösse ("size") von t:

$$s(t) = \begin{cases} 1 & \text{falls } t = \square \\ 1 + s(t_\ell) + s(t_r) & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$

- $h(t)$ = Höhe ("height") von t:

$$h(t) = \begin{cases} 0 & \text{falls } t = \square \\ 1 + \max\{h(t_\ell), h(t_r)\} & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$



$i(t) = 8, e(t) = 9, s(t) = 17, h(t) = 4$

Beziehungen zwischen den Parametern

Für alle binären Bäume t gelten folgende Aussagen:

- $e(t) = i(t) + 1$, und damit $s(t) = 2 \cdot i(t) + 1 = 2 \cdot e(t) - 1$

Beweis:

$$e(\square) - i(\square) = 1 - 0 = 1$$
$$\begin{aligned} e(\langle \bigcirc, t_\ell, t_r \rangle) - i(\langle \bigcirc, t_\ell, t_r \rangle) &= e(t_\ell) + e(t_r) - (i(t_\ell) + i(t_r) + 1) \\ &= e(t_\ell) - i(t_\ell) + e(t_r) - i(t_r) - 1 \\ &= 1 + 1 - 1 = 1 \end{aligned}$$

- $h(t) \leq i(t)$

  Beweis:

$$
\begin{aligned}
h(\square) &= 0 = i(\square) \\
h(\langle \bigcirc, t_\ell, t_r \rangle) &= 1 + \max\{h(t_\ell), h(t_r)\} \\
&\leq 1 + \max\{i(t_\ell), i(t_r)\} \\
&\leq 1 + i(t_\ell) + i(t_r) \\
&= i(\langle \bigcirc, t_\ell, t_r \rangle)
\end{aligned}
$$

- $e(t) \leq 2^{h(t)}$, also $\log e(t) \leq h(t)$

  Beweis:

$$
\begin{aligned}
e(\square) &= 1 = 2^0 = 2^{h(\square)} \\
e(\langle \bigcirc, t_\ell, t_r \rangle) &= e(t_\ell) + e(t_r) \\
&\leq 2^{h(t_\ell)} + 2^{h(t_r)} \\
&\leq 2 \cdot 2^{\max\{h(t_\ell), h(t_r)\}} \\
&= 2^{1+\max\{h(t_\ell), h(t_r)\}} \\
&= 2^{h(\langle \bigcirc, t_\ell, t_r \rangle)}
\end{aligned}
$$

Höhe von Knoten

- Für $a \in I(t) \cup E(t)$:

  $h(a, t) =$ Höhe von a in $t =$ Abstand von a zur Wurzel von t

- rekursive Beschreibung:

$$
\begin{aligned}
h(\square, \square) &= 0 \\
h(a, \langle \bigcirc, t_\ell, t_r \rangle) &= \begin{cases} 0 & a = \bigcirc \\ h(a, t_\ell) + 1 & a \in E(t_\ell) \cup I(t_\ell) \\ h(a, t_r) + 1 & a \in E(t_r) \cup I(t_r) \end{cases}
\end{aligned}
$$

Innere und äussere Weglänge

- innere Weglänge von t :

$$w_i(t) = \sum_{a \in I(t)} h(a, t)$$

- äussere Weglänge von t :

$$w_e(t) = \sum_{a \in E(t)} h(a, t)$$

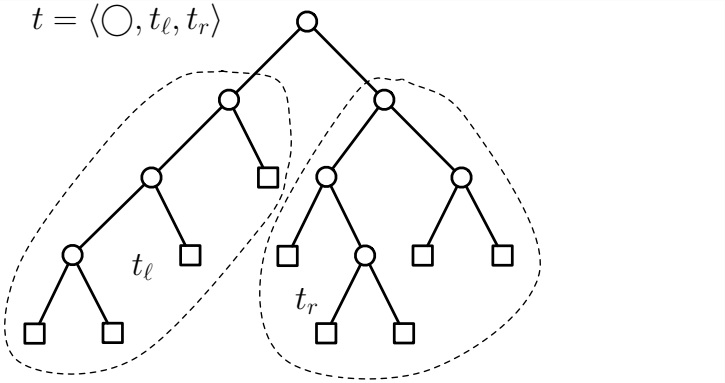
- mittlere innere Weglänge von t :

$$\overline{w}(t) = \frac{w_i(t)}{i(t)}$$

- mittlere äussere Weglänge = mittlere Höhe :

$$\overline{h}(t) = \frac{w_e(t)}{e(t)}$$

Beispiel



$$w_i(t) = 14, w_e(t) = 30, i(t) = 8, \overline{w}(t) = \frac{14}{8} = \frac{7}{4}, \overline{h}(t) = \frac{30}{9} = \frac{10}{3}$$

- Allgemein gilt: $w_e(t) = w_i(t) + 2\,i(t)$

Binäre Bäume mit Bewertung

- A totalgeordnete Menge (z.B. $\mathbb{Z}, \mathbb{R}, \ldots$)
- Für $t = \langle \bigcirc, t_\ell, t_r \rangle \in \mathcal{B}$ und $v : I(t) \to A$ sei

$$v_\ell : I(t_\ell) \to A : a \mapsto v(a)$$
$$v_r : I(t_r) \to A : a \mapsto v(a)$$

die Restriktion von v auf den linken bzw. rechten Teilbaum

Binäre Suchbäume

- Ein Paar (t, v) mit $t \in \mathcal{B}, v : I(t) \to A$ heisst *binärer Suchbaum* über A, wenn
  - $t = \square$ (und somit $v = \lambda =$ leere Funktion), oder
  - $t = \langle \bigcirc, t_\ell, t_r \rangle$ mit $v(\bigcirc) = a$ und
  1. (t_ℓ, v_ℓ) ist ein binärer Suchbaum über $A_{<a} = \{b \in A;\ b < a\}$
  2. (t_r, v_r) ist ein binärer Suchbaum über $A_{>a} = \{b \in A;\ b > a\}$

  Insbesondere gilt also
  $\max_{i \in I(t_\ell)} v(i) < v(\bigcirc) = a < \min_{i \in I(t_r)} v(i)$
- $\mathcal{BS}(A)$: Menge der binären Suchbäume über A
- $\mathcal{BS}_n(A)$: Menge der $(t, v) \in \mathcal{BS}(A)$ mit $t \in \mathcal{B}_n$

- Struktur:

$$\mathcal{BS}(A) = \biguplus_{a \in A} (\{a\} \times \mathcal{BS}(A_{<a}) \times \mathcal{BS}(A_{>a}))$$

- Anzahl:

$$\sharp \mathcal{BS}_n(A) = \binom{\sharp A}{n} \cdot c_n = \binom{\sharp A}{n} \cdot \frac{1}{n+1} \binom{2n}{n}$$

Suchen in binären Suchbäumen

Suche, ob ein gegebenes $a \in A$ in $(t, v) \in \mathcal{BS}(A)$ vorkommt:

- falls $t = \square$: FAIL
- falls $t = \langle \bigcirc, t_\ell, t_r \rangle$
  - falls $v(\bigcirc) = a$: FOUND
  - falls $v(\bigcirc) > a$: suche a in (t_ℓ, v_ℓ)
  - falls $v(\bigcirc) < a$: suche a in (t_r, v_r)

- Die (mittlere) innere Weglänge von t ist ein Maß für den (mittleren) Aufwand erfolgreicher Suche in t.
- Die (mittlere) äussere Weglänge von t ist ein Maß für den (mittleren) Aufwand erfolgloser Suche in t.

Rekursive Konstruktion von binären Suchbäumen

- Für eine Folge $\mathbf{a} = a_1 a_2 \dots a_n \in A^+$ sei

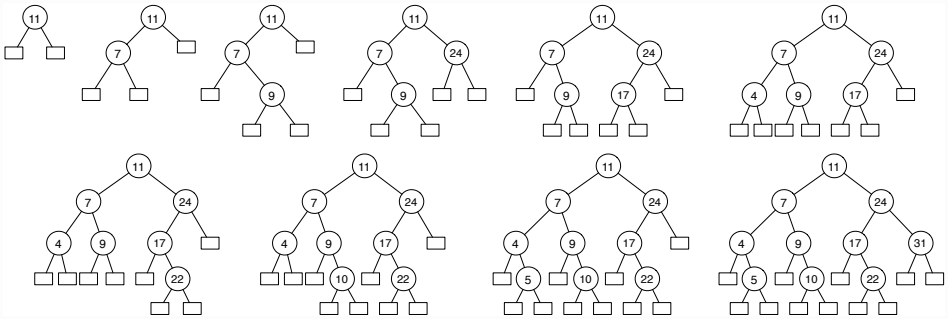
$$\mathbf{a}_< : \quad \text{die Teilfolge der } a_i < a_1,$$
$$\mathbf{a}_> : \quad \text{die Teilfolge der } a_i > a_1$$

- Für die leere Folge λ sei $bs(\lambda) = (\square, \lambda)$
- Für $a \in A^+$ wird $bs(\mathbf{a}) = (t, v)$ definiert durch

$$v(\bigcirc) = a_1, \ bs(\mathbf{a}_<) = (t_\ell, v_\ell), \ bs(\mathbf{a}_>) = (t_r, v_r)$$

- Ein sequentieller Algorithmus zur Konstruktion verwendet sukzessives Einfügen bei erfolgloser Suche

Konstruktion des binären Suchbaumes zu
$\mathbf{a} = (11, 7, 9, 24, 17, 4, 22, 10, 5, 31)$

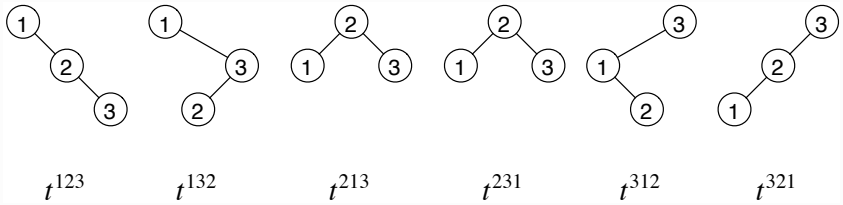


- Problem: wie gross ist der Aufwand für (erfolgreiche) Suche, wenn man aus Daten sequentiell einen binären Suchbaum konstruiert hat?
- Wird $a \in A$ in den A-Suchbaum (t, v) (erfolgreich) gesucht, so ist die Höhe $h(b, t)$ des Knotens $b \in I(t)$ mit $v(b) = a$ relevant.
- Damit ist klar: es kommt sowohl auf das gesuchte Datum an, als auch auf den Suchbaum (d.h. die Reihenfolge, in der die Daten zur Konstruktion des Suchbaumes verwendet wurden).

Permutationsmodell

- Für n (verschiedene) Daten $\in A$ betrachte alle $n!$ Permutationen als gleichwahrscheinliche inputs für die Konstruktion eines bs.
- Man kann annehmen, dass $A = \{1, 2, \dots, n\}$ und alle $\sigma \in \mathcal{S}_n$ inputs sind. Für $\sigma \in \mathcal{S}_n$ sei $bs(\sigma) = (t^\sigma, v^\sigma)$, also $t \in \mathcal{B}_n$.
  - Vorsicht: verschiedene $\sigma \in \mathcal{S}_n$ können denselben Suchbaum liefern!
  - v^σ ist durch t^σ eindeutig bestimmt, also redundant!
- Es wird mit gleicher Wahrscheinlichkeit nach jedem der Daten $k \in \{1, 2, \dots, n\}$ gesucht
- Für einen festen A-Suchbaum (t, v) ist die (mittlere) innere Weglänge $w_i(t)$ bzw. $\overline{w}(t) = w_i(t)/n$ das Maß für den Suchaufwand.

Beispiel: die Suchbäume zu den Permutationen aus $\mathcal{S}_3$



Es gilt

$$w_i(t^{123}) = w_i(t^{132}) = w_i(t^{312}) = w_i(t^{321}) = 3, w_i(t^{213}) = w_i(t^{231}) = 2$$

also

$$w_3 = \frac{1}{3!}\sum_{\sigma\in\mathcal{S}_3} w_i(\sigma) = \frac{3+3+2+2+3+3}{6} = \frac{8}{3}, \overline{w}_3 = \frac{1}{3}w_3 = \frac{8}{9}$$

- Gesucht ist

$$w_n = \frac{1}{n!}\sum_{\sigma\in\mathcal{S}_n} w_i(t^\sigma) \quad \text{bzw.} \quad \overline{w}_n = \frac{1}{n!}\sum_{\sigma\in\mathcal{S}_n} \overline{w}(t^\sigma) = \frac{1}{n}\,w_n$$

- Wir werden sehen: die w_n erfüllen die Quicksort-Rekursion!

$$w_n = \frac{2}{n}\sum_{k=1}^{n-1} w_k + n - 1, \quad w_0 = 0$$

- Folgerung: der mittlere Suchaufwand bei binären Suchbäumen im Permutationsmodell ist

$$\overline{w}_n = \frac{1}{n}w_n = (2 + \frac{2}{n})H_n - 4 \in \Theta(\log n)$$

- Wie oft tritt jeder binäre Baum $t \in \mathcal{B}_n$ bei der Konstruktion von binären Suchbäumen auf?
- $s_t = \sharp\{\sigma \in \mathcal{S}_n\,;\, t^\sigma = t\}$
- Behauptung:

$$s_t = \frac{n!}{\prod_{a\in I(t)} i(t_a)}$$

  Dabei bezeichnet t_a den Teilbaum von t mit Wurzel a
- Beispiel: $\sigma = [5,1,3,2,7,6,4] \in \mathcal{S}_7$ erzeugt $t^\sigma \in \mathcal{B}_7$ mit $\text{code}(t^\sigma) =$ ○○□○○□□○□□○○□□□
  Identifiziert man die Zahlen $a \in \{1,\ldots,7\}$ mit den Knoten, an denen sie in t^σ stehen, so gilt $i(t_1) = 4, i(t_2) = 1, i(t_3) = 3, i(t_4) = 1, i(t_5) = 7, i(t_6) = 1$ und $i(t_7) = 2$ und es gibt $s_{t^\sigma} = \frac{7!}{4\cdot1\cdot3\cdot1\cdot7\cdot1\cdot2} = 30$ Permutationen aus $\mathcal{S}_7$, die den gleichen binären Baum erzeugen.

Zum Beweis:

- Betrachte die Abbildung

$$\mathcal{S}_n \ni \sigma \mapsto (P_\sigma, \sigma_<, \sigma_>^{\ominus\sigma_1}) \in \binom{2..n}{\sigma_1-1} \times \mathcal{S}_{\sigma_1-1} \times \mathcal{S}_{n-\sigma_1}$$

  Dabei ist (mit $\sigma_1 = k$):
  - $P_\sigma = \{i_1, i_2, \ldots, i_{k-1}\} \subseteq \binom{2..n}{k-1}$ mit $2 \le i_1 < i_2 < \ldots < i_{k-1} \le n$ die Positionen $2 \le i \le n$ mit $\sigma_i < k$
  - $\sigma_< = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_{k-1}} \in \mathcal{S}_{k-1}$
  - $\sigma_>^{\ominus k} = (\sigma_{j_1} - k)(\sigma_{j_2} - k)\ldots(\sigma_{j_{n-k}} - k) \in \mathcal{S}_{n-k}$, wobei $j_1 < j_2 < \ldots < j_{n-k} \le n$ die Positionen $2 \le j \ne n$ mit $\sigma_j > k$ sind

Beispiel

$$\mathcal{S}_n \ni \sigma \mapsto (P_\sigma, \sigma_<, \sigma_>^{\ominus\sigma_1}) \in \binom{2..n}{\sigma_1-1} \times \mathcal{S}_{\sigma_1-1} \times \mathcal{S}_{n-\sigma_1}$$

mit $n = 9$

- $\sigma = (6\,8\,1\,3\,7\,9\,5\,4\,2)$, also $k = \sigma_1 = 6$
- $P_\sigma = (3\,4\,7\,8\,9) \in \binom{2..9}{5}$
- $\sigma_< = (1\,3\,5\,4\,2) \in \mathcal{S}_5$
- $\sigma_>^{\ominus 6} = (2\,1\,3) \in \mathcal{S}_3$

- Diese Abbildung

$$\sigma \mapsto (P_\sigma, \sigma_<, \sigma_>^{\ominus\sigma_1}) : \mathcal{S}_n \to \biguplus_{1\le k\le n} \binom{2..n}{k-1} \times \mathcal{S}_{k-1} \times \mathcal{S}_{n-k}$$

ist eine Bijektion!

- Es gilt für alle $\sigma \in \mathcal{S}_n$ und mit $\sigma_\ell = \sigma_<$, $\sigma_r = \sigma_>^{\ominus\sigma_1}$

$$t^\sigma = \langle \bigcirc, t^{\sigma_\ell}, t^{\sigma_r} \rangle$$

- Damit ist per Induktion für $t \in \mathcal{B}_n$ mit $t_\ell \in \mathcal{B}_{k-1}$, $t_r \in \mathcal{B}_{n-k}$

$$\begin{aligned} s_t = \sharp\{\sigma \in \mathcal{S}_n\,;\, t^\sigma = t\} &= \binom{n-1}{k-1} \cdot s_{t_\ell} \cdot s_{t_r} \\ &= \frac{(n-1)!}{(k-1)!(n-k)!} \cdot \frac{(k-1)!}{\prod_{a\in I(t_\ell)} i((t_\ell)_a)} \cdot \frac{(n-k)!}{\prod_{b\in I(t_r)} i((t_r)_b)} \\ &= \frac{n!}{\prod_{a\in I(t)} i(t_a)} \end{aligned}$$

- Mit

$$\mathbf{p}_n : \mathcal{B}_n \mapsto [0,1] : t \mapsto \mathbf{p}_n(t) = \frac{\sharp\{\sigma \in \mathcal{S}_n\,;\, t^\sigma = t\}}{n!} = \frac{1}{\prod_{a\in I(t)} i(t_a)}$$

wird eine Wahrscheinlichkeitsverteilung $\mathbf{p}_n$ auf $\mathcal{B}_n$ definiert: sie gibt für jeden binären Baum $\in \mathcal{B}_n$ die Wahrscheinlichkeit an, mit welcher er bei der Konstruktion binärer Suchbäume unter dem Permutationsmodell auftritt.

- Es gilt für $t \in \mathcal{B}_n$ mit $t_\ell \in \mathcal{B}_{k-1}$, $t_r \in \mathcal{B}_{n-k}$

$$\mathbf{p}_n(t) = \frac{1}{n} \cdot \mathbf{p}_{k-1}(t_\ell) \cdot \mathbf{p}_{n-k}(t_r)$$

- Beweis der Quicksort-Rekursion für die mittlere Weglänge

$$\begin{aligned} w_n &= \frac{1}{n!} \sum_{\sigma\in\mathcal{S}_n} w_i(t^\sigma) = \sum_{t\in\mathcal{B}_n} \mathbf{p}_n(t) \cdot w_i(t) \\ &= \sum_{k=1}^{n} \sum_{\substack{t_\ell\in\mathcal{B}_{k-1} \\ t_r\in\mathcal{B}_{n-k}}} \frac{1}{n} \cdot \mathbf{p}_{k-1}(t_\ell) \cdot \mathbf{p}_{n-k}(t_r)\,(w_i(t_\ell) + w_i(t_r) + n) \\ &= \frac{1}{n} \sum_{k=1}^{n} (w_{k-1} + w_{n-k} + n - 1) = \frac{2}{n} \sum_{k=0}^{n-1} w_k + n - 1 \end{aligned}$$

Quicksort und Suchbäume

In dieser Notiz soll dargestellt werden, daß es einen tieferen Zusammenhang zwischen *quicksort* und dem bekannten Konzept der binären Suchbäume gibt. Das quantitative Verhalten von *quicksort* leitet sich demnach aus Eigenschaften der Weglänge von Such- bzw. Prioritätsbäumen ab. Dies gibt einen kleinen Eindruck von der in jüngster Zeit aufgekommenen Analysemethode der "randomisierten Suchbäume".

1 Suchbäume und Prioritätsbäume

- Ist $M \subset \mathbf{Z}$ eine (endliche) Menge, so ist die Menge $\mathcal{BST}_M$ der *binären Suchbäume* ("binary search trees") über M rekursiv definiert durch
  - $\mathcal{BST}_\emptyset := \{\emptyset\}$
  - falls $\sharp M \geq 1$: $\mathcal{BST}_M := \bigcup_{m \in M} (\mathcal{BST}_{M^{<m}} \times \{m\} \times \mathcal{BST}_{M^{>m}})$

  Dabei bezeichnet $\emptyset$ den "leeren Baum", und $M^{<m} := \{x \in M\,;\, x < m\}$, $M^{>m} := \{x \in M\,;\, x > m\}$.
  Eine binärer Suchbaum über M ist also ein mit den Elementen von M an den Knoten beschrifteter Baum, bei dem für jeden Knoten gilt: die Beschriftung dieses Knotens ist echt größer (kleiner) als alle Beschriftungen von Knoten des an ihm angehängten linken (rechten) Teilbaums.
  Ist $\sharp M = n$, so enhält $\mathcal{BST}_M$ genau $c_n = \frac{1}{n+1}\binom{2\,n}{n}$ Elemente (CATALAN-Zahlen!), denn jeder binäre Baum mit n Knoten lässt sich auf genau eine Weise mit den Elementen von M so beschriften, daß ein binärer Suchbaum entsteht.

- Ist $M \subset \mathbf{Z}$ eine (endliche) Menge, so ist die Menge $\mathcal{BPT}_M$ der *binären Prioritätsbäume* ("binary priority trees") über M rekursiv definiert durch
  - $\mathcal{BPT}_\emptyset := \{\emptyset\}$
  - falls $\sharp M \geq 1$ und $m = \min M$:
    $\mathcal{BPT}_M := \bigcup_{M' \subseteq M \setminus \{m\}} \left(\mathcal{BPT}_{M'} \times \{m\} \times \mathcal{BPT}_{M \setminus \{m\} \setminus M'}\right)$

  Ein binärer Prioritätsbaum über M ist also ein mit den Elementen von M an den Knoten beschrifteter Baum, bei dem für jeden Knoten gilt: die Beschriftung dieses Knotens ist echt kleiner als alle Beschriftungen von Knoten der an ihm angehängten Teilbäume.
  Ist $\sharp M = n$, so enhält $\mathcal{BPT}_M$ genau $n!$ Elemente — es ist einfach, eine Bijektion zwischen den Permutationen von M und den Elementen von $\mathcal{BPT}_M$ anzugeben.

- Ist $M \subset \mathbf{Z}$ eine endliche Menge und $\pi : M \to \mathbf{Z}$ eine injektive Abbildung ("Prioritätsfunktion"), so gibt es genau ein $t \in \mathcal{BST}_M$ derart, daß $\pi(t) \in \mathcal{BPT}_{\pi(M)}$. Hierbei bezeichnet $\pi(t)$ den mit den Elementen der Menge $\pi(M)$ beschrifteten Baum, der aus t dadurch hervorgeht, daß man an jedem Knoten die Beschriftung m durch ihre Priorität $\pi(m)$ ersetzt. Dieser eindeutig

bestimmte Baum $t \in \mathcal{BST}_M$ wird für spätere Verwendung mit $t_{M,\pi}$ bezeichnet. Sein Bild $\pi(t_{M,\pi}) \in \mathcal{BPT}_{\pi(M)}$ wird mit t_π abgekürzt.

- Ist t irgendein Baum (mit Wurzel), so bezeichnet man als *Weglänge* $w(t)$ von t die Summe aller Längen von Wegen, die von irgendeinem einem Knoten von t zur Wurzel führen, also die Summe aller Distanzen von Baumknoten zur Wurzel. Eine äquivalente Formulierung:

$$w(t) = \sharp\{\,(x,y)\,;\ x,y \text{ Knoten in } t \text{ mit } x \prec_t y\,\}$$

  wobei $\prec_t$ die (strikte) Vorgängerordnung in t bezeichnet.
  Für Binärbäume t gilt offensichtlich

$$w(t) = \begin{cases} 0 & \text{falls } t = \emptyset \\ w(t') + w(t'') + \sharp t' + \sharp t'' & \text{falls } t = (t', s, t'') \end{cases}$$

2 Mittlere Weglänge in Prioritätsbäumen

- Auf Grund ihrer rekursiven Definition lassen binäre Prioritätsbäume folgende Zerlegung zu:

  > ist $t \in \mathcal{BPT}_{\{1,\ldots,n+1\}}$, so entspricht diesem Baum ein-eindeutig ein Tripel (M', t_1, t_2) mit $M' \subseteq \{2,\ldots,n+1\}$, $t_1 \in \mathcal{BPT}_{\{1,\ldots,\sharp M'\}}$ und $t_2 \in \mathcal{BPT}_{\{1,\ldots,n-\sharp M'\}}$

  Dabei entspricht die Menge M' der Menge der Beschriftungen des linken Teilbaumes von t, entsprechend ist $\{2,\ldots,n+1\} \setminus M'$ die Menge der Beschriftungen des rechten Teilbaumes von t. Aus der Kenntnis von M' und t_1 (bzw. t_2) kann man den linken (bzw. rechten) Teilbaum von t durch entsprechende ordnungstreue Umbenennung rekonstruieren.

- Es geht nun um die *mittlere Weglänge* für binäre Prioritätsbäume mit n Knoten. Sei also

$$w_n := \sum_{t \in \mathcal{BPT}_{\{1,\ldots,n\}}} w(t)$$

  Aus der gerade beschriebenen Zerlegungseigenschaft und dem im vorigen Abschnitt erwähnten Verhalten der Weglängen beim Übergang zu den Teilbäumen folgt

$$w_{n+1} = \sum_{j=0}^{n} \binom{n}{j} \sum_{t_1,t_2} (w(t_1) + w(t_2) + \sharp t_1 + \sharp t_2)$$

  wobei die Summe $\sum_{t_1,t_2}$ über alle Paare $(t_1,t_2) \in \mathcal{BPT}_{\{1,\ldots,j\}} \times \mathcal{BPT}_{\{1,\ldots,n-j\}}$ läuft.
  Beachtet man nun $\sharp t_1 + \sharp t_2 = n$ und $bpt_k := \sharp\mathcal{BPT}_{\{1,\ldots,k\}} = k!$, so erhält man

$$\begin{aligned} w_{n+1} &= \sum_{j=0}^{n} \binom{n}{j} \left(w_j\, \sharp bpt_{n-j} + \sharp bpt_n\, w_{n-j} + n\, bpt_j\, bpt_{n-j}\right) \\ &= n! \sum_{j=0}^{n} \left(\frac{w_j}{j!} + \frac{w_{n-j}}{(n-j)!} + n\right) \end{aligned}$$

Für die mittleren Weglängen $\overline{w}_n := w_n/n!$ gilt also

$$\overline{w}_{n+1} = n + \frac{2}{n+1}\sum_{j=0}^{n}\overline{w}_j$$

und dies ist gerade die *quicksort*-Rekursion!

3 Quicksort und binäre Suchbäume

- Ist $M \subset \mathbf{Z}$ eine endliche Menge und L eine Umordnung (permutation) von M, so gehört zu jeder Exekution `quicksort(L)` ein binärer Suchbaum $\tau \in \mathcal{BST}_M$, der "splitter-Baum". Induktiv beschrieben: zu der leeren Liste gehört der leere Baum, zu einer Liste $L = [a]$ der Länge 1 gehört der Baum mit einem Knoten, der mit a beschriftet ist. Für eine Liste L der Länge ≥ 2 wird `split(L)` aufgerufen — dies liefert ein splitting $[L', s, L'']$, wobei das Element s der splitter ist. Der splitter-Baum dieser Exekution besteht nun aus der Wurzel s und den beiden Teilbäumen τ' und τ'', die zu den Exekutionen von `quicksort(L')` bzw `quicksort` L'' gehören. Wegen der randomisierten Arbeitsweise von `split` werden verschieden Aufrufe `quicksort(L)` i.a. verschiedene splitter-Bäume erzeugen.

- Die Anzahl der Vergleichsoperationen bei einem splitting $L \to [L', s, L'']$ ist

$$\sharp L - 1 = \sharp L' + \sharp L'' = \sharp\tau' + \sharp\tau''$$

Per Induktion erkennt man, daß die Anzahl der Vergleichsoperationen bei einem Aufruf `quicksort(L)` gerade die Weglänge des zugehörigen splitter-Baumes ist:

$$\sharp\{\,(x,y)\ ;\ x \prec_\tau y\,\} = w(\tau)$$

- Betrachten wir nun folgende Variante von *quicksort*, genannt *deterministisches quicksort mit Prioritätsfunktion*:
Wieder sei $M \subset \mathbf{Z}$ (endliche) Menge und L irgendeine Permutation von M. Ferner sei $\pi : M \to \mathbf{Z}$ eine Prioritätsfunktion.

  `dqs(L,`π`)` : der Ablauf ist wie bei `quicksort(L)`, aber mit dem Unterschied, daß bei jedem Aufruf `split(...)` das Element höchster Priorität (= kleinster Wert von π) der zu splittenden Teilliste zum splitter gemacht wird.

  Der zur Exekution von `dqs(L,`π`)` gehörende splitter-Baum ist $t_{M,\pi}$! (Siehe ersten Abschnitt für die Definition von $t_{M,\pi}$). Daher gilt:

  Die Anzahl der Vergleichsoperationen beim Aufruf `dqs(L,`π`)` ist gleich der Weglänge von $t_{M,\pi}$ (und das ist auch die Weglänge von τ_π) — diese Anzahl hängt also nur von der Prioritätsfunktion π ab, nicht von L.

- Betrachten wir schließlich folgende Version von *quicksort*, genannt *quicksort mit randomisierter Priorität*:
Wieder sei $M \subset \mathbf{Z}$ (endliche) Menge und L irgendeine Permutation von M.

  `rqs(L)` : falls $\sharp M = n$ ist, wird zunächst irgendeine Prioritätsfunktion $\pi : M \to \{1, ..., n\}$ gewählt, mit gleicher Wahrscheinlichkeit für alle $n!$ solchen Funktionen.. Dann wird `dqs(L,`π`)` exekutiert.

  Die Anzahl der Vergleichsoperationen bei Exekution von `rqs(L)` ist also gleich der Weglänge $w(t_\pi)$, und wenn π über alle Prioritätsfunktionen variiert, läuft t_π über $\mathcal{BPT}_{\{1,...,n\}}$. Daher gilt:

  Für jede (!) feste Permutation L von M ist die mittlere Anzahl von Vergleichsoperationen bei Aufruf von `rqs(L)` gleich der mittleren Weglänge $\mathbf{E}[w(t), t \in \mathcal{BPT}_{\{1,...,\sharp M\}}]$.
  Diese Aussage überträgt sich dann natürlich auch auf die Situation, wo über alle möglichen Permutationen L von M gemittelt wird.

- Abschließend ist zu bemerken, daß *randomized quicksort* `rqs` das bekannte *quicksort* simuliert, indem die Zufallsauswahl der splitter, die ja üblicherweise bei Aufruf von `split` vorgenommen wird, in die vorab bestimmte Prioritätsfunktion π verlegt wird. Gleichverteilung bei Auswahl von π überträgt sich auf die Gleichverteilung bei Auswahl der splitter. Deshalb gilt:

  Für jede (!) feste Permutation L von M ist die mittlere Anzahl von Vergleichsoperationen bei Aufruf von `quicksort(L)` gleich der mittleren Weglänge $\mathbf{E}[w(t), t \in \mathcal{BPT}_{\{1,...,\sharp M\}}]$
  Diese Aussage überträgt sich dann natürlich auch auf die Situation, wo über alle möglichen Permutationen L von M gemittelt wird.

Vergleich von Sortierkomplexitäten

$S(n) =$ minimale Anzahl von Vergleichen zum Sortieren von Listen der Länge n im worst case:

$$S(n) \geq \lceil \log n! \rceil$$

$M(n) =$ minimale Anzahl von Vergleichen zum Sortieren von Listen der Länge n im worst case mittels Mergesort

$$\begin{aligned} M(n) &= M(\lceil \frac{n}{2} \rceil) + M(\lfloor \frac{n}{2} \rfloor) + n - 1 \\ &= n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 \end{aligned}$$

$B(n) =$ minimale Anzahl von Vergleichen zum Sortieren von Listen der Länge n im worst case mittels Insertionsort (mit binärem Einfügen)

$$\begin{aligned} B(n) &= \sum_{k=2}^{n} \lceil \log k \rceil \\ &= n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 \end{aligned}$$

$F(n) =$ minimale Anzahl von Vergleichen zum Sortieren von Listen der Länge n im worst case mittels merge insertion (Ford-Johnson Algorithmus, 1959)

$$\begin{aligned} F(n) &= \sum_{k=2}^{n} \lceil \log \frac{3}{4} k \rceil \\ &= n\lceil \log \frac{3}{4} n \rceil - \lfloor 2^{\lfloor \log 6n \rfloor}/3 \rfloor + \frac{1}{2} \lfloor \log 6n \rfloor \end{aligned}$$

n	1	2	3	4	5	6	7	8	9	10	11	12	13
$M(n) = B(n)$	0	1	3	5	8	11	14	17	21	25	29	33	37
$F(n)$	0	1	3	5	7	10	13	16	19	22	26	30	34
$S(n)$	0	1	3	5	7	10	13	16	19	22	26	30^*	34^{**}
$\lceil \log n! \rceil$	0	1	3	5	7	10	13	16	19	22	26	29	33

$*$: M. Wells, 1965; $**$: M. Peczarski, 2002

Vorbemerkungen über binäre Bäume

$\mathcal{B}$: Menge der binären Bäume, rekursiv definiert durch die Regeln:

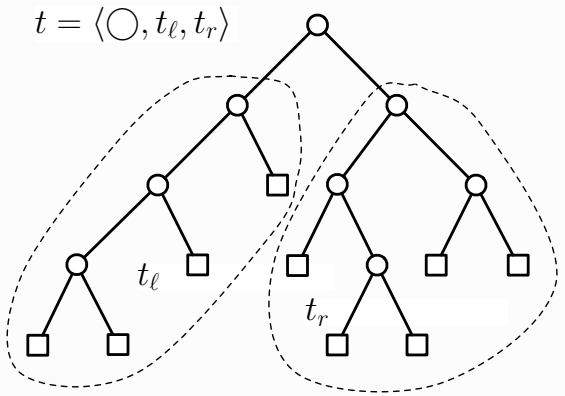
- $\square$ ist ein binärer Baum
- sind t_ℓ, t_r binäre Bäume, so ist auch $t = \langle \bigcirc, t_\ell, t_r \rangle$ ein binärer Baum
- nur das, was durch die beiden vorigen Regeln erzeugt werden kann, ist ein binärer Baum

Übliche Darstellung:

Bäume mit Wurzel, wobei jeder Knoten zwei oder keinen Knoten als (geordnete) Nachfolger hat.

- zwei Nachfolger: mit $\bigcirc$ bezeichnete inneren Knoten,
- kein Nachfolger: mit $\square$ bezeichneten äusseren Knoten (Blätter).

- $\square$ ist ein binärer Baum, der nur aus einem einzigen (äusseren) Knoten besteht;
- $t = \langle \bigcirc, t_\ell, t_r \rangle$ ist ein Baum mit Wurzel $\bigcirc$, an die zwei binäre Bäume t_ℓ und t_r als linker bzw. rechter Teilbaum angehängt sind.



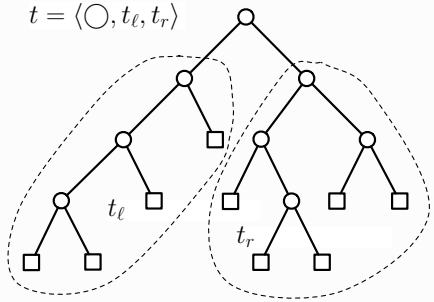
Lineare Codierung von Binärbbäumen (Wörter über dem Alphabet $\{\bigcirc, \square\}$)

$$\text{code}(\square) = \square \quad \text{code}(\langle \bigcirc, t_\ell, t_r \rangle) = \bigcirc\, \text{code}(t_\ell)\, \text{code}(t_r)$$

Entsprechend: Generierung mittels einer kontextfreien Grammatik in BNF $G_{\mathcal{B}}$

- Variablensymbol B (auch Startsymbol)
- Terminalsymbolen $\square$ und $\bigcirc$
- Produktionen

$$B \to \square \mid \bigcirc BB$$



$\text{code}(t_\ell) = \bigcirc\bigcirc\bigcirc\, \square\, \square\, \square\, \square,$

$\text{code}(t_r) = \bigcirc\bigcirc\square\bigcirc\square\, \square\bigcirc\square\, \square$

$\text{code}(t) = \bigcirc\, \text{code}(t_\ell)\, \text{code}(t_r)$

$= \bigcirc\bigcirc\bigcirc\bigcirc\, \square\, \square\, \square\, \square\bigcirc\bigcirc\, \square\bigcirc\, \square\, \square\bigcirc\square\, \square$

Bemerkungen:

- streicht man in jedem von $G_{\mathcal{B}}$ erzeugten Wort das letzte Symbol und identifiziert man

  $$\bigcirc \sim \text{ "linke Klammer"} \qquad \square \sim \text{ "rechte Klammer"},$$

  so entsprechen diese Wörter genau den wohlgeformten Klammerausdrücken entsprechender Länge.
- die von $G_{\mathcal{B}}$ erzeugte Sprache ist kontextfrei, aber nicht regulär
- die Anzahl der binären Bäume mit n inneren (und $n+1$ äusseren) Knoten ist

$$c_n = \frac{1}{n+1}\binom{2n}{n} \in \Theta\left(\frac{4^n}{n^{3/2}}\right)$$

Parameter für Binärbäume (und auch für andere Typen von Bäumen) , die induktiv definiert oder beschrieben werden:

- $i(t) =$ Anzahl der inneren ("internal") Knoten von t:

$$i(t) = \begin{cases} 0 & \text{falls } t = \square \\ 1 + i(t_\ell) + i(t_r) & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$

- $e(t) =$ Anzahl der äusseren ("external") Knoten von t:

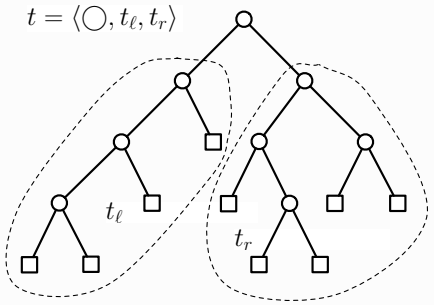
$$e(t) = \begin{cases} 1 & \text{falls } t = \square \\ e(t_\ell) + e(t_r) & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$

- $s(t) = i(t) + e(t) =$ Grösse ("size") von t:

$$s(t) = \begin{cases} 1 & \text{falls } t = \square \\ 1 + s(t_\ell) + s(t_r) & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$

- $h(t) =$ Höhe ("height") von t:

$$h(t) = \begin{cases} 0 & \text{falls } t = \square \\ 1 + \max\{h(t_\ell), h(t_r)\} & \text{falls } t = \langle \bigcirc, t_\ell, t_r \rangle \end{cases}$$



$i(t) = 8, e(t) = 9, s(t) = 17, h(t) = 4$

Für alle binären Bäume t gelten folgende Aussagen:

1. $e(t) = i(t) + 1$, und damit $s(t) = 2 \cdot i(t) + 1 = 2 \cdot e(t) - 1$

   Beweis

$$\begin{aligned} e(\square) - i(\square) &= 1 - 0 = 1 \\ e(\langle \bigcirc, t_\ell, t_r \rangle) - i(\langle \bigcirc, t_\ell, t_r \rangle) &= e(t_\ell) + e(t_r) - (i(t_\ell) + i(t_r) + 1) \\ &= e(t_\ell) - i(t_\ell) + e(t_r) - i(t_r) - 1 \\ &= 1 + 1 - 1 = 1 \end{aligned}$$

2. $h(t) \leq i(t)$

Beweis

$$
\begin{aligned}
h(\square) &= 0 = i(\square) \\
h(\langle \bigcirc, t_\ell, t_r \rangle) &= 1 + \max\{h(t_\ell), h(t_r)\} \\
&\leq 1 + \max\{i(t_\ell), i(t_r)\} \\
&\leq 1 + i(t_\ell) + i(t_r) \\
&= i(\langle \bigcirc, t_\ell, t_r \rangle)
\end{aligned}
$$

3. $e(t) \leq 2^{h(t)}$, also $\log e(t) \leq h(t)$

Beweis

$$
\begin{aligned}
e(\square) &= 1 = 2^0 = 2^{h(\square)} \\
e(\langle \bigcirc, t_\ell, t_r \rangle) &= e(t_\ell) + e(t_r) \\
&\leq 2^{h(t_\ell)} + 2^{h(t_r)} \\
&\leq 2 \cdot 2^{\max\{h(t_\ell), h(t_r)\}} \\
&= 2^{1+\max\{h(t_\ell), h(t_r)\}} \\
&= 2^{h(\langle \bigcirc, t_\ell, t_r \rangle)}
\end{aligned}
$$

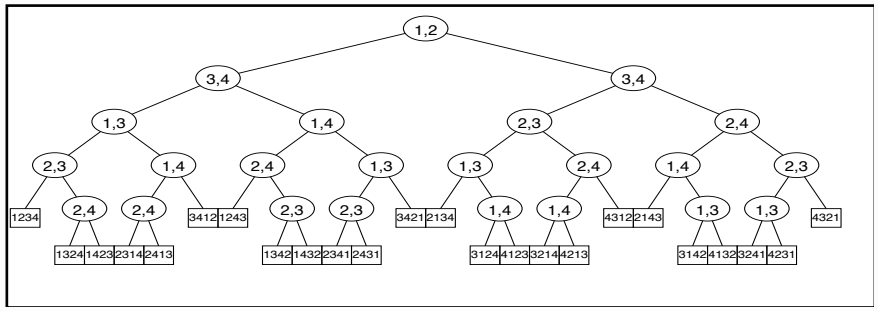
Untere Schranke für das Sortieren

Vergleichbasierte Sortieralgorithmen können mittels beschrifteter binärer Bäume (Entscheidungsbäume) veranschaulicht werden:

- Eingabe ist Liste $(x_1, x_2, \ldots, x_n)$ der Länge n, wobei die x_i paarweise verschieden (Permutationsmodell)
- innere Knoten sind mit Paaren (i, j) $(1 \leq i < j \leq n)$ beschriftet
- äussere Knoten (Blätter) sind mit Permutationen von $\{1, 2, \ldots, n\}$ beschriftet
- jeder Weg von der Wurzel zu einem Blatt identifiziert schrittweise die Struktur der Eingabe anhand der Inversionen:
  an einem mit (i, j) beschrifteten Knoten
  - gehe weiter in den linken Teilbaum, falls $x_i < x_j$
  - gehe weiter in den rechten Teilbaum, falls $x_i > x_j$

  an einem Blatt: die Permutation stellt die Grössenrelationen der Eingabe dar

Beispiel: mergesort



- für jeden binären Baum t gilt

$$e(t) \leq 2^{h(t)}$$

wobei $e(t)$ = Anzahl der Blätter, $h(t)$ = Höhe, also

$$\log e(t) \leq h(t)$$

- Operiert ein Sortieralgorithmus $\mathcal{A}$ auf Eingaben der Länge n,
so muss der zugehörige Entscheidungsbaum $t_{\mathcal{A}}(n)$ mindestens $n!$ Blätter haben,
denn alle $n!$ möglichen Permutationen müssen identifiziert werden können.
- Somit gilt für die Anzahl $V_{\mathcal{A}}(n)$ im worst-case

$$V_{\mathcal{A}}(n) = h(t_{\mathcal{A}}(n)) \geq \lceil \log e(t_{\mathcal{A}}(n)) \rceil \geq \lceil \log n! \rceil$$

- Erinnerung: STIRLINGs Formel

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi \cdot n} \left(1 + O(\frac{1}{n})\right)$$

- Theorem (GA, Th. 2.22):
Jeder vergleichsbasierte Sortieralgorithmus benötigt auf Folgen der Länge n im worst-case mindestens

$$n \log n - n \log e + O(\log n) \approx n \log n - 1.44n \quad \text{Vergleiche}$$

average-case Analyse

- Jeder Entscheidungsbaum für ein Sortierverfahren auf inputs der Länge n hat genau $n!$ Blätter
- mittlere Höhe eines Binärbaumes

$$\overline{h}(t) = \frac{1}{e(t)} \sum_{b \in E(t)} h(b, t)$$

wobei $E(t)$ = Menge der Blätter von t, $h(b,t)$ = Höhe des Blattes b in t

- Induktiv:

$$\overline{h}(\square) = 0$$

$$\overline{h}(\langle \bigcirc, t_\ell, t_r \rangle) = \frac{e(t_\ell)}{e(t)} \overline{h}(t_\ell) + \frac{e(t_r)}{e(t)} \overline{h}(t_r) + 1$$

- Lemma (GA, Lemma 2.24): Für jeden Binärbaum gilt

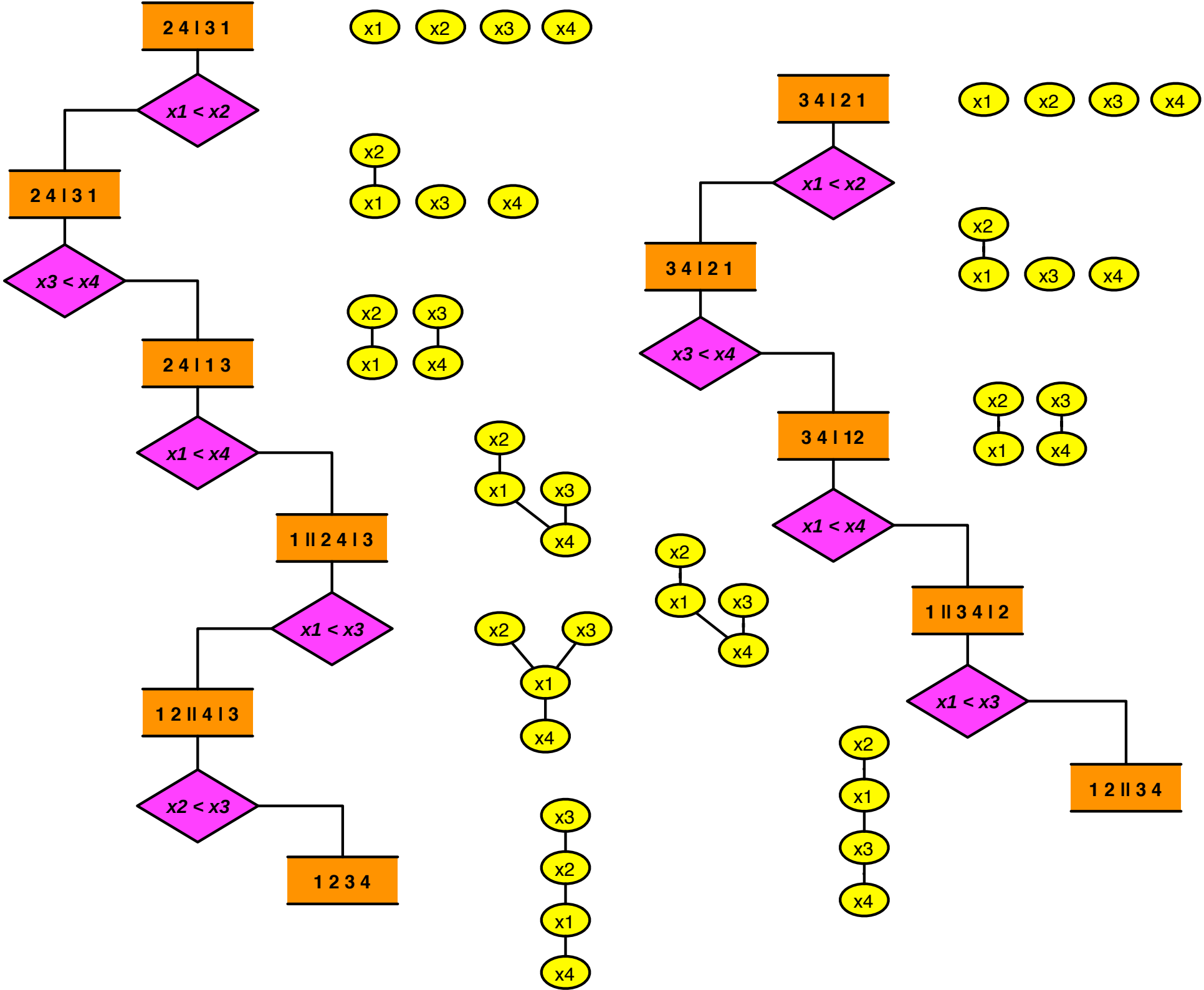
$$\overline{h}(t) \geq \log e(t)$$

- Theorem (GA, Theorem 2.26):
Jeder vergleichsbasierte Sortieralgorithmus benötigt auf Folgen der Länge n im average-case mindestens

$$n \log n - n \log e + O(\log n) \approx n \log n - 1.44n \quad \text{Vergleiche}$$

Bemerkungen

- Sortieralgorithmen, die $\Theta(n \log n)$ Vergleiche für inputs der Länge n benötigen (im worst-case bzw. average-case), sind *asymptotisch optimal*
- *Mergesort* und *Heapsort* sind asymptotisch optimale Sortierverfahren im worst-case (und daher auch im average-case)
- *Selectionsort* (*Bubblesort*) und *Insertionsort* sind weder im worst-case, noch im average-case, asymptotisch optimal
- *Quicksort* ist nur im average-case ein asymptotisch optimales Sortierverfahren
- Es gibt andere Sortieralgorithmen (z.B. *Bucketsort*) mit asymptotisch linearer Laufzeit: die sind aber nicht vergleichsbasiert oder machen einschränkende Annahmen über die Natur bzw. Verteilung der zu sortierenden Elemente


2 4 | 3 1
x1 < x2
2 4 | 3 1
x3 < x4
2 4 | 1 3
x1 < x4
1 || 2 4 | 3
x1 < x3
1 2 || 4 | 3
x2 < x3
1 2 3 4
x1 x2 x3 x4
x2 x1 x3 x4
x2 x3 x1 x4
x2 x1 x3 x4
x2 x1 x3 x4
x3 x2 x1 x4
3 4 | 2 1
x1 < x2
3 4 | 2 1
x3 < x4
3 4 | 12
x1 < x4
1 || 3 4 | 2
x1 < x3
1 2 || 3 4
x1 x2 x3 x4
x2 x1 x3 x4
x2 x3 x1 x4
x2 x1 x3 x4
x2 x1 x3 x4
x2 x1 x3 x4

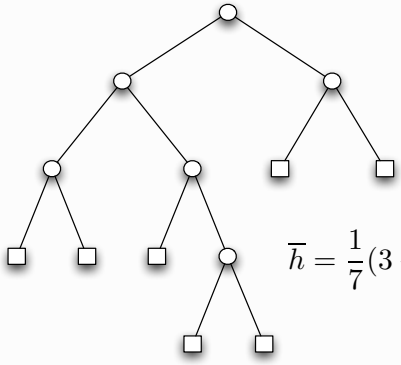
- mittlere Höhe von Binärbäumen
- Entropie
- gewichtete mittlere Höhe von Binärbäumen
- Quellcodierung, Datenkompression
- SHANNONs Theorem
- optimale Quellcodierung (HUFFMAN)

Zur mittleren Höhe von Binärbäumen

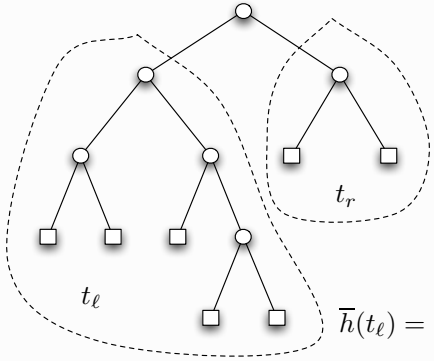
$$\overline{h}(t) = \frac{1}{e(t)} \sum_{b \in E(t)} h(b,t)$$

wobei

- t : Binärbaum
- $E(t)$: Menge der Blätter (external nodes) von t
- $e(t) = \sharp E(t)$: Anzahl der Blätter von t
- $h(b,t)$: Höhe des Blattes b in t



$$\overline{h} = \frac{1}{7}(3+3+3+4+4+2+2) = \frac{21}{7} = 3$$



$$\overline{h}(t) = 1 + \frac{5}{7} \cdot \overline{h}(t_\ell) + \frac{2}{7} \cdot \overline{h}(t_r) = 3$$

$$\overline{h}(t_r) = \frac{1}{2}(1+1) = \frac{2}{2}$$

$$\overline{h}(t_\ell) = \frac{1}{5}(2+2+2+3+3) = \frac{12}{5}$$

Zusammenhang mit dem rekursiven Aufbau von Binärbäumen

$$\begin{aligned}
\overline{h}(\square) &= 0 \\
\overline{h}(\langle \bigcirc, t_\ell, t_r \rangle) &= \frac{1}{e(t)} \sum_{b \in E(t)} h(b,t) \\
&= \frac{1}{e(t)} \left(\sum_{b \in E(t_\ell)} [h(b,t_\ell)+1] + \sum_{b \in E(t_r)} [h(b,t_r)+1] \right) \\
&= \frac{e(t_\ell)+e(t_r)}{e(t)} + \frac{e(t_\ell)}{e(t)} \cdot \overline{h}(t_\ell) + \frac{e(t_r)}{e(t)} \cdot \overline{h}(t_r) \\
&= 1 + \frac{e(t_\ell)}{e(t)} \cdot \overline{h}(t_\ell) + \frac{e(t_r)}{e(t)} \cdot \overline{h}(t_r)
\end{aligned}$$

Fundamentale Ungleichung für Binärbäume

$$\overline{h}(t) \geq \log e(t)$$

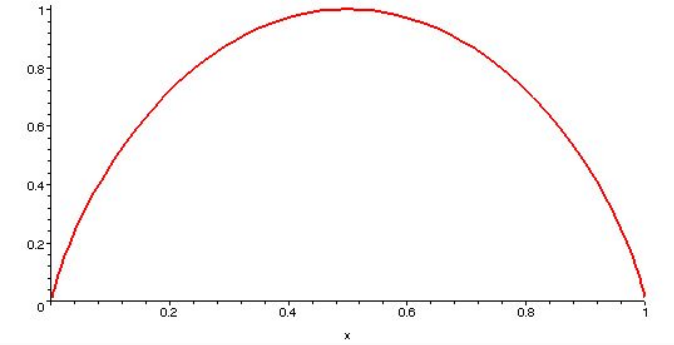
folgt per Induktion über der rekursiven Aufbau

$$\begin{aligned}
\overline{h}(\square) &= 0 = \log 1 \\
\overline{h}(\langle \bigcirc, t_\ell, t_r \rangle) &= 1 + \frac{e(t_\ell)}{e(t)} \cdot \overline{h}(t_\ell) + \frac{e(t_r)}{e(t)} \cdot \overline{h}(t_r) \\
&\geq 1 + \frac{e(t_\ell)}{e(t)} \cdot \log e(t_\ell) + \frac{e(t_r)}{e(t)} \cdot \log e(t_r) \\
&= 1 + \underbrace{\frac{e(t_\ell)}{e(t)} \cdot \log \frac{e(t_\ell)}{e(t)} + \frac{e(t_r)}{e(t)} \cdot \log \frac{e(t_r)}{e(t)}}_{-H(\frac{e(t_\ell)}{e(t)}, \frac{e(t_r)}{e(t)})} + \log e(t) \\
&= \underbrace{1 - H(\frac{e(t_\ell)}{e(t)}, \frac{e(t_r)}{e(t)})}_{\geq 0} + \log e(t)
\end{aligned}$$

Dabei ist

$$H(x, 1-x) = -x \log x - (1-x) \log(1-x)$$

die "Entropiefunktion"



SHANNONs Entropie

X : Zufallsvariable, die endlich-viele Werte, z.B. $1, 2, \ldots, n$ annimmt

$p_k = P[X = k]$: Wahrscheinlichkeit für Eintreten des Ereignisses "$X = k$"

also

$$p_k \geq 0 \ (1 \leq k \leq n) \ \text{ mit } \ \sum_{k=1}^{n} p_k = 1$$

Bezeichnung: $\boldsymbol{p} = \langle p_1, p_2, \ldots, p_n \rangle$

"Information(sgehalt)" des Ereignisses "$X = k$"

$$I[X = k] = -\log p_k$$

"Entropie" = Erwartungswert der Information

$$H(\boldsymbol{p}) = H(p_1, \ldots, p_n) = \mathbf{E}(I, \boldsymbol{p}) = -\sum_{k=1}^{n} p_k \log p_k$$

Eigenschaften der Entropiefunktion

1. $H(p_1, \ldots, p_n) \geq 0$
2. $H(p_1, \ldots, p_n) = 0 \;\Leftrightarrow\; p_k = \delta_{i,k}$ für ein $i \in \{1, \ldots, n\}$
3. $H(p_1, \ldots, p_n)$ wird maximal für die Gleichverteilung $p_1 = \ldots = p_n = \frac{1}{n}$
4. $H(p_1, \ldots, p_n)$ ist invariant unter Permutation der Komponenten p_k
5. $H(p_1, \ldots, p_n, 0) = H(p_1, \ldots, p_n)$
6. $H(1/n, \ldots, 1/n) \leq H(1/(n+1), \ldots, 1/(n+1))$
7. $H(p_1, \ldots, p_n)$ ist stetige Funktion der Variablen p_k
8. $H(1/mn, \ldots, 1/mn) = H(1/m, \ldots, 1/m) + H(1/n, \ldots, 1/n)$
9. für $p_1, \ldots p_m \geq 0$ und $q_1, \ldots, q_n \geq 0$ mit
   $p_1 + \cdots + p_m = p$, $q_1 + \cdots q_n = q$ und $p + q = 1$ gilt

$$H(p_1, \ldots, p_m, q_1 \ldots, q_n) = H(p, q) + p \cdot H(p_1/p, \ldots, p_n/p) + q \cdot H(q_1/q, \ldots, q_n/q)$$

Bemerkung:

- Alle diese Eigenschaften — ausgenommen vielleicht 3. (s.u.) — lassen sich ausgehend von der Definition leicht nachrechnen.
  Was wichtiger ist: diese Eigenschaften erscheinen als plausible Forderungen, wenn man ausgeht von der intuitiven Vorstellung von der Entropie als dem

  "Informationsgewinn beim Durchführen eines Experiments"

  So beschreibt 8. das Verhalten bei der Kombination unabhängiger und gleichverteilter Experimente zu einem "Produktexperiment" und 9. bei der Ausführung eines Experiments in zwei Stufen.

Zum Beweis der Eigenschaft 3. der Entropiefunktion:

Folgende Aussage bezeichnet man in der Informationstheorie als "key lemma":

Für Wahrscheinlichkeitsverteilungen $\boldsymbol{p} = (p_1, \ldots, p_n)$ und $\boldsymbol{q} = (q_1, \ldots, q_n)$ gilt

$$H(p_1, \ldots, p_n) \leq -\sum_{k=1}^{n} p_k \cdot \log q_k$$

und es gilt "=" genau dann, wenn $\boldsymbol{p} = \boldsymbol{q}$.

Als Konsequenz hat man, indem man für $\boldsymbol{q}$ die Gleichverteilung $q_1 = \ldots = q_n = \frac{1}{n}$ wählt:

$$H(p_1, \ldots, p_n) \leq \log n$$

Beweis des "key lemmas" :

Die Logarithmusfunktion ist konvex und es gilt daher

$$\ln x \leq x - 1 \quad (x > 0)$$

mit "=" genau dann, wenn $x = 1$. Also ist

$$\ln \frac{q_k}{p_k} \leq \frac{q_k}{p_k} - 1 \quad (1 \leq k \leq n)$$

$$\sum_{1 \leq k \leq n} p_k \ln \frac{q_k}{p_k} \leq \sum_{1 \leq k \leq n} q_k - \sum_{1 \leq k \leq n} p_k = 0$$

$$\sum_{1 \leq k \leq n} p_k \ln q_k \leq \sum_{1 \leq k \leq n} p_k \ln p_k$$

mit "=" genau dann, wenn $\boldsymbol{p} = \boldsymbol{q}$.

Theorem:

Die Eigenschaften 1.-9. bestimmen die Entropiefunktion eindeutig (bis auf einen konstanten Faktor), d.h.

Eine Funktion mit den Eigenschaften 1.-9. ist notwendig von der Form

$$H(p_1, \ldots, p_n) = -c \sum_{k=1}^{n} p_k \log p_k$$

mit einer Konstanten $c > 0$.

Binäre Bäume mit Gewichten (auf den Blättern)

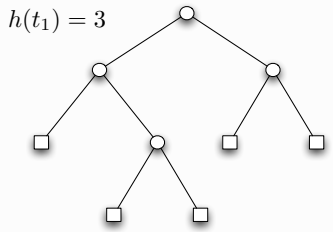
- betrachten $\langle t, \boldsymbol{p} \rangle$ wobei
  - t : binärer Baum
  - $\boldsymbol{p} = (p_b)_{b \in E(t)}$ Wahrscheinlichkeitsverteilung auf den Blättern von t
- untersuchen Erwartungswert der Höhe der Blätter
  ("gewichtete mittlere Höhe", "gewichtete externe Pfadlänge")

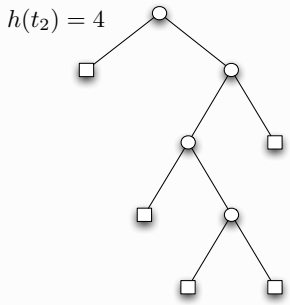
$$\overline{h}(t, \boldsymbol{p}) = \sum_{b \in E(t)} p_b \cdot h(b, t)$$

- für den Fall der Gleichverteilung auf $E(t)$ ist das gerade $\overline{h}(t)$
- Erinnerung:

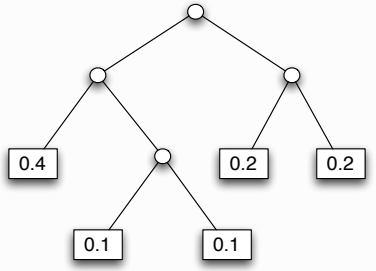
$$\overline{h}(t) \geq \log e(t)$$

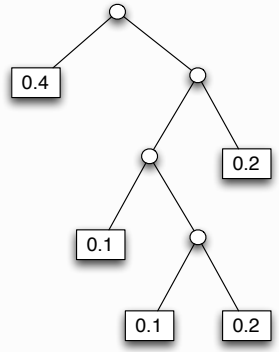
("information theory bound")



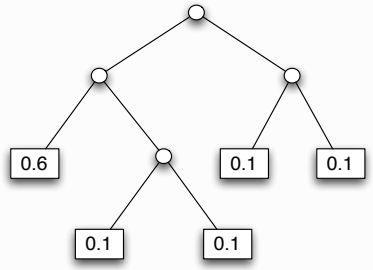


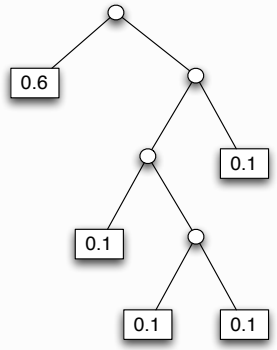
$$\overline{h}(t_1) = \frac{1}{5}(2+3+3+2+2) = \frac{12}{5} \quad \overline{h}(t_2) = \frac{1}{5}(1+3+4+4+2) = \frac{14}{5}$$





$$h(t_1, p) = 0.4 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 + 0.2 \cdot 2 + 0.2 \cdot 2 = 2.2$$
$$h(t_2, p) = 0.4 \cdot 1 + 0.1 \cdot 3 + 0.1 \cdot 4 + 0.2 \cdot 4 + 0.2 \cdot 2 = 2.3$$





$$h(t_1, q) = 0.6 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 + 0.1 \cdot 2 + 0.1 \cdot 2 = 2.2$$

$$h(t_2, q) = 0.6 \cdot 1 + 0.1 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 + 0.1 \cdot 2 = 1.9$$

Fundamentales Problem der Informationstheorie

- wie klein kann $\overline{h}(t, \boldsymbol{p})$ bei gegebenem $\boldsymbol{p}$ gemacht werden, indem man den Binärbaum t geeignet wählt?

Antwort: SHANNONs Quellcodierungstheorem

- untere Schranke: für jeden gewichteten Binärbaum $\langle t, \boldsymbol{p} \rangle$ gilt

$$H(\boldsymbol{p}) \leq \overline{h}(t, \boldsymbol{p})$$

- obere Schranke: zu jeder WV $\boldsymbol{p}$ gibt es einen Binärbaum t mit

$$\overline{h}(t, \boldsymbol{p}) < H(\boldsymbol{p}) + 1$$

Der Beweis für den ersten Teil (untere Schranke für $\overline{h}(t, \boldsymbol{p})$) geht genauso wie der Beweis für $\log e(t) \leq \overline{h}(t)$ im Fall der Gleichverteilung.

Sei $t = \langle \bigcirc, t_\ell, t_r \rangle$ Binärbaum mit Knotengewichten

- $(p_1, \ldots, p_m)$ auf $E(t_\ell) = (b_1, \ldots, b_m)$
- $(q_1, \ldots, q_n)$ auf $E(t_r) = (c_1, \ldots, c_n)$

Dabei sind

- $\boldsymbol{p} = (p_1, \ldots, p_m, q_1, \ldots, q_n)$
- $\boldsymbol{p}_\ell = (p_1/p, \ldots, p_m/p)$ mit $p = p_1 + \cdots + p_m$
- $\boldsymbol{p}_r = (q_1/q, \ldots, q_n/q)$ mit $q = q_1 + \cdots + q_n$

Wahrscheinlichkeitsverteilungen auf $E(t)$ bzw. $E(t_\ell)$ bzw. $E(t_r)$.

Dann gilt (Induktion!)

$$\begin{aligned}
\overline{h}(t, \boldsymbol{p}) &= \sum_{b \in E(t)} p_b \cdot h(b, t) \\
&= \sum_{b \in E(t_\ell)} p_b \cdot h(b, t) + \sum_{c \in E(t_r)} p_c \cdot h(c, t) \\
&= \sum_{1 \leq i \leq m} p_i \cdot (h(b_i, t_\ell) + 1) + \sum_{1 \leq j \leq n} q_j \cdot (h(c_j, t_r) + 1) \\
&= p + q + p \cdot \overline{h}(t_\ell, \boldsymbol{p}_\ell) + q \cdot \overline{h}(t_r, \boldsymbol{p}_r) \\
&\geq 1 + p \cdot H(\boldsymbol{p}_\ell) + q \cdot H(\boldsymbol{p}_r) \\
&= \underbrace{1 - H(p, q)}_{\geq 0} + H(\boldsymbol{p}) \ \geq H(\boldsymbol{p})
\end{aligned}$$

wegen Eigenschaften 3. und 9. der Entropiefunktion.

(Verlustfreie) Datenkompression durch Quellcodierung mit variabler Länge

- $A = \{a, b, c, \ldots\}$: endliche Menge von "Nachrichten" (Quellalphabet)
- $\{0, 1\}$: "Kanalalphabet"
- (binäre) Codierung ist injektive Abbildung

$$\Phi : A \to \{0,1\}^+$$

fortgesetzt zu Homomorphismus $\Phi : A^* \to \{0,1\}^*$,
falls die Decodierbedingung (*unique decipherability*) erfüllt ist:

(UD) für jedes $w \in \{0,1\}^*$ gilt $\sharp\Phi^{-1}(w) \leq 1$

- $\Phi(A) = \{\Phi(a), \Phi(b), \Phi(c), \ldots\}$: Menge der Codewörter, "Code"

- Beispiele ($A = \{a, b, c, d\}$)

$$\Phi_1 : \begin{cases} a \mapsto 00 \\ b \mapsto 01 \\ c \mapsto 10 \\ d \mapsto 11 \end{cases} \qquad \Phi_2 : \begin{cases} a \mapsto 0 \\ b \mapsto 111 \\ c \mapsto 110 \\ d \mapsto 101 \end{cases} \qquad \Phi_3 : \begin{cases} a \mapsto 01 \\ b \mapsto 011 \\ c \mapsto 110 \\ d \mapsto 101 \end{cases}$$

- Φ_1 : Codierung mit konstanter Länge

$$\Phi_1(abadc) = 00 \cdot 01 \cdot 00 \cdot 11 \cdot 10 = 0001001110$$

- Φ_2 : Codierung mit variabler Länge

$$\Phi_2(abadc) = 0 \cdot 111 \cdot 0 \cdot 101 \cdot 110 = 01110101110$$

- Φ_3 : keine Codierung!

$$\Phi_3(bda) = 011 \cdot 101 \cdot 01 = 01110101 = 01 \cdot 110 \cdot 101 = \Phi_3(acd)$$

- Eine Codierung Φ (ein Code $\Phi(A)$) hat die *Präfix-Eigenschaft* (ist ein *Präfixcode*), wenn gilt:

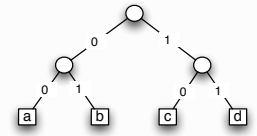
(PP) kein Codewort ist Präfix eines anderen Codewortes

- Φ_1 (und allgemein alle Codes konstanter Länge) sowie Φ_2 haben die (PE), die Abbildung Φ_3 nicht
- es gilt offensichtlich: (PP) $\Rightarrow$ (UD), aber (UD) $\not\Rightarrow$ (PP)
- Beispiele ($A = \{a, b, c, d, e\}$)

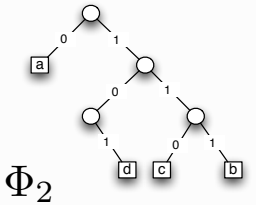
$$\Phi_4 : \begin{cases} a \mapsto 0 \\ b \mapsto 100 \\ c \mapsto 1010 \\ d \mapsto 1011 \\ e \mapsto 11 \end{cases} \qquad \Phi_5 : \begin{cases} a \mapsto 00 \\ b \mapsto 101 \\ c \mapsto 010 \\ d \mapsto 001 \\ e \mapsto 11 \end{cases}$$

Φ_4 hat (PP), Φ_5 hat (UD), aber nicht (PP)

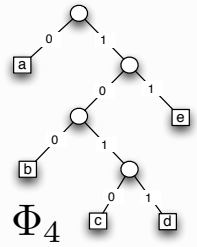
Präfixcodes sind binäre Bäume in einem erweiterten Sinn:
innere Knoten können einen (rechten oder linken) oder zwei (rechten und linken) Nachfolger haben



Φ_1



Φ_2



Φ_4

Vergleich von Codierungen — Datenkompression

		Φ_4	Φ_5
$aaaea$	$\mapsto$	000110	0000001100
$abaedbc$	$\mapsto$	0100011101110101011	0010001100101010

Welche Codierung ist "besser"?

Das hängt davon ab, mit welchen Wahrscheinlichkeiten die "Quellsymbole" a, b, c, d, e vorkommen!

- Eine *Quelle* $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ ist ein Paar, bestehend aus
  - einem *Quellalphabet* $A = \{a, b, c, \ldots\}$
  - einer Wahrscheinlichkeitsverteilung $\boldsymbol{p} = (p_a, p_b, p_c, \ldots)$ auf A.
- Für Codierungen $\Phi : A \to \{0,1\}^+$ einer Quelle $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ ist

$$\mu(\mathcal{Q}, \Phi) = \sum_{x \in A} p_x \cdot |\Phi(x)|$$

  mittlere oder *erwartete* Codewortlänge des Codes $\mathcal{C} = \Phi(A)$.
- Für Präfixcodes:

mittlere Codewortlänge $=$ gewichtete mittlere Höhe des entsprechenden Binärbaumes (im erweiterten Sinn)

Beispiel
$\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ mit $A = \{a, b, c, d\}$ und $\boldsymbol{p} = (p_a, p_b, p_c, p_d) = (0.9, 0.05, 0.025, 0.025)$

- $\Phi_1 : a \mapsto 00, b \mapsto 01, c \mapsto 10, d \mapsto 11$

$$\mu(\mathcal{Q}, \Phi_1) = 0.9 \cdot 2 + 0.05 \cdot 2 + 0.025 \cdot 2 + 0.025 \cdot 2 = 2$$

- $\Phi_2 : a \mapsto 0, b \mapsto 111, c \mapsto 110, d \mapsto 101$

$$\mu(\mathcal{Q}, \Phi_2) = 0.9 \cdot 1 + 0.05 \cdot 3 + 0.025 \cdot 3 + 0.025 \cdot 3 = 1.2$$

Beispiel
$\mathcal{Q} = \langle A, \boldsymbol{q} \rangle$ mit $A = \{a, b, c, d\}$ und $\boldsymbol{q} = (q_a, q_b, q_c, q_d) = (0.35, 0.25, 0.25, 0.15)$

- $\Phi_1 : a \mapsto 00, b \mapsto 01, c \mapsto 10, d \mapsto 11$

$$\mu(\mathcal{Q}, \Phi_1) = 0.35 \cdot 2 + 0.25 \cdot 2 + 0.25 \cdot 2 + 0.15 \cdot 2 = 2$$

- $\Phi_2 : a \mapsto 0, b \mapsto 111, c \mapsto 110, d \mapsto 101$

$$\mu(\mathcal{Q}, \Phi_2) = 0.35 \cdot 1 + 0.25 \cdot 3 + 0.25 \cdot 3 + 0.15 \cdot 3 = 2.3$$

Folgerung (SHANNON):

Bei gegebener Quellverteilung $\boldsymbol{p} = (p_a, p_b, p_c, \ldots)$ gibt die Entropie $H(\boldsymbol{p})$ ein Maß dafür an, welche Kompression (mittlere Wortlänge) bei keiner Codierung unterschritten werden kann.

Bemerkungen:

- für optimale Codierung kann man sich auf die Verwendung von (echten) Binärbäumen beschränken
- – Präfixcodes lassen sich "online" decodieren
  – Codes ohne (PP) lassen sich i.a. nicht "online" decodieren
- Verwendung von (UD)-Codes, die nicht die (PP) haben, kann die Situation nicht verbessern:

  zu jeder Quelle $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ und zu jedem (UD)-Code $\Phi(A)$ gibt es einen (PP)-Code $\Psi(A)$ mit $\mu(\mathcal{Q}, \Phi) = \mu(\mathcal{Q}, \Psi)$ (Satz von MACMILLAN)
- Konstruktion optimaler Präfixcodes: Verfahren von HUFFMAN

Existenz von Präfix-Codes mit gegebenen Wortlängen $\ell_1, \ell_2, \ldots, \ell_n$

- Es existiert ein Präfixcode $\{w_1, w_2, \ldots, w_n\} \subset \{0,1\}^*$ mit Wortlängen $|w_k| = \ell_k$ $(1 \leq k \leq n)$ genau dann, wenn

$$\sum_{1 \leq k \leq n} 2^{-\ell_k} \leq 1$$

(Ungleichung von KRAFT)

- Beachte: ist t ein Binärbaum (im ursprünglichen Sinn), so gilt immer

$$\sum_{b \in E(t)} 2^{-h(b,t)} = 1$$

Beweis der Ungleichung von KRAFT (notwendig):

Es sei $\ell_n = \max_{1 \leq k \leq n} \ell_k$. Die Wortmengen

$$w_k \cdot \{0,1\}^{\ell_n - \ell_k} \subseteq \{0,1\}^{\ell_n} \ (1 \leq k \leq n)$$

sind wegen (PP) paarweise disjunkt, deshalb

$$\sum_{1 \leq k \leq n} 2^{\ell_n - \ell_k} \leq 2^{\ell_n}$$

– Für $n = 1$ bzw. $n = 2$ leisten $\{0^{\ell_1}\}$ bzw. $\{0^{\ell_1}, 1^{\ell_2}\}$ das Gewünschte

– Sei die Behauptung für $n > 1$ bewiesen.
$1 \leq \ell_1 \leq \ldots \leq \ell_n \leq \ell_{n+1}$ genüge der Ungleichung von KRAFT.

Die gilt dann auch für die $n+1$ Zahlen

$$\ell_1, \ell_2, \ldots, \ell_{n-1}, \ell_n, \ell_n$$

und wegen $2^{-\ell_n} + 2^{-\ell_n} = 2^{-(\ell_n - 1)}$ auch für die n Zahlen

$$\ell_1, \ell_2, \ldots, \ell_{n-1}, \ell_n - 1$$

Es existiert also ein Präfixcode

$$\{w_1, w_2, \ldots, w_n\} \text{ mit } |w_k| = \ell_k \ (1 \leq k < n) \text{ und } |w_n| = \ell_n - 1$$

Der Präfixcode

$$\{w_1, w_2, \ldots, w_{n-1}, w_n 0, w_n 1^{\ell_{n+1} - \ell_n + 1}\}$$

leistet das Verlangte.

Beispiel zu Ungleichung von KRAFT

Wortlängen	Code
$2,3,3,3,5,7$	$\{00, 010, 110, 111, 01110, 0111111\}$
$2,3,3,3,4$	$\{00, 010, 110, 111, 0111\}$
$2,2,3,3$	$\{00, 01, 110, 111\}$
$2,2,2$	$\{00, 01, 11\}$
$1,2$	$\{0, 11\}$
0	$\{\epsilon\}$

Beweis des Theorems von SHANNON (obere Schranke)

$\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$: Quelle mit $A = \{a_1, \ldots, a_n\}$, $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$,
wobei $p_k > 0$ $(1 \leq k \leq n)$.

Mit

$$\ell_k = \lceil -\log p_k \rceil \quad (1 \leq k \leq n)$$

sei

$$-\log p_k \leq \ell_k < 1 - \log p_k \quad (1 \leq k \leq n)$$

und somit (linke Ungleichung)

$$\sum_{1 \leq k \leq n} 2^{-\ell_k} \leq 1$$

Es existiert also ein Präfixcode

$$\Phi : A \to \{w_1, w_2, \ldots, w_n\} : a_i \mapsto w_i \ (1 \leq i \leq n)$$

mit Wortlängen

$$|w_k| = \ell_k = 2^{\lceil -\log p_k \rceil} \quad (1 \leq k \leq n)$$

Eine Abschätzung für die mittlere Wortlänge dieses Codes ergibt sich aus der rechten Ungleichung

$$\mu(\mathcal{Q}, \Phi) = \sum_{1 \leq k \leq n} p_k \cdot \ell_k < \sum_{1 \leq k \leq n} p_k \cdot (1 - \log p_k) = 1 + H(\boldsymbol{p})$$

Konstruktion optimaler (Präfix-)Codes (HUFFMAN)

- *optimale* Codierung Φ für eine Quelle $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$

$$\mu(\mathcal{Q}, \Phi) = \min\{\, \mu(\mathcal{Q}, \Psi) \,;\, \Psi \text{ Codierung für } \mathcal{Q} \,\}$$

- Das Theorem von SHANNON garantiert für optimales Φ:

$$H(\boldsymbol{p}) \leq \mu(\mathcal{Q}, \Phi) < 1 + H(\boldsymbol{p})$$

- Die Konstruktion eines optimalen Φ kann mit Hilfe eines "GREEDY"-Algorithmus ausgeführt werden, der sich am Beweis der Ungleichung von KRAFT orientiert

Für eine Quelle $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ mit $\sharp A \geq 2$ gilt:

- ist Φ optimal für $\mathcal{Q}$ und sind $a, b \in A$ mit $p_a > p_b$,
  so ist $|\Phi(a)| \leq |\Phi(b)|$
  Begründung:
  andernfalls könnte man die Codierungen von a und b vertauschen, d.h.

$$\Psi : \begin{cases} a \mapsto \Phi(b) \\ b \mapsto \Phi(a) \\ c \mapsto \Phi(c) \quad (c \in A \setminus \{a, b\}) \end{cases}$$

  und damit die mittlere Codelänge verkleinern:

$$\begin{aligned} \mu(\mathcal{Q}, \Psi) &= \mu(\mathcal{Q}, \Phi) - p_a \cdot (|\Phi(a)| - |\Psi(a)|) - p_b \cdot (|\Phi(b)| - |\Psi(b)|) \\ &= \mu(\mathcal{Q}, \Phi) - \underbrace{(p_a - p_b) \cdot (|\Phi(a)| - |\Phi(b)|)}_{>0} \end{aligned}$$

- ist Φ optimal für $\mathcal{Q}$, so ist der Code $\Phi(A)$ ein Binärbaum
  (im strikten Sinn)

– ist Φ optimal für $\mathcal{Q}$, so kann man annehmen (d.h., durch Umordnung erreichen), dass es Symbole $a, b \in A$ mit minimalen Wahrscheinlichkeiten p_a, p_b gibt, d.h.

$$p_a, p_b \leq \min_{c \in A \setminus \{a,b\}} p_c,$$

die als Geschwister codiert sind, d.h. es gibt ein $w \in \{0,1\}^*$ mit

$$\Phi(a) = w \cdot 0 \quad \text{und} \quad \Phi(b) = w \cdot 1$$

Begründung:
Knoten auf dem höchsten Niveau eines Binärbaumes (im engeren Sinne) treten immer als Geschwisterpaare auf.
Durch Umordung der Wahrscheinlichkeiten auf dem höchsten Niveau kann man die angegebene Situation erreichen, ohne die mittlere Wortlänge zu ändern.

– Fusion von Quellsymbolen $a, b \in A$:

$$\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$$

$$A' = (A \setminus \{a,b\}) \cup \{\alpha\}$$

$$p'_x = \begin{cases} p_x & \text{für } x \in A \setminus \{a,b\} \\ p_a + p_b & \text{für } x = \alpha \end{cases}$$

$$\mathcal{Q}' = \langle A', \boldsymbol{p}' \rangle$$

- Φ (strikter) Präfixcode für $\mathcal{Q}$, bei dem $\Phi(a)$ und $\Phi(b)$ Geschwister sind, d.h. $\Phi(a) = w \cdot 0$, $\Phi(b) = w \cdot 1$ für ein $w \in \{0,1\}^*$

$$\Phi' : A \to \{0,1\}^+ : \begin{cases} x \mapsto \Phi(x) & \text{für } x \in A \setminus \{a,b\} \\ \alpha \mapsto w \end{cases}$$

  $\Rightarrow \Phi'$ (strikter) Präfixcode für $\mathcal{Q}'$

- Φ' (strikter) Präfixcode für $\mathcal{Q}'$

$$\Phi : A \to \{0,1\}^+ : \begin{cases} x \mapsto \Phi'(x) & \text{für } x \in A \setminus \{a,b\} \\ a \mapsto \Phi'(\alpha) \cdot 0 \\ b \mapsto \Phi'(\alpha) \cdot 1 \end{cases}$$

  $\Rightarrow \Phi$ (strikter) Präfixcode für $\mathcal{Q}$, bei dem $\Phi(a)$ und $\Phi(b)$ Geschwister sind

- hierbei gilt

$$\begin{aligned} \mu(\mathcal{Q}, \Phi) - \mu(\mathcal{Q}', \Phi') &= p_a \cdot |\Phi(a)| + p_b \cdot |\Phi(b)| - p'_\alpha \cdot |\Phi'(\alpha)| \\ &= p_a \cdot |\Phi(a)| + p_b \cdot |\Phi(b)| - (p_a + p_b) \cdot (|\Phi(a)| - 1) \\ &= p_a + p_b = p'_\alpha \end{aligned}$$

Folgerung

- Entsteht die Quelle $\mathcal{Q}' = \langle A', \boldsymbol{p}' \rangle$ aus der Quelle $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ durch Fusion zweier Symbole $a, b \in A$ mit minimalen Wahrscheinlichkeiten p_a, p_b, so gilt (mit dem obigen Zusammenhang zwischen Φ und Φ')

$$\Phi \text{ optimal für } \mathcal{Q} \quad \Leftrightarrow \quad \Phi' \text{ optimal für } \mathcal{Q}'$$

  Diese Aussage erlaubt die rekursive Konstruktion optimaler Präfixcodes.

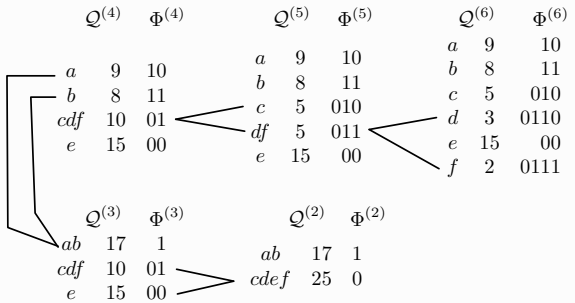
- Realisierung dieser Idee

$$\begin{array}{ccccccccc} \mathcal{Q}^{(2)} & \leftarrow & \mathcal{Q}^{(3)} & \leftarrow & \cdots & \leftarrow & \mathcal{Q}^{(n-1)} & \leftarrow & \mathcal{Q}^{(n)} = \mathcal{Q} \\ \Phi^{(2)} & \rightarrow & \Phi^{(3)} & \rightarrow & \cdots & \rightarrow & \Phi^{(n-1)} & \rightarrow & \Phi^{(n)} = \Phi \end{array}$$

Dabei

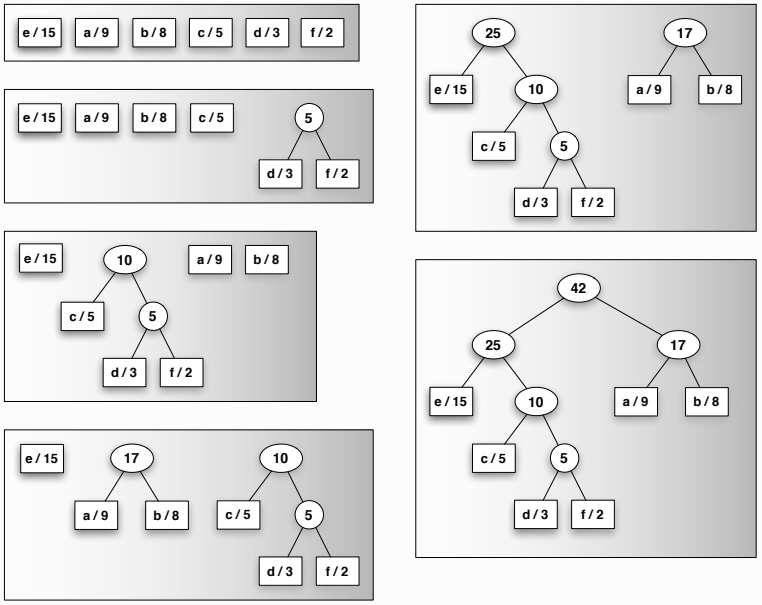
- $\mathcal{Q}^{(k)}$: Quelle mit k Symbolen
- $\Phi^{(k)}$: optimaler Präfixcode für $\mathcal{Q}^{(k)}$
- $\mathcal{Q}^{(k-1)} \leftarrow \mathcal{Q}^{(k)}$: Fusion von zwei Symbolen mit minimaler W.keit
- $\Phi^{(k-1)} \rightarrow \Phi^{(k)}$: Konstruktion entlang Umkehrung der Fusion
- $\mathcal{Q}^{(2)} = \langle \{a, b\}, (p_a, p_b) \rangle$
- $\Phi^{(2)} = \langle a \mapsto 0,\ b \mapsto 1 \rangle$

Beispiel zur Konstruktion von HUFFMAN
(mit Symbolhäufigkeiten statt Wahrscheinlichkeiten)



$$\mu(\mathcal{Q}^{(6)}, \Phi^{(6)}) = \frac{9 \cdot 2 + 8 \cdot 2 + 5 \cdot 3 + 3 \cdot 4 + 15 \cdot 2 + 2 \cdot 4}{42}$$
$$= \frac{5 + 10 + 17 + 25}{42} + 1 = \frac{99}{42} = 2.35714284\ldots$$

NB $\quad H(\frac{9}{42}, \frac{8}{42}, \frac{5}{42}, \frac{3}{42}, \frac{15}{42}, \frac{2}{42}) = 2.309050472\ldots$



Implementierung der HUFFMAN-Konstruktion

- Quelle $\mathcal{Q} = \langle A, \boldsymbol{p} \rangle$ mit $\sharp A$, $\boldsymbol{p} = (p_1, \ldots, p_n)$
- konstruiere Folge $F^{(n)}, F^{(n-1)}, \ldots, F^{(3)}, F^{(2)}, F^{(1)}$ von Wäldern

$$F^{(k)} = \{(t_1^{(k)}, g_1), \ldots, (t_k^{(k)}, g_k)\}$$

  $t_1^{(k)}, \ldots, t_k^{(k)}$ Binärbäume
  $g_1, \ldots, g_k$ Gewichte mit $g_1 \geq g_2 \geq \ldots \geq g_k$ und $g_1 + \cdots + g_k = 1$
  - $F^{(n)} = \{(\square, p_1), \ldots, (\square, p_n)\}$
  - $F^{(k)} \rightarrow F^{(k-1)}$: mit $s = \langle \bigcirc, t_{k-1}^{(k)}, t_k^{(k)} \rangle$, $\quad h = g_{k-1} + g_k$

$$F^{(k-1)} = \left(F^{(k)} \setminus \left\{ (t_{k-1}^{(k)}, g_{k-1}), (t_k^{(k)}, g_k) \right\} \right) \cup (s, h)$$

    (nach Gewichten ordnen!)
  - $t_1^{(1)}$ ist der HUFFMAN-Code

Komplexität der HUFFMAN-Konstruktion

- Information über die Gewichte g_j in einer priority queue organisieren! Diese Queue als min-heap implementieren.
- Aufbau des heaps: $\mathcal{O}(n)$
- $n-1$ heap-Operationen ($2\times$ DELETEMIN, $1\times$ REHEAP) mit Aufwand $\mathcal{O}(\log n)$
- Gesamtaufwand (Vergleichsoperationen): $\mathcal{O}(n \log n)$

- HUFFMANs Konstruktion ist ein klassischer GREEDY-Algorithmus.
- Weitere bekannte GREEDY-Algorithmen
  - Zahldarstellung in Positionssystemen (incl. FIBONACCI)
  - Minimale Gerüste (Spannbäume) (KRUSKAL, PRIM)
  - Kürzeste Wege (DIJKSTRA)
  - Knapsack ("fractional")
- GREEDY gehört mit DIVIDE-AND-CONQUER, DYNAMIC PROGRAMMING, BACKTRACKING mit BRANCH-AND-BOUND, RANDOMIZATION zu den fundamentalen Entwurfsprinzipien für Algorithmen
- das Typische an GREEDY-Problemen: optimale Lösungen von Teilproblemen lassen sich immer zu global-optimalen Lösungen fortsetzen
- GREEDY kann man nicht immer einsetzen, aber wenn ja, ist es sehr effizient
- man kann genau charakterisieren, in welchen Situationen GREEDY funktioniert ($\Rightarrow$ "Matriode")

Literaturhinweise

- V. HEUN behandelt in *Grundlegende Algorithmen* die HUFFMAN-Konstruktion in Kapitel 6.5 (Datenkompression). Alles über Datenkompression erfährt man in
  - S. C. SALOMON, *Data Compression — The Complete Reference*, Springer, 1997.
- Für eine Diskussion des Entropiebegriffs, Quellcodierung etc. muss man in Bücher über Informationstheorie schauen, z.B.
  - D. WELSH, *Codes and Cryptography*, Oxford UP, 1988.
  - R. J. MCELIECE, *The Theory of Information and Coding*, Addison-Wesley, 1977.
- Alle soliden Lehrbücher über Algorithmen(-entwurf) behandeln mehr oder weniger ausführlich die HUFFMAN-Konstruktion im Kontext der GREEDY-Algorithmen, siehe z.B. Kapitel 16 in
  - T. H. CORMEN, C. L. LEISERSON, R. L. RIVEST, C. STEIN, *An Introduction to Algorithms* (2nd. ed.), MIT Press, 2001.

  Dort erfährt man auch etwas über den theoretischen Hintergrund (Matroide).

- Zweidimensionale Quelle

$$\mathcal{P} = \langle A \times B, \mathbf{p} = (p_{a,b})_{(a,b)\in A\times B}\rangle$$

  A, B endliche Alphabete, $\mathbf{p} = (p_{a,b})_{(a,b)\in A\times B}$ WV (Matrix)
- Der Vektor $(r_a)_{a\in A}$ der Zeilensummen

$$r_a = \sum_{b\in B} p_{a,b}$$

  ist eine WV auf $A \to$ Quelle $\mathcal{R} = \langle A, (r_a)_{a\in A}\rangle$.
- Der Vektor $(c_b)_{b\in B}$ der Spaltensummen

$$c_b = \sum_{a\in A} p_{a,b}$$

  ist eine WV auf $B \to$ Quelle $\mathcal{C} = \langle C, (r_b)_{b\in B}\rangle$.
- Die Quellen $\mathcal{R}$ und $\mathcal{C}$ heissen Marginalquellen zu $\mathcal{P}$.
- $\mathcal{P}$ Produktquelle zu $\mathcal{R}, \mathcal{C}$, geschrieben $\mathcal{P} = \mathcal{R} \times \mathcal{C}$ falls

$$\mathbf{p} = \mathbf{r} \cdot \mathbf{c}, \quad \text{d.h. } p_{a,b} = r_a \cdot c_b \quad \text{für alle } (a,b) \in A \times B$$

- Beispiel: $\mathcal{P} = \langle A \times B, \mathbf{p}\rangle$ mit $A = \{a, b\}, B = \{x, y, z\}$ und

$$\mathbf{p} = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{pmatrix} 0.2 & 0.05 & 0.15 \\ 0.1 & 0.2 & 0.3 \end{pmatrix} \end{array}$$

  Marginalquellen:

$$\mathcal{R} = \langle A = \{a, b\}, \mathbf{r} = \{0.4, 0.8\}\rangle$$

$$\mathcal{C} = \langle B = \{x, y, z\}, \mathbf{c} = \{0.3, 0.25, 0.45\}\rangle$$

  Produkt der Marginalquellen: $\mathcal{R} \times \mathcal{C} = \langle A \times B, \mathbf{r} \cdot \mathbf{c}\rangle$ mit

$$\mathbf{r} \cdot \mathbf{c} = \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} \cdot \begin{pmatrix} 0.3 & 0.25 & 0.45 \end{pmatrix} = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{pmatrix} 0.12 & 0.1 & 0.18 \\ 0.18 & 0.15 & 0.27 \end{pmatrix} \end{array}$$

- Entropien

$$\begin{aligned} H(\mathcal{P}) &= 2.408694969\ldots \\ H(\mathcal{R}) &= 0.970950594\ldots \\ H(\mathcal{C}) &= 1.539491070\ldots \\ H(\mathcal{R} \times \mathcal{C} &= 2.510441664\ldots \end{aligned}$$

- Es gilt im Beispiel

$$\begin{aligned} H(\mathcal{P}) &\le H(\mathcal{R}) + H(\mathcal{C}) \\ H(\mathcal{R} \times \mathcal{C}) &= H(\mathcal{R}) + H(\mathcal{C}) \end{aligned}$$

- Zeigen Sie die Ungleichung

$$H(\mathcal{P}) \le H(\mathcal{R}) + H(\mathcal{C})$$

- Zeigen Sie, dass Gleichheit genau dann gilt, wenn die beiden Marginalquellen $\mathcal{R}$ und $\mathcal{C}$ unabhängig sind, d.h. wenn

$$p_{a,b} = r_a \cdot c_b \quad \text{für alle } (a,b) \in A \times B$$

  gilt, also $\mathcal{P} = \mathcal{R} \times \mathcal{C}$.

  Hinweis: das “key lemma” benutzen!

- Auf der Menge $\mathbb{B}^n$ der Bitstrings der Länge n als Alphabet wird mit Hilfe eines p mit $0 \leq p \leq 1$ eine Quelle definiert, bei der jedes $w = w_1 w_2 \ldots w_n$ die Wahrscheinlichkeit

$$bin_p^{(n)}(w) = p^{||w||}(1-p)^{n-||w||}$$

  erhält. Dabei ist $||w|| = \sharp_1(w)$ das sog. HAMMING-Gewicht.
- Welches ist die Entropie $H_p^{(n)}$ der Quelle $Q_p^{(n)} = \left(\mathbb{B}^n, bin_p^{(n)}\right)$? Drücken Sie dies mit Hilfe der Entropiefunktion $H(x, 1-x)$ aus.

- Berechnen Sie für die Quelle $Q_{1/8}^{(3)}$ deren Entropie, sowie einen optimalen binären Präfixcode und bestimmen Sie dessen mittlere (erwartete) Wortlänge.

  (Hinweis: verwenden Sie bei der Berechnung der Entropie den numerischen Wert $\log_2 7 = 2.80735\ldots$; bei der Berechnung des Codes ist es bequemer, mit Häufigkeiten statt mit Wahrscheinlichkeiten zu rechnen.)

- Sei $\mu_p^{(n)}$ die mittlere (erwartete) Wortlänge eines optimalen Präfixcodes für die Quelle $Q_p^{(n)}$. Zeigen Sie:

$$\lim_{n\to\infty} \frac{\mu_p^{(n)}}{n} = H(p, 1-p).$$

- A, B endliche Alphabete, A : Quellalphabet, B : Zielalphabet
- Ein (stationärer, gedächtnisfreier) Kanal gegeben durch Matrix

$$\mathbf{p} = \left[p_{a,b}\right]_{(a,b)\in A\times B}$$

  - $p_{a,b}$ ist die (bedingte) Wahrscheinlichkeit, dass $b \in B$ empfangen wird, falls $a \in A$ gesendet wurde
  - Die Kanalmatrix $\mathbf{p}$ ist eine *stochastische Matrix*, d.h. für jedes $a \in A$ ist $\mathbf{p}_a = (p_{a,b})_{b\in B}$ eine WV auf B, d.h. $\sum_{b\in B} p_{a,b} = 1$, m.a.W.

$$\mathcal{R}_a = (B, \mathbf{p}_a) \quad \text{ist eine Quelle auf } B$$

Man schreibt auch $p(b|a)$ für $p_{a,b}$ im Sinne von: "bedingte Wahrscheinlichkeit"

- Stationarität und Gedächtnisfreiheit des Kanals äussern sich in der Aussage für die Übertragung von Wörtern:

$$A^n \ni (a_1, a_2, \ldots, a_n) \mapsto (b_1, b_2, \ldots, b_n) \in B^n$$
$$\text{mit Wahrscheinlichkeit} \quad p_{a_1,b_1} \cdot p_{a_2,b_2} \cdots p_{a_n,b_n}$$

- Es genügt hier, die Übertragung von einzelnen Symbolen zu untersuchen.

1. Der störungsfreie Kanal: $A = B$ und

$$\mathbf{p} = E^{(m)}, \text{ d.h. } p_{a,b} = \delta_{a,b} = \begin{cases} 1 & \text{für } a = b \\ 0 & \text{für } a \neq b \end{cases}$$

2. Der total gestörte Kanal: $\sharp A = m, \sharp B = n$ und

$$\mathbf{p} = (p_{a,b})_{a\in A, b\in B} \text{ mit } p_{a,b} = \frac{1}{n} \quad (a \in A, b \in B)$$

3. Der binäre symmetrische Kanal BSC_p: $A = B = \mathbb{B} = \{0,1\}$ mit

$$\mathbf{p} = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix} \end{array}$$

4. Der binäre Auslöschungskanal BEC_p: $A = \mathbb{B} = \{0,1\}$, $B = A \cup \{*\}$ mit

$$\mathbf{p} = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & * & 1 \end{array} \\ \begin{pmatrix} 1-p & p & 0 \\ 0 & p & 1-p \end{pmatrix} \end{array}$$

- Eine Quelle $\mathcal{R} = (A, \mathbf{r})$ ist gegeben durch eine WV $\mathbf{r} = (r_a)_{a\in A}$ auf A
  - r_a ist die Wahrscheinlichkeit, dass $a \in A$ gesendet wird.
- Der Vektor

$$\mathbf{c} = (c_b)_{b\in B} = \mathbf{r} \cdot \mathbf{p}, \quad \text{d,h. } c_b = r_a \cdot p_{a,b} \quad (b \in B)$$

ist eine WV auf dem Zielalphabet B, d.h.

$$\mathcal{C} = (B, \mathbf{c}) \quad \text{ist eine Quelle auf } B$$

  - c_b ist die Wahrscheinlichkeit für das Empfangen von $b \in B$, falls die $a \in A$ mit Wahrscheinlichkeit p_a gesendet werden.

- Eine WV $\mathbf{q}$ auf $A \times B$ wird definiert durch

$$q_{a,b} = r_a \cdot p_{a,b} \quad (a \in A, b \in B)$$

$\mathcal{Q} = (A \times B, \mathbf{q})$ bezeichnet die die entsprechende Quelle.

- Für $(a, b) \in A \times B$ ist $q_{a,b}$ die Wahrscheinlichkeit für das Auftreten des Sende-Empfangspaares (a, b), falls die $a \in A$ mit Wahrscheinlichkeit p_a gesendet werden.
- Wegen

$$\sum_{b \in B} q_{a,b} = r_a \quad (a \in A) \quad \text{und} \quad \sum_{a \in A} q_{a,b} = c_b \quad (b \in B)$$

sind $\mathbf{r}$ und $\mathbf{c}$ die Marginalverteilungen von $\mathbf{q}$.

- Für jedes $b \in B$ ist

$$\mathbf{c}_b = \left(\frac{q_{a,b}}{c_b} \right)_{a \in A} =$$

eine WV auf A und $\mathcal{C}_b = (A, \mathbf{c}_b)$ eine Quelle auf dem Alphabet A.

- $\frac{q_{a,b}}{c_b}$ ist die (bedingte) Wahrscheinlichkeit, dass $a \in A$ gesendet worden ist, falls $b \in B$ empfangen wurde, auch als $p(a|b)$ geschrieben.

- Jede der Quellen $\mathcal{R}_a$ $(a \in A)$ hat die Entropie

$$H(\mathcal{R}_a) = -\sum_{b \in B} p_{a,b} \cdot \log p_{a,b}.$$

Die erwartete Entropie (gewichtete Entropie bezüglich der Quellverteilung $\mathbf{r}$ auf A) ist dann

$$H(\mathcal{C}|\mathcal{R}) = \sum_{a \in A} r_a \cdot H(\mathcal{R}_a).$$

Man spricht von der *bedingten Entropie* von $\mathcal{C}$ bezüglich $\mathcal{R}$.

- Jede der Quellen $\mathcal{C}_b$ $(b \in B)$ hat die Entropie

$$H(\mathcal{C}_b) = -\sum_{a \in A} (q_{a,b}/c_b) \cdot \log(q_{a,b}/c_b).$$

Die erwartete Entropie (gewichtete Entropie bezüglich der Quellverteilung $\mathbf{c}$ auf B) ist dann

$$H(\mathcal{R}|\mathcal{C}) = \sum_{b \in B} c_b \cdot H(\mathcal{C}_b).$$

Man spricht von der *bedingten Entropie* von $\mathcal{R}$ bezüglich $\mathcal{C}$.

- Die Entropie der Quelle $\mathcal{Q} = (A \times B, \mathbf{q})$ ist

$$H(\mathcal{Q}) = H(\mathbf{q}) = -\sum_{(a,b)\in A\times B} q_{a,b} \cdot \log q_{a,b}$$

- Es gilt

$$H(\mathcal{Q}) = H(\mathcal{R}) + H(\mathcal{C}|\mathcal{R}) = H(\mathcal{C}) + H(\mathcal{R}|\mathcal{C})$$

Beweis: Einsetzen und ausrechnen!

- Es gilt

$$H(\mathcal{Q}) \le H(\mathcal{R}) + H(\mathcal{C})$$

Beweis: $\mathbf{r}^t \cdot \mathbf{c} = (r_a \cdot c_b)_{(a,b)\in A\times B}$ ist eine WV auf $A \times B$.
Aus dem Fundamentallemma von GIBBS folgt:

$$\begin{aligned} H(Q) &= -\sum_{(a,b)\in A\times B} q_{a,b} \cdot \log q_{a,b} \le -\sum_{(a,b)\in A\times B} q_{a,b} \cdot \log(r_a \cdot c_b) \\ &= -\sum_{(a,b)\in A\times B} q_{a,b} \cdot \log r_a - \sum_{(a,b)\in A\times B} q_{a,b} \cdot \log c_b \\ &= -\sum_{a\in A} r_a \underbrace{(\sum_{b\in B} p_{a,b})}_{r_a} \cdot \log r_a - \sum_{b\in B} \underbrace{(\sum_{a\in A} q_{a,b})}_{c_b} \cdot \log c_b \\ &= H(\mathcal{R}) + H(\mathcal{C}), \end{aligned}$$

wobei Gleichheit genau dann gilt, wenn $q_{a,b} = r_a \cdot c_b$ ist für alle $(a, b) \in A \times B$, d.h. wenn $p_{a,b} = c_b$ unabhängig von $a \in A$ ist.

- Als *wechselseitige Information* eines Paares $(a, b) \in A \times B$ bezeichnet man

$$I(a, b) = \log \frac{q_{a,b}}{r_a \cdot c_b} = \log q_{a,b} - \log r_a - \log c_b$$

- $I(a, b)$ kann positiv oder negativ sein. Insbesondere

$$I(a, b) = 0 \quad \Leftrightarrow \quad q_{a,b} = r_a \cdot c_b \quad \Leftrightarrow \quad p_{a,b} = c_b$$

1. Der störungsfreie Kanal: es ist

$$q_{a,b} = \begin{cases} r_a & \text{für } a = b \\ 0 & \text{für } a \neq b \end{cases}$$

und somit

$$I(a, b) = \begin{cases} -\log c_b = -\log r_a & \text{für } a = b \\ \text{nicht def.} & \text{für } a \neq b \end{cases}$$

2. Der total gestörte Kanal: wegen $p_{a,b} = \frac{1}{n}$ ist

$$q_{a,b} = r_a \cdot p_{a,b} = \frac{r_a}{n} \quad (a \in A, b \in B)$$

$$c_b = \sum_{a\in A} q_{a.b} = \frac{1}{n} \quad (b \in B)$$

$$I(a, b) = \log \frac{q_{a,b}}{r_a \cdot c_b} = \log 1 = 0 \quad (a \in A, b \in B)$$

3. Der BSC_p mit $p = \frac{1}{4}$, $\mathbf{r} = \left(\frac{1}{2}, \frac{1}{2}\right)$:

$$\mathbf{c} = \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\end{pmatrix}\begin{pmatrix}\frac{3}{4} & \frac{1}{4}\\ \frac{1}{4} & \frac{3}{4}\end{pmatrix} = \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\end{pmatrix}$$

$$\mathbf{q} = \begin{pmatrix}\frac{3}{8} & \frac{1}{8}\\ \frac{1}{8} & \frac{3}{8}\end{pmatrix}$$

$$\left(\frac{q_{a,b}}{r_a \cdot c_b}\right)_{a,b\in\mathbb{B}} = \begin{pmatrix}\frac{3}{2} & \frac{1}{2}\\ \frac{1}{2} & \frac{3}{2}\end{pmatrix}$$

$$(I(a,b))_{a,b\in\mathbb{B}} = \begin{pmatrix}0.58\ldots & -1\\ -1 & 0.58\ldots\end{pmatrix}$$

4. Der BEC_p mit $p = \frac{1}{4}$ und $\mathbf{r} = \left(\frac{1}{2}, \frac{1}{2}\right)$

$$\mathbf{c} = \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\end{pmatrix}\begin{pmatrix}\frac{3}{4} & \frac{1}{4} & 0\\ 0 & \frac{1}{4} & \frac{3}{4}\end{pmatrix} = \begin{pmatrix}\frac{3}{8} & \frac{1}{4} & \frac{3}{8}\end{pmatrix}$$

$$\mathbf{q} = \begin{pmatrix}\frac{3}{8} & \frac{1}{8} & 0\\ 0 & \frac{1}{8} & \frac{3}{8}\end{pmatrix}$$

$$\left(\frac{q_{a,b}}{r_a \cdot c_b}\right)_{a,b\in\mathbb{B}} = \begin{pmatrix}2 & 1 & 0\\ 0 & 1 & 2\end{pmatrix}$$

$$(I(a,b))_{a,b\in\mathbb{B}} = \begin{pmatrix}1 & 0 & *\\ * & 0 & 1\end{pmatrix}$$

- Als *wechelseitige Information* zwischen Senderverteilung $\mathcal{R}$ und Empfängerverteilung $\mathcal{C}$ für den durch die Kanalmatrix $\mathbf{p}$ gegebenem Kanal bezeichnet man den Erwartungswert der wechselseitigen Information:

$$\begin{aligned} I_{\mathbf{p}}(\mathcal{R},\mathcal{C}) &= \sum_{a\in A, b\in B} q_{a,b}\cdot I(a,b)\\ &= H(\mathcal{R}) + H(\mathcal{C}) - H(\mathcal{Q}) \geq 0 \end{aligned}$$

Offensichtlich gilt auch

$$I_{\mathbf{p}}(\mathcal{R},\mathcal{C}) = H(\mathcal{R}) - H(\mathcal{R}|\mathcal{C}) = H(\mathcal{C}) - H(\mathcal{C}|\mathcal{R})$$

Die Gleichheit der beiden Differenzen folgt aus den obigen Darstellungen von $H(\mathcal{Q})$.

1. Der störungsfreie Kanal

$$I(\mathcal{R},\mathcal{C}) = -\sum_{a\in A} r_a \log r_a = H(\mathcal{R})$$

2. Der total gestörte Kanal

$$I(\mathcal{R},\mathcal{C}) = \sum_{a\in A, b\in B} \frac{r_a}{n}\cdot 0 = 0$$

3. Der $\mathrm{BSC}_{1/4}$

$$I(\mathcal{R},\mathcal{C}) = \frac{3}{8}\cdot\log\frac{3}{2} + \frac{1}{8}\cdot\log\frac{1}{2} + \frac{1}{8}\cdot\log\frac{1}{2} + \frac{3}{8}\cdot\log\frac{3}{2} \simeq 0.185\ldots$$

4. Der $\mathrm{BEC}_{1/4}$

$$I(\mathcal{R},\mathcal{C}) = \frac{3}{8}\cdot 1 + \frac{1}{8}\cdot 0 + \frac{1}{9}\cdot 0 + \frac{3}{8}\cdot 1 = \frac{6}{8} = 0.75$$

- Als *Kapazität* eines Kanals bezeichnet man

$$\max_{\mathbf{r}} I_{\mathbf{p}}(\mathcal{R}, \mathcal{C})$$

wobei über alle WVen $\mathbf{r}$ auf A maximiert wird (optimale Anpassung der Quelle $\mathcal{R}$ an die Übertragungseigenschaften des Kanals)

- Diese Grösse hängt also nur noch von der Kanalmatrix $\mathbf{p}$ ab, ist also ein Charakteristikum des Kanals.

- Die Kapazität des binären symmetrischen Kanals mit Fehlerwahrscheinlichkeit p ist $1 - H(p)$.
- Beweis: Für den BSC_p ist

$$\mathbf{p} = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

und damit

$$\begin{aligned} \mathbf{c} = \mathbf{r} \cdot \mathbf{p} &= \begin{bmatrix} r & 1-r \end{bmatrix} \cdot \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix} \\ &= \begin{bmatrix} r+p-2rp & 1-r-p+2rp \end{bmatrix}. \end{aligned}$$

Damit ist

$$H(\mathcal{C}) = -(r+p-2rp)\cdot\log(r+p-2rp) - (1-r-p+2rp)\cdot\log(1-r-p+2rp)$$

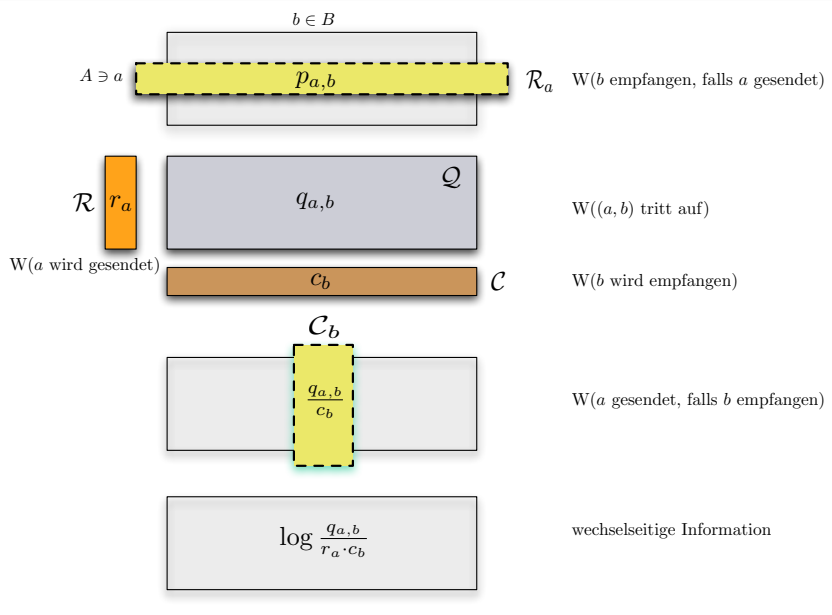
und

$$H(\mathcal{C}|\mathcal{R}) = r \cdot H(p) + (1-r) \cdot H(p) = H(p).$$

Insbesondere ist hier $H(\mathcal{C}|\mathcal{R})$ nur vom Kanalparameter p und nicht von der Quellverteilung $\mathbf{r} = (r, 1-r)$ anhängig!
Dann ist aber

$$\begin{aligned} \max_{\mathbf{r}} I_{\mathbf{p}}(\mathcal{R}, \mathcal{C}) &= \max_{\mathbf{r}} (H(\mathcal{C}) - H(\mathcal{C}|\mathcal{R})) \\ &= \max_{\mathbf{r}} H(\mathcal{C}) - H(p) \\ &= \max_{\mathbf{r}} H(r+p-2rp) - H(p) \end{aligned}$$

Die Entropiefunktion $H(x)$ wird maximal für $x = 1/2$.
Das gesuchte Maximum wird also angenommen, wenn $r + p - 2rp = 1/2 = 1 - r - p + 2rp$, also für $r = 1/2$.
Das Maximum hat den Wert 1.
Also ist $1 - H(p)$ der maximale Wert der wechselseitigen Information.



- Idee: Nicht-konstruktive Existenzbeweise für Strukturen mit bestimmten Eigenschaften
- Drei Beispiele
  - Das MaxCut-Problem
  - Das MaxSat-Problem
  - Das Ramsey-Problem
- Probabilistische Formulierung

- Das MaxCut-Problem
  - $G = (V, E)$ Graph mit Knotenmenge V, $\sharp V = n$, Kantenmenge $E \subseteq \binom{V}{2}$, $\sharp E = m$.
  - *Schnitt* von G: $c = (A, B)$ mit $A, B \subseteq (V), A \cap B = \emptyset, A \cup B = V$.
  - Wert von c:

$$v(c) := \sharp\{e \in E\,;\, e = \{a, b\} \text{ mit } a \in A, b \in B\}$$

  - MaxCut: bestimme Schnitt v, für den $v(c)$ maximal wird, bzw. bestimme

$$v(G) = \max_{c=(a,b) \text{ Schnitt von } G} v(c)$$

  - Das Problem MaxCut ist NP-hart! (im Gegensatz zum entsprechenden Minimierungsproblem → Flüsse in Netzwerken)

- Behauptung: $v(G) \geq \frac{m}{2}$, d.h. es gibt in jedem Graphen G immer einen Schnitt c mit $v(c) \geq \frac{m}{2}$.
- Beweis: Definiere für jede Kante $e \in E$ und jeden Schnitt $c = (A, B)$ von G:

$$\chi_e(c) := \begin{cases} 1 & \text{falls } e = \{a, b\} \text{ und } a \in A, b \in B \\ 0 & \text{sonst} \end{cases}$$

Dann ist für jeden Schnitt $c = (A, B)$ von G und jede Kante $e \in E$

$$v(c) = \sum_{e \in E} \chi_e(c) \quad \text{und} \quad \sum_{c \text{ Schnitt von } G} \chi_e(c) = 2^{n-1}$$

- (Forts.)

$$\begin{aligned} \sum_{c \text{ Schnitt v. } G} v(c) &= \sum_{c \text{ Schnitt v. } G} \sum_{e \in E} \chi_e(c) \\ &= \sum_{e \in E} \sum_{c \text{ Schnitt v. } G} \chi_e(c) \\ &= \sum_{e \in E} 2^{n-1} = m \cdot 2^{n-1} \end{aligned}$$

Die Summe (links) hat 2^n Glieder, es muss also mindestens ein Glied geben, das einen Beitrag $\geq \frac{m}{2}$ liefert!
Also ist $v(G) \geq \frac{m}{2}$.
Dies zeigt nicht, wie man einen solchen maximalen Schnitt findet!

- Das MaxSat-Problem
  - $X_n = \{x_1, x_2, \ldots, x_n\}$ aussagenlogische Variable,
    $V_n = X_n \cup \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_1}\}$ Literale
  - AL Formel in Klauselform (konjunktive Normalform)

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

    wobei die Klauseln C_i Disjunktionen von Literalen sind.
  - Keine Klausel enthalte zueinander komplementäre Literale.
  - Bewertung $\phi : X_n \to \{\mathbf{t}, \mathbf{f}\}$ wird zu Bewertung von Literalen, Klauseln und Formeln wie üblich fortgesetzt.
  - Wert von ϕ:

$$v(\phi) := \sharp\{C_i\,;\, \phi(C_i) = \mathbf{t}\}$$

  - MaxSat: finde zu Formel F Bewertung ϕ, die möglichst viele der Klauseln wahr macht, bzw. bestimme

$$v(F) = \max_{\phi} v(\phi)$$

  - Das Problem MaxSat ist NP-hart!

- $v(F) \geq \frac{m}{2}$, d.h. es gibt immer eine Bewertung ϕ, die mindestens die Hälfte der Klauseln von F wahr macht.
- Beweis: siehe Übungen!
- der Beweis zeigt nicht, wie man ein solches ϕ findet!

- Das Ramsey-Problem (einfachster Fall)
  - Zu $k \in \mathbb{N}$ sei $R(k)$ die kleinste Zahl $N \in \mathbb{N}$ mit der Eigenschaft:
    Jeder Graph $G = (V, E)$ mit $\sharp V \geq N$ enthält eine k-Clique oder eine k-Coclique.
  - Bekannte Werte: $R(2) = 2, R(3) = 6, R(4) = 18$.
  - Es gilt:

$$2^{\frac{k-2}{2}} \leq R(k) \leq \binom{2k-2}{k-1}.$$

    (also exponentielles Wachstum).

- Beweis der oberen Schranke
  - Definiere für $a, b \in \mathbb{N}$: $R(a, b)$ ist die kleinste Zahl $N \in \mathbb{N}$ mit der Eigenschaft:
    Jeder Graph $G = (V, E)$ mit $\sharp V \geq N$ enthält eine a-Clique oder eine b-Coclique.
  - Offenbar gilt $R(a, 1) = 1 = R(1, b)$ für alle $a, b \in \mathbb{N}$.
  - Es gilt die Rekursionsungleichung (siehe nächste Seite)

$$R(a+1, b+1) \leq R(a+1, b) + R(a, b+1) \quad (a, b \in \mathbb{N})$$

  - Aus den Induktionsannahmen

$$R(a+1, b) \leq \binom{a+b-1}{a} \quad R(a.b+1) \leq \binom{a+b-1}{b}$$

    folgt

$$R(a{+}1, b{+}1) \leq \binom{a+b-1}{a} + \binom{a+b-1}{b} \leq \binom{a+b}{a} = \binom{a+b}{b}$$

- Beweis der oberen Schranke (Forts.)
  - Es gilt also

$$R(a+1,b+1) \leq \binom{a+b}{a} \quad (a,b \in \mathbb{N})$$

und insbesondere

$$R(k+1) = R(k+1,k+1) = \binom{2k}{k}$$

- Beweis der Rekursionsungleichung:
  $G = (V,E)$ enthalte weder eine $(a+1)$-Clique, noch eine $(b+1)$-Coclique, d.h. es ist $\sharp V < R(a+1,b+1)$.
  $v \in V$ sei beliebiger Knoten,
  $A_v = \{u \in V\,;\, \{u,v\} \in E\}$, $B_v = \{u \in V\,;\, \{u,v\} \notin E\}$,
  - $G'_v = (A_v, E|_{A_v})$ enthält weder eine a-Clique, noch eine $(b+1)$-Coclique, also $\sharp A_v < R(a,b+1)$,
  - $G''_v = (B_v, E|_{B_v})$ enthält weder eine $(a+1)$-Clique, noch eine b-Coclique, also $\sharp B_v < R(a+1,b)$.

  Somit ist

$$\sharp V = \sharp A_v + \sharp B_v + 1 < R(a,b+1) + R(a+1,b)$$

und speziell mit $\sharp V = R(a+1,b+1) - 1$ folgt:

$$R(a+1,b+1) \leq R(a+1,b) + R(a,b+1)$$

- Beweis der unteren Schranke
  - Ω_n = alle Graphen G mit Knoten $V_n = \{1,2,\ldots,n\}$
  - $\sharp\Omega_n = 2^{\binom{n}{2}}$
  - Für $G \in \Omega_n$, $A \subseteq V_n$ mit $\sharp A = k$ sei

$$\chi_A(G) = \begin{cases} 1 & \text{falls } A \text{ eine } k\text{-Clique oder } k\text{-Coclique von } G \\ 0 & \text{sonst} \end{cases}$$

  - Anzahl der k-Clique und k-Cocliquen eines Graphen G

$$v(G) = \sum_{\substack{A \subseteq V_n \\ \sharp A = k}} \chi_A(G)$$

  - Offensichtlich gilt für jedes $A \subseteq V_n$ mit $\sharp A = k$:

$$\sum_{G \in \Omega_n} \chi_A(G) = 2 \cdot 2^{\binom{n}{2} - \binom{k}{2}}$$

- Beweis der unteren Schranke (Forts.)
  - Nun folgt

$$(*) \quad \sum_{G \in \Omega_n} v(G) = \sum_{G \in \Omega_n} \sum_{\substack{A \subseteq V_n \\ \sharp A = k}} \chi_A(G)$$
$$= \sum_{\substack{A \subseteq V_n \\ \sharp A = k}} \sum_{G \in \Omega_n} \chi_A(G) = \binom{n}{k} \cdot 2 \cdot 2^{\binom{n}{2} - \binom{k}{2}}$$

  - Die Summe $(*)$ hat $2^{\binom{n}{2}}$ Glieder.
  - Speziell für $n = 2^{\frac{k-2}{2}}$ ist

$$\binom{n}{k} \cdot 2 \cdot 2^{\binom{n}{2} - \binom{k}{2}} < n^k \cdot 2 \cdot 2^{\binom{n}{2} - \binom{k}{2}} = 2^{\binom{n}{2} + 1 - \frac{k}{2}} < 2^{\binom{n}{2}}$$

  - Mindestens einer der Terme der Summe $(*)$ muss einen Beitrag < 1 liefern.

- Beweis der unteren Schranke (Forts.)
  - Folgerung: es muss ein $G \in \Omega_n$ geben mit $v(G) < 1$. Dann bleibt aber nur $v(G) = 0$, d.h. G hat weder eine k-Clique, noch eine k-Coclique.
  - Es ist nicht klar, wie man ein solches G findet!

- Das eben gezeigte Resultat ist der Spezialfall eines Satzes von F.P. Ramsey (1930):
- r, k, ℓ seien natürliche Zahlen. Dann gibt es eine natürliche Zahl $N = N(r, k, \ell)$ mit der Eigenschaft:
  - Färbt man die k-elementigen Teilmengen einer N-elementigen Menge A auf irgendeine Weise mit r Farben, so gibt es immer eine ℓ-elementige Menge $B \subseteq A$, deren sämtliche k-elementigen Teilmengen dieselbe Farbe haben.
- Der vorangehende Satz behandelt den Fall $k = 2, r = 2$.
- Der Fall $k = 1$ entspricht dem Schubfachprinzip.
- Der Beweis im allgemeinen Fall benutzt eine analoge Induktion.
- Daraus hat sich eine umfangreiche Theorie entwickelt, die nach diesem englischen Logiker benannt ist.

- Simple Tatsache:
  - Ω sei endliche Menge, $P : \Omega \to [0, 1]$ Wahrscheinlichkeitsverteilung auf Ω, d.h. $\sum_{\omega \in \Omega} P(\omega) = 1$.
  - $X : \Omega \to \mathbb{R}$ Zufallsvariable mit Erwartungswert

$$E[X] = \sum_{\omega \in \Omega} P(\omega) \cdot X(\omega)$$

  - Dann gibt es ein $\omega \in \Omega$ mit $X(\omega) \geq E[X]$ (analog mit $\leq$).

- Erweiterung:
  - $X_1, X_2, \ldots, X_m$ seien Zufallsvariable, $X = X_1 + X_2 + \cdots + X_m$ (Unabhängigkeit wird nicht vorausgesetzt!)
  - Dann gibt es ein $\omega \in \Omega$ mit $X(\omega) \geq E[X] = \sum_{1 \leq i \leq m} E[X_i]$ (analog mit $\leq$).
  - Das ist von Interesse, wenn man die $E[X_i]$ und damit $E[X]$ ausrechnen kann.

- MaxCut-Problem
  - $G = (V, E)$ endlicher Graph mit $\sharp V = n$ und $\sharp E = n$.
  - $\Omega_n := \{\text{Schnitte } c = (A, B) \text{ von } G\}$ mit Gleichverteilung $P_n(c) = \frac{1}{2^n}$ $(c \in \Omega_n)$
  - Für jedes $e = \{a, b\} \in E$ ist $\chi_e : \Omega_n \to \{0, 1\}$ Zufallsvariable auf (Ω_n, P_n) mit
$$E[\chi_e] = \frac{1}{2}$$
  - $v = \sum_{e \in E} \chi_e$ ist Zufallsvariable auf (Ω_n, P_n) mit
$$E[v] = \sum_{e \in E} E[\chi_e] = \frac{m}{2}$$
  - Es muss also ein $c \in \Omega_n$ geben mit $v(c) \geq \frac{m}{2}$.

- MaxSat-Problem
  - $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ aussagenlogische Formel in Klauselform mit Variablen $X_n = \{x_1, x_2, \ldots, x_n\}$
  - $\Omega_n = \{\text{Bewertungen } \phi : X_n \to \{\mathbf{t}, \mathbf{f}\}\}$ mit Gleichverteilung $P_n(\phi) = \frac{1}{2^n}$
  - Für jede Klausel C_i ist $\chi_i : \Omega_n \to \{0, 1\}$ eine Zufallsvariable auf (Ω_n, P_n) mit
$$E[\chi_i] = 1 - \frac{1}{2^{n-k}} \geq \frac{1}{2}$$
  - $v = \sum_{1 \leq i \leq m} \chi_e$ ist Zufallsvariable auf (Ω_n, P_n) mit
$$E[v] = \sum_{1 \leq i \leq m} E[\chi_e] \geq \frac{m}{2}$$
  - Es muss also ein $\phi \in \Omega_n$ geben mit $v(\phi) \geq \frac{m}{2}$.

- Ramsey-Problem
  - $\Omega_n = \{\text{Graphen } G = (V_n, E)\}$, wobei $V_n = \{1, 2, \ldots, n\}$ mit Gleichverteilung $P_n(G) = 2^{-\binom{n}{2}}$
  - Für $A \subseteq V_n$ mit $\sharp A = k$ ist $\chi_A : \Omega_n \to \{0, 1\}$ eine Zufallsvariable auf (Ω_n, P_n) mit
$$E[\chi_A] = 2^{-\binom{k}{2}+1}$$
  - $v = \sum_{\substack{A \subseteq V_n \\ \sharp A = k}} \chi_A$ ist Zufallsvariable auf (Ω_n, P_n) mit
$$E[v] = \sum_{\substack{A \subseteq V_n \\ \sharp A = k}} E[\chi_A] = \binom{n}{k} \cdot 2^{-\binom{k}{2}+1}$$
  - Für $n \geq 2^{\frac{k-2}{2}}$ ist $E[v] < 1$. Es muss dann also ein $G \in \Omega_n$ geben mit $v(G) < 1$, d.h. $v(G) = 0$.

- $\mathbb{B}^n = \{0,1\}^n$: Bitvektoren der Länge n
- $\|\mathbf{a}\|$: HAMMING-Gewicht von $\mathbf{a}$
- $\mathbb{B}^n_k$: Vektoren $u \in \mathbb{B}^n$ mit HAMMING-Gewicht $= k$.
- $\mathbb{B}^n_{\leq k}$: Vektoren $u \in \mathbb{B}^n$ mit HAMMING-Gewicht $\leq k$.
- $S_k(\mathbf{a}) = \mathbf{a} \oplus \mathbb{B}^n_{\leq k}$: HAMMING-Kugel mit Radius k um $\mathbf{a}$.
- $\mathtt{bin}_{n,p}$: Binomialverteilung zum Parameter p $(0 < p < 1)$ auf $\mathbb{B}^n$, d.h.

$$\begin{aligned}\mathtt{bin}_{n,p}(\mathbf{a}) &= \mathtt{bin}_{n,p}(a_1, a_2, \ldots, a_n) \\ &= \prod_{1 \leq i \leq n} p^{a_i}(1-p)^{1-a_i} = p^{\|\mathbf{a}\|}(1-p)^{n-\|\mathbf{a}\|}\end{aligned}$$

Fakten:

- $\sharp\mathbb{B}^n = 2^n$,
- $\sharp\mathbb{B}^n_k = \binom{n}{k}$,
- $\sharp\mathbb{B}^n_{\leq k} = \sum_{0 \leq j \leq k} \binom{n}{j}$.
- $\binom{n}{k} \sim_{n\to\infty} \frac{n^k}{k!}$
- $\sum_{0 \leq j \leq \lambda \cdot n} \binom{n}{j} \sim_{n\to\infty} 2^{n \cdot H(\lambda)}$

- Parameter der Binomialverteilung
  - Mittelwert

$$\mu_{n,p} = \sum_{\mathbf{a} \in \mathbb{B}^n} \|\mathbf{a}\| \cdot \mathtt{bin}_{n,p}(\mathbf{a}) = \sum_{k=0}^{n} k \cdot \binom{n}{k} \cdot p^k (1-p)^{n-k} = n \cdot p$$

  - Varianz

$$\begin{aligned}\sigma^2_{n,p} &= \sum_{\mathbf{a} \in \mathbb{B}^n} (\|\mathbf{a}\| - \mu_{n,p})^2 \cdot \mathtt{bin}_{n,p}(\mathbf{a}) \\ &= \sum_{k=0}^{n} (k - \mu_{n,p})^2 \cdot \binom{n}{k} \cdot p^k (1-p)^{n-k} = n \cdot p \cdot (1-p)\end{aligned}$$

- CHEBYCHEV-Abschätzung für die Binomialverteilung

$$\mathtt{bin}_{n,p}\{\mathbf{a} \in \mathbb{B}_n ;\ |\|\mathbf{a}\| - \mu_{n,p}| \geq c \cdot \sigma_{n,p}\} \leq \frac{1}{c^2}$$

Beweis: mit

$$X = \{\mathbf{a} \in \mathbb{B}^n; \left|\|\mathbf{a}\| - \mu_{n,p}\right| \geq c \cdot \sigma_{n,p}\}$$

$$Y = \mathbb{B}^n \setminus X = \{\mathbf{a} \in \mathbb{B}^n; \left|\|\mathbf{a}\| - \mu_{n,p}\right| < c \cdot \sigma_{n,p}\}$$

gilt

$$\begin{aligned}\sigma^2_{n,p} &= \left(\sum_{\mathbf{a} \in X} + \sum_{\mathbf{a} \in Y}\right) (\|\mathbf{a}\| - \mu_{n,p})^2 \cdot \mathtt{bin}_{n,p}(\mathbf{a}) \\ &\geq c^2 \cdot \sigma^2_{n,p} \cdot \mathtt{bin}_{n,p}(X)\end{aligned}$$

- Kanalmodell: BSC_p binärer symmetrischer Kanal (ohne Gedächtnis) mit Fehlerwahrscheinlichkeit p:

$$\mathbb{B}_n \ni \mathbf{a} \rightsquigarrow \mathbf{b} = \mathbf{a} \oplus \mathbf{f} \quad \text{mit Wahrscheinlichkeit } \mathtt{bin}_{n,p}(\mathbf{f})$$

- (n, K)-*Code* : Teilmenge $\mathcal{C} \subset \mathbb{B}^n$ mit $\sharp\mathcal{C} = K$.
- *Coderate* von $\mathcal{C}$:

$$R(\mathcal{C}) = \frac{1}{n} \cdot \log_2 \sharp\mathcal{C}, \quad \text{also} \quad K = 2^{n \cdot R(\mathcal{C})}$$

- Decodierung mit Radius r $(0 \leq r < n)$:
  wird $\mathbf{a} \in \mathcal{C}$ gesendet und $\mathbf{b} \in \mathbb{B}^n$ empfangen,

$$\mathbf{a} \rightsquigarrow \mathbf{b} = \mathbf{a} \oplus \mathbf{f}$$

  so wird decodiert zu:
  - $\mathbf{a}' \in \mathcal{C}$, falls $\mathbf{a}'$ das einzige Element von $\mathcal{C} \cap S_r(\mathbf{b})$ ist;
  - Fehlanzeige (oder beliebiges Element von $\mathcal{C}$), falls $\sharp\big(\mathcal{C} \cap S_r(\mathbf{b})\big) \neq 1$.
- Fehlertypen:
  - Fehler 1. Art: $\|\mathbf{f}\| > r$.
  - Fehler 2. Art: $\|\mathbf{f}\| \leq r$, aber $\sharp\big(\mathcal{C} \cap S_r(\mathbf{b})\big) \geq 2$.

- Fehler 1. Art
  Sei $\epsilon > 0$ und $r = \lfloor n \cdot p + \sqrt{\frac{2}{\epsilon}} \cdot \sigma_{n,p} \rfloor$. Dann gilt

$$\begin{aligned} P^{(1)} &= \mathtt{bin}_{n,p}\{\mathbf{f} \in \mathbb{B}^n ;\ \|\mathbf{f}\| > r\} \\ &\leq \mathtt{bin}_{n,p}\left\{\mathbf{f} \in \mathbb{B}^n ;\ |\|\mathbf{f}\| - n \cdot p| > \sqrt{\frac{2}{\epsilon}} \cdot \sigma_{n,p}\right\} \leq \frac{\varepsilon}{2} \end{aligned}$$

  wegen der CHEBYCHEV-Abschätzung.

- Fehler 2. Art
  Für einen (n, K)-Code $\mathcal{C}$ mit r-Decodierung und Vektoren $\mathbf{a}, \mathbf{b} \in \mathbb{B}^n$ wird definiert:

$$\chi_{\mathcal{C},r}(\mathbf{a}, \mathbf{b}) = \begin{cases} 1 & \text{falls es ein } \mathbf{a}' \neq \mathbf{a} \text{ gibt mit } \mathbf{a}, \mathbf{a}' \in \mathcal{C} \cap S_r(\mathbf{b}) \\ 0 & \text{sonst.} \end{cases}$$

- Dann gilt bei Summation über alle (n, K)-Codes $\mathcal{C}$ für $\mathbf{a} \in \mathcal{C} \cap S_r(\mathbf{b})$ und mit $t = \sharp\mathbb{B}_{\leq r}$:

$$\begin{aligned} N = \sum_{\mathcal{C}} \chi_{\mathcal{C},r}(\mathbf{a}, \mathbf{b}) &= \binom{2^n - 1}{K - 1} - \binom{2^n - t}{K - 1} \\ &= \binom{2^n - 1}{K} \frac{K}{2^n - K} - \binom{2^n - t}{K} \frac{K}{2^n - K - t + 1}, \end{aligned}$$

  unabhängig von $\mathbf{a}$ und $\mathbf{b}$.

Dabei ist

$$\frac{\binom{2^n-1}{K}}{\binom{2^n}{K}} = \frac{(2^n-1)\cdots(2^n-K)}{(2^n)\cdots(2^n-K+1)} = 1-\frac{K}{2^n}$$

und

$$1 > \frac{\binom{2^n-t}{K}}{\binom{2^n}{K}} > 1-t\cdot\frac{K}{2^n}$$

Die letzte Ungleichung folgt aus der Tatsache, dass für $0 \le k \le n$ gilt:

$$\frac{\binom{n-x}{k}}{\binom{n}{k}} > 1-x\cdot\frac{k}{n} \quad \text{für} \quad 0 < x < 1.$$

Beide Polynome nehmen für $x = 0$ und $x = 1$ gleiche Werte an und das Polynom auf der linken Seite hat $n, n-1, \ldots, n-k+1$ als einfache Nullstellen, also negative erste und positive zweite Ableitung im Intervall $0 \le x \le 1$.

- Annahme:
  - ein Code $\mathcal{C}$ wird aus der Menge $\mathcal{C}_{n,K}$ aller (n,K)-Codes mit Gleichverteilung genommen,
  - $\mathbf{a}$ wird aus $\mathcal{C}$ wird mit Gleichverteilung gezogen und übertragen,
  - $\mathbf{a} \rightsquigarrow \mathbf{b}$ mit Wahrscheinlichkeit $\mathtt{bin}_{n,p}(\mathbf{a} \oplus \mathbf{b})$.
- Abschätzung für die mittlere Wahrscheinlichkeit für das Auftreten von Fehlern der zweiten Art:

$$\begin{aligned}\overline{P}^{(2)} &= \frac{1}{\binom{2^n}{K}\cdot K}\sum_{\substack{\mathcal{C}\in\mathcal{C}_{n,K}\\ \mathbf{a},\mathbf{b}\in\mathbb{B}^n}} \chi_{\mathcal{C},r}(\mathbf{a},\mathbf{b})\cdot \mathtt{bin}_{n,p}(\mathbf{a}\oplus\mathbf{b}) \\ &= \frac{N}{\binom{2^n}{K}\cdot K}\sum_{\mathbf{a}\in\mathbb{B}^n}\sum_{\mathbf{b}\in\mathbb{B}^n_{\le r}} \mathtt{bin}_{n,p}(\mathbf{b}) \\ &= \frac{N\cdot 2^n}{\binom{2^n}{K}\cdot K}\cdot \mathtt{bin}_{n,p}(\mathbb{B}^n_{\le r}) \le \frac{N\cdot 2^n}{\binom{2^n}{K}\cdot K}.\end{aligned}$$

- Mit der Abschätzung für N ergibt sich

$$\overline{P}^{(2)} \le 1-\left(1-t\cdot\frac{K}{2^n}\right)\cdot\frac{2^n}{2^n-K-t+1} < t\cdot\frac{K}{2^n}.$$

- Wählt man nun wie bei der Abschätzung des Fehlers 1. Art $r = \lfloor n\cdot p + \sqrt{\frac{2}{\epsilon}}\cdot\sigma_{n,p}\rfloor$, so hat man mit $\rho = \frac{r}{n} \sim p$ und

$$K = 2^{n\cdot R}, t \le 2^{n\cdot H(\rho)}$$

also

$$\overline{P}^{(2)} \le 2^{n\cdot(R+H(\rho)-1)} = 2^{n\cdot(R-C(\rho))}$$

wobei

$$C(\rho) = 1-H(\rho) = 1+\rho\cdot\log\rho+(1-\rho)\cdot\log(1-\rho)$$

die *Kapazität* des binären symmetrischen Kanals mit Fehlerwahrscheinlichkeit ρ ist.

- Für $R < C(\rho)$ gilt $2^{n(R-C(\rho))} \to_{n\to\infty} 0$
- Bei Vorgabe eines $\varepsilon > 0$ kann man also $\overline{P}^{(2)} < \varepsilon/2$ erreichen, indem man n genügend gross macht.
- Das bedeutet: betrachtet man Codes $\mathcal{C}$ mit Rate $R(\mathcal{C}) < C(p)$ so ist für hinreichend grosses n im Mittel
  – d.h. bei gleichverteilter zufälliger Auswahl der Codes und der gesendeten Codevektoren –
  die W.keit $\overline{P}^{(2)}$ für das Auftreten von Fehlern 2. Art $< \varepsilon/2$.

- Insgesamt wird für den BSC_p für Coderaten $\lesssim C(p)$ für hinreichend grosse Länge n die Fehlerwahrscheinlichkeit im Mittel $< \varepsilon$ werden.
- Es muss also auch (lange) Codes mit Rate $\lesssim C(p)$ geben, bei denen die Fehlerwahrscheinlichkeit $< \varepsilon$ ist!
- Dies ist eine nicht-konstruktive Existenzaussage!
- SHANNONs Theorem in Prosa:
  Fehlerkorrigierende Informationsübertragung über gestörte Kanäle ist für Coderaten, die (beliebig wenig) unterhalb der Kanalkapazität liegen, mit beliebig kleinen Fehlerwahrscheinlichkeiten prinzipiell möglich.

Shannons Theorem zur Kanalcodierung
Fazit

- *With many profound scientific discoveries it is possible with the aid of hindsight to see that the times were ripe for the breakthrough. Not so with information theory!*
  While of course Shannon was not working in a vacuum in the 1940's, his results were so breathtakingly original that even the communication specialists of the day were at loss to understand their significance.
  R. J. MCELIECE in *The Theory of Information and Coding*, Encyclopedia of Mathematics and Its Applications, vol. 3, Addison-Wesley, 1977.

Arithmetik

- Grundlegende Tatsachen über den Ring $\mathbb{Z}$
- Euklidischer Algorithmus
- Lösung ganzzahliger Gleichungssysteme
- Binärer euklidischer Algorithmus

Grundlegende Tatsachen über den Ring $\mathbb{Z}$

- $\langle \mathbb{Z}; +, \cdot \rangle$ ist ein nullteilerfreier Ring
- Divisionseigenschaft

$$\forall a \in \mathbb{Z}, b \in \mathbb{Z}_{>0}\ \exists q, r \in \mathbb{Z}\ :\ a = b \cdot q + r,\ 0 \leq r < b$$

- Bezeichnungen

$$q = \lfloor \frac{a}{b} \rfloor = (a \,\mathtt{div}\, b) \qquad : \text{Quotient}$$
$$r = a - \lfloor \frac{a}{b} \rfloor \cdot b = (a \,\mathtt{mod}\, b) \qquad : \text{Rest}$$

- Teilbarkeit

$$b \text{ teilt } a\ :\ b|a\ \Leftrightarrow\ (a \,\mathtt{mod}\, |b|) = 0$$

- **grösster gemeinsamer Teiler** für $(a, b) \in \mathbb{Z} \times \mathbb{Z}, (a, b) \neq (0, 0)$ ist $d \in \mathbb{Z}_{>0}$ mit

$$d|a \wedge d|b \wedge (\ \forall c \in \mathbb{Z}_{>0} : c|a \wedge c|b \Rightarrow c|d\ )$$

  Bezeichnung: $d = \mathtt{ggT}(a, b)$
  Tatsache: $d = \mathtt{ggT}(a, b) = \max\{c \in \mathbb{Z}_{>0}\,;\, c|a \wedge c|b\}$
  Konvention (GA): $\mathtt{ggT}(0, 0) = 0$

- **kleinstes gemeinsames Vielfaches** für $(a, b) \in \mathbb{Z} \times \mathbb{Z}, (a, b) \neq (0, 0)$ ist $m \in \mathbb{Z}_{>0}$ mit

$$a|m \wedge b|m \wedge (\ \forall c \in \mathbb{Z}_{>0} : a|c \wedge b|c \Rightarrow m|c\ )$$

  Bezeichnung: $m = \mathtt{kgV}(a, b)$
  Tatsache: $m = \mathtt{kgV}(a, b) = \min\{c \in \mathbb{Z}_{>0}\,;\, a|c \wedge b|c\}$

- $p \in \mathbb{Z}_{>1}$ ist **Primzahl**, wenn gilt

$$\forall a \in [1..p] : (a|p\ \Rightarrow\ a = 1 \vee a = p)$$

- **Euklid**:

$$p \text{ Primzahl}\ \Rightarrow\ (\ \forall a, b \in \mathbb{Z}\ :\ p|(a \cdot b) \Rightarrow p|a \vee p|b\ )$$

- **Euklid**: es gibt unendlich viele Primzahlen
- **Fundamentalsatz der Arithmetik** (Gauss, GA Theorem 7.21)

  Die Zerlegung natürlicher Zahlen in ihre Primteiler (mit Vielfachheiten) ist eindeutig, d.h.
  zu jeder natürlichen Zahl $n > 1$ gibt es eindeutig bestimmte Primzahlen $p_1 < p_2 < \ldots < p_k$ und Exponenten $e_1, e_2, \ldots, e_k \in \mathbb{N}_{>0}$ mit

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$$

- Die Berechnung dieser Darstellung (Faktorisierung) ist mutmasslich ein algorithmisch sehr aufwendiges Problem.

- Sind $a, b \in \mathbb{Z}_{>0}$ mit

$$a = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k},\ b = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdots p_k^{\beta_k}$$

mit Primzahlen $p_1 < p_2 < \ldots < p_k$ und Exponenten $\alpha_1, \alpha_2, \ldots, \alpha_k, \beta_1, \beta_2, \ldots, \beta_k \geq 0$, so gilt

$$\texttt{ggT}(a,b) = p_1^{\min\{\alpha_1,\beta_1\}} p_2^{\min\{\alpha_2,\beta_2\}} \cdots p_k^{\min\{\alpha_k,\beta_k\}}$$
$$\texttt{kgV}(a,b) = p_1^{\max\{\alpha_1,\beta_1\}} p_2^{\max\{\alpha_2,\beta_2\}} \cdots p_k^{\max\{\alpha_k,\beta_k\}}$$

ggT und kgV ganzer Zahlen lassen sich also prinzipiell mittels Faktorisierung berechnen — das ist aber keine effiziente Methode!

- $\texttt{ggT}(a,b) \cdot \texttt{kgV}(a,b) = a \cdot b$
- Ganze Zahlen a, b heissen **teilerfremd (relativ prim)**, wenn $\texttt{ggT}(a,b) = 1$.

Euklidischer Algorithmus — Grundalgorithmus

Von EUKLID stammt ein effizientes Verfahren zur ggT-Berechnungs mittels iterierter Division mit Rest:

rekursive Version:

Euclid-rek (int a,int b)
  if $b = 0$ **then**
    return(a)
  else
    return(Euclid-rek(b, a mod b))
  end if

iterative Version:

Euclid-iter (int a,int b)
  $\alpha := a, \beta := b$
  while $\beta \neq 0$ **do**
    $(\alpha, \beta) := (\beta, \alpha \bmod \beta)$
  end while
  return(α)

basierend auf

Theorem (GA, Thm. 7.19)

$$\forall a, b \in \mathbb{Z} : \texttt{ggT}(a,b) = \texttt{ggT}(b, a \bmod b)$$

Schema der Ausführung von Euklids Algorithmus

$$\begin{array}{rcll}
a_0 & := & a & \\
a_1 & := & b & \\
a_0 & = & a_1 \cdot q_0 + a_2 & (q_0 \in \mathbb{Z}, 0 < a_2 < a_1) \\
a_1 & = & a_2 \cdot q_1 + a_3 & (q_1 \in \mathbb{N}_+, 0 < a_3 < a_2) \\
a_2 & = & a_3 \cdot q_2 + a_4 & (q_2 \in \mathbb{N}_+, 0 < a_4 < a_3) \\
 & \vdots & & \\
a_{n-2} & = & a_{n-1} \cdot q_{n-2} + a_n & (q_{n-2} \in \mathbb{N}_+, 0 < a_n < a_{n-1}) \\
a_{n-1} & = & a_n \cdot q_{n-1} & (q_{n-1} \geq 2, a_{n+1} = 0)
\end{array}$$

$\langle q_0, q_1, q_2, \ldots, q_{n-1} \rangle$: *Quotientenfolge*
$\langle a_0, a_1, a_2, \ldots, a_n, 0 \rangle$: *Restefolge*

Beispiel: $a = 57, b = 33$

$$\begin{aligned}
57 &= 33 \cdot 1 + 24 \\
33 &= 24 \cdot 1 + 9 \\
24 &= 9 \cdot 2 + 6 \\
9 &= 6 \cdot 1 + 3 \\
6 &= 3 \cdot 2
\end{aligned}$$

Quotientenfolge: $\langle q_0, q_1, \ldots, q_4 \rangle = \langle 1, 1, 2, 1, 2 \rangle$
Restefolge: $\langle a_0, a_1, a_2, \ldots, a_6 \rangle = \langle 57, 33, 24, 9, 6, 3, 0 \rangle$

- **Terminierung**

$$a_0 \geq a_1 > a_2 > \cdots > a_n > a_{n+1} = 0 \text{ für ein } n \geq 1$$

- **Korrektheit**

$$\begin{aligned} \texttt{ggT}(a,b) &= \texttt{ggT}(a_0, a_1) \\ &= \texttt{ggT}(a_1, a_2) \\ &= \texttt{ggT}(a_2, a_3) \\ &= \cdots = \texttt{ggT}(a_n, a_{n+1}) = \texttt{ggT}(a_n, 0) = a_n \end{aligned}$$

Effizienz wird gesichert durch den **Satz von Lamé** (1845):

Wenn die Berechnung von $\texttt{ggT}(a,b)$ *für* $a > b > 0$ *genau* k *Aufrufe der Prozedur* `Euclid-rek` *(bzw.* k *Schleifendurchläufe der while-Schleife in* `Euclid-iter`*) erfordert, dann gilt* $a \geq f_k$ *und* $b \geq f_{k-1}$*, wobei* f_i *die* i*-te* FIBONACCI*-Zahl ist.*

Aus den bekannten Aussagen über die FIBONACCI-Zahlen ergibt sich:

Sind $a, b \in \mathbb{N}$ *mit* $a \geq b$*, dann ist die Zahl der Divisionschritte im Euklidischen Algorithmus für* $a, b \leq 4.8 \cdot \log_{10}(a) + 2$.

Eine alternative Betrachtung:

Euklids Algorithmus für $(a,b) = (a_0, a_1)$ erzeugt zu der Folge

$$a_i = a_{i+1} \cdot q_i + a_{i+2} \quad (0 \leq i \leq n)$$

von Divisionsschritten eine Quotientenfolge $\langle q_0, q_1, \ldots, q_n \rangle$ mit

$$q_i \geq 1 \;\; (0 \leq i < n) \quad \text{und} \quad q_n \geq 2$$

Es gilt

$$a_{i+2} \cdot (q_i + 1) < a_{i+1} \cdot q_i + a_{i+2} = a_i \;\; (0 \leq i < n)$$

und somit

$$q_i + 1 < a_i / a_{i+2} \;\; (0 \leq i < n) \;\text{ und }\; q_n = a_n / a_{n+1} \; .$$

Ausgehend von $a = a_0 \geq a_1 = b$ ergibt sich

$$\begin{aligned} 2^{n+1} &\leq q_n \prod_{i=0}^{n-1} (q_i + 1) < \frac{a_n}{a_{n+1}} \prod_{i=0}^{n-1} \frac{a_i}{a_{i+2}} \leq \frac{a_0 \cdots a_n}{a_2 \cdots a_{n+1}\, a_{n+1}} \\ &= \frac{a_0\, a_1}{a_{n+1}\, a_{n+1}} \leq \left(\frac{a}{\texttt{ggT}(a,b)} \right)^2 \end{aligned}$$

- **Theorem** : *Für* $a, b \in \mathbb{N}$ *benötigt die Berechnung von* `ggT(a,b)` *höchstens*

$$\lfloor 2 \cdot \mathit{log}_2 \mathit{max}(a,b) \rfloor + 1 \;\; \textit{Divisionsschritte}$$

(Linearität in der Problemgrösse)

Untersuchung im logarithmischen Komplexitätsmodell:

$\ell_\beta(a)$: Grösse (Länge) von a bei der Darstellungen in Basis β

Jeder Divisionschritt $a_i = a_{i+1} \cdot q_i + a_{i+2}$ erfordert $\ell_\beta(a_{i+1}) \cdot \ell_\beta(q_i)$ Operationen in β-Arithmetik.

Gesamtaufwand, gemessen in Basis-β-Operationen für Euklids Algorithmus (a, b):

$$\sum_{i=0}^{n} \ell_\beta(a_{i+1}) \cdot \ell_\beta(q_i)$$

und dies ist, unter der Annahme $a \geq b$, beschränkt durch

$$\begin{aligned} \ell_\beta(a_1) \cdot \left(\sum_{i=0}^{n-1} \ell_\beta(q_i + 1) + \ell_\beta(q_n) \right) &\sim \ell_\beta(b) \cdot \ell_\beta(q_n \cdot \prod_{i=0}^{n-1} (q_i + 1)) \\ &\leq 2\,\ell_\beta(b) \cdot (\ell_\beta(a) - \ell_\beta(\mathsf{ggT}(a, b)) + 1) \end{aligned}$$

Euklidischer Algorithmus — erweiterte Form

Eine fundamental wichtige Eigenschaft des ggT für ganze Zahlen ist:

> *der* $\mathsf{ggT}(a, b)$ *lässt sich als Linearkombination von* a *und* b *mit ganzzahligen Koeffizienten darstellen (*BÉZOUT*-Beziehung)*
>
> $$\forall a, b \in \mathbb{Z}\ \exists s, t \in \mathbb{Z} : a \cdot s + b \cdot t = \mathsf{ggT}(a, b)$$

Tatsächlich ist $\mathsf{ggT}(a, b)$ die kleinste positive Zahl $\in \mathbb{Z}$, die sich als Linearkombination von a und b darstellen lässt.

Die algebraische Sicht:

- Für $a, b \in \mathbb{Z}$ ist
  $$H_{a,b} = \{\, a \cdot u + b \cdot v\,;\, u, v \in \mathbb{Z} \,\}$$
  eine Untergruppe (sogar ein "Ideal") von $\mathbb{Z}$,
  d.h. $x, y \in H_{a,b} \Rightarrow x - y \in H_{a,b}$ (und $u \cdot x \in H_{a,b}$ für alle $u \in \mathbb{Z}$)
- Divisionseigenschaft von $\mathbb{Z} \Rightarrow$ ist $H \neq \{0\}$ eine Untergruppe von $\mathbb{Z}$, so gilt
  $$H = k \cdot \mathbb{Z} = \{k \cdot u\,;\, u \in \mathbb{Z}\,\}$$
  wobei $k = \min\{h \in H\,;\, h > 0\}$
- Für $a, b \in \mathbb{Z}$ mit $(a, b) \neq (0, 0)$ gilt $H_{a,b} = k \cdot \mathbb{Z}$ mit $k = a \cdot s + b \cdot t$ und es ist $k = \mathsf{ggT}(a, b)$, denn
  - $a \in H_{a,b} \Rightarrow k \,|\, a$ und $b \in H_{a,b} \Rightarrow k \,|\, b$
  - $c | a \wedge c \,|\, b \Rightarrow c \,|\, (a \cdot s + b \cdot t) = k$
- Das liefert noch kein algorithmisches Verfahren zur Berechnung von (s, t)


```
EXTENDED_EUCLID (int a, int b)
{   /* returns (d, s, t), where d = gcd(a, b) = as + bt */
    if (b == 0) return (a, 1, 0);
    else
    {
        (d', s', t') = Extended_Euclid(b, a mod b);
        s = t';
        t = s' − t' · (a div b);
        return (d', s, t);
    }
}
```

Iterative Version des erweiterten Euklidischen Algorithmus:

seien $a \geq b > 0$
für $i = 0, 1, 2, \ldots$ seien $\boldsymbol{\alpha}^{(i)} = \langle s_i, t_i, a_i \rangle \in \mathbb{Z}^3$ und $q_i \in \mathbb{N}_+$ definiert durch

$\boldsymbol{\alpha}^{(0)} := \langle 1, 0, a \rangle$
$\boldsymbol{\alpha}^{(1)} := \langle 0, 1, b \rangle$
$i := 1$
while $a_i \neq 0$ **do**
  $q_{i-1} := \lfloor a_{i-1}/a_i \rfloor$
  $\boldsymbol{\alpha}^{(i+1)} := \boldsymbol{\alpha}^{(i-1)} - q_{i-1} \cdot \boldsymbol{\alpha}^{(i)}$
  $i := i + 1$
end while

Beispiel $a = 57, b = 33$

$$\begin{aligned}
\boldsymbol{\alpha}^{(0)} &= \langle 1, 0, 57 \rangle \\
\boldsymbol{\alpha}^{(1)} &= \langle 0, 1, 33 \rangle & q_0 &= \lfloor 57/33 \rfloor = 1 \\
\boldsymbol{\alpha}^{(2)} &= \boldsymbol{\alpha}^{(0)} - q_0 \cdot \boldsymbol{\alpha}^{(1)} = \langle 1, -1, 24 \rangle & q_1 &= \lfloor 33/24 \rfloor = 1 \\
\boldsymbol{\alpha}^{(3)} &= \boldsymbol{\alpha}^{(1)} - q_1 \cdot \boldsymbol{\alpha}^{(2)} = \langle -1, 2, 9 \rangle & q_2 &= \lfloor 24/9 \rfloor = 2 \\
\boldsymbol{\alpha}^{(4)} &= \boldsymbol{\alpha}^{(2)} - q_2 \cdot \boldsymbol{\alpha}^{(3)} = \langle 3, -5, 6 \rangle & q_3 &= \lfloor 9/6 \rfloor = 1 \\
\boldsymbol{\alpha}^{(5)} &= \boldsymbol{\alpha}^{(3)} - q_3 \cdot \boldsymbol{\alpha}^{(4)} = \langle -4, 7, 3 \rangle & q_4 &= \lfloor 6/3 \rfloor = 2 \\
\boldsymbol{\alpha}^{(6)} &= \boldsymbol{\alpha}^{(4)} - q_4 \cdot \boldsymbol{\alpha}^{(5)} = \langle 11, -19, 0 \rangle
\end{aligned}$$

Bézout-Beziehung

$$57 \cdot (-4) + 33 \cdot 7 = 3 = \mathtt{ggT}(57, 33)$$

Notabene

$$57 \cdot 11 - 33 \cdot (-19) = 0,\ 11 = \frac{33}{\mathtt{ggT}(57, 33)},\ 19 = \frac{57}{\mathtt{ggT}(57, 33)}$$

Die Eigenschaften des erweiterten Euklidischen Algorithmus ergeben sich leicht aus einer Darstellung in Matrizenschreibweise

$\boldsymbol{\alpha} = \langle \alpha_1, \alpha_2, \alpha_3 \rangle := \langle 1, 0, a \rangle$
$\boldsymbol{\beta} = \langle \beta_1, \beta_2, \beta_3 \rangle := \langle 0, 1, b \rangle$
while $\beta_3 \neq 0$ **do**

$$q := \alpha_3 \ \mathtt{div}\ \beta_3$$

$$\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} := \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix}$$

end while
$\langle s, t, d \rangle := \langle \alpha_1, \alpha_2, \alpha_3 \rangle$

Es ist

$$\begin{pmatrix} \boldsymbol{\alpha}^{(i)} \\ \boldsymbol{\alpha}^{(i+1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_{i-1} \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_0 \end{pmatrix} \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \end{pmatrix}$$

für $0 \leq i \leq n$;

daher ist

$$\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} \begin{pmatrix} a \\ b \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

eine Schleifeninvariante.

Mit $\boldsymbol{\alpha}^{(i)} = \langle s_i, t_i, a_i \rangle$ ist

$$a \cdot s_i + b \cdot t_i = a_i \quad \text{für } 0 \leq i \leq n + 1$$

Daraus ergibt sich die BÉZOUT-Beziehung:

$$a \cdot s_n + b \cdot t_n = a_n = \mathtt{ggT}(a,b)$$

sowie

$$a \cdot s_{n+1} + b \cdot t_{n+1} = a_{n+1} = 0$$

mit

$$s_{n+1} = (-1)^{n+1} \frac{b}{\mathtt{ggt}(a,b)} \quad \text{und} \quad t_{n+1} = (-1)^n \frac{a}{\mathtt{ggt}(a,b)}$$

Beweis (Induktion über Anzahl der Divisionen im EA)
beachte: ist $\langle q_0, q_1, \ldots, q_{n-1}\rangle$ die Quotientenfolge für EA(a_0, a_1), so ist $\langle q_1, q_2, \ldots, q_{n-1}\rangle$ die Quotientenfolge für EA(a_1, a_2), wobei $\mathtt{ggt}(a_0, a_1) = \mathtt{ggt}(a_1, a_2)$, daher

$$\begin{aligned}
\begin{pmatrix} s_n & t_n \\ s_{n+1} & t_{n+1} \end{pmatrix} &= \begin{pmatrix} 0 & 1 \\ 1 & -q_n \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_0 \end{pmatrix} \\
&= \begin{pmatrix} s'_{n-1} & t'_{n-1} \\ s'_n & t'_n \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_0 \end{pmatrix} = \begin{pmatrix} s'_{n-1} & t'_{n-1} \\ \frac{(-1)^n a_2}{\mathtt{ggt}(a_1,a_2)} & \frac{(-1)^{n+1} a_1}{\mathtt{ggt}(a_1,a_2)} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_0 \end{pmatrix} \\
&= \begin{pmatrix} t'_{n-1} & s'_{n-1} - q_0 t'_{n-1} \\ \frac{(-1)^{n+1} a_1}{\mathtt{ggt}(a_0,a_1)} & \frac{(-1)^n a_2}{\mathtt{ggt}(a_0,a_1)} - q_0 \frac{(-1)^{n+1} a_1}{\mathtt{ggt}(a_0,a_1)} \end{pmatrix} \\
&= \begin{pmatrix} t'_{n-1} & s'_{n-1} - q_0 t'_{n-1} \\ \frac{(-1)^{n+1} a_1}{\mathtt{ggt}(a_0,a_1)} & \frac{(-1)^n a_0}{\mathtt{ggt}(a_0,a_1)} \end{pmatrix}
\end{aligned}$$

Beispiel $a = 57, b = 33$:

Quotientenfolge : $\langle q_0, \ldots, q_4\rangle = \langle 1, 1, 2, 1, 2\rangle$

$$\begin{aligned}
\begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} &= \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix} \\
\begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} &= \begin{pmatrix} -1 & 2 \\ 3 & -5 \end{pmatrix} \\
\begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} &= \begin{pmatrix} 3 & -5 \\ -4 & 7 \end{pmatrix} \\
\begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} &= \begin{pmatrix} -4 & 7 \\ 11 & -19 \end{pmatrix}
\end{aligned}$$

EUKLIDs Algorithmus (in der erweiterten Form von BÉZOUT) behandelt ein fundamentales Problem:

Lösen linearer Gleichungen in $\mathbb{Z}$

denn es gilt für $a, b, c \in \mathbb{Z}$ mit $(a, b) \neq (0, 0)$:

$$\exists\, u, v \in \mathbb{Z} \;:\; a \cdot u + b \cdot v = c \quad \Leftrightarrow \quad \mathtt{ggT}(a,b) \,|\, c$$

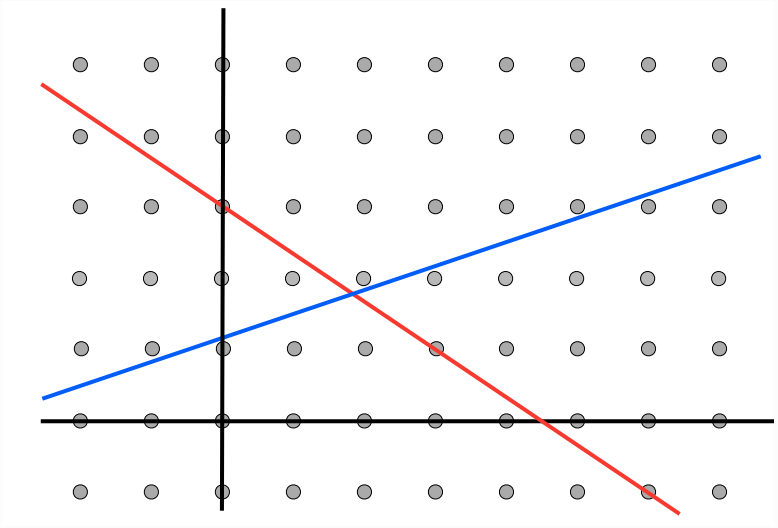
und wenn diese Bedingung erfüllt ist, ist die Menge der ganzzahligen Lösungen der Geradengleichung

$$a \cdot x + b \cdot y = c$$

gegeben durch

$$(x, y) = \frac{c}{d} \cdot (s, t) + \frac{k}{d} \cdot (-b, a) \quad (k \in \mathbb{Z})$$

wobei s, t Bézout-Koeffizienten für (a, b) und $d = \mathtt{ggT}(a, b)$ sind, d.h., $a \cdot s + b \cdot t = d$



Zum Beweis:

- es gilt

$$H_{a,b} = \{\, a \cdot u + b \cdot v \,;\, u, v \in \mathbb{Z} \,\} = \mathrm{ggT}(a,b) \cdot \mathbb{Z}$$

  daher ist $a \cdot x + b \cdot y = c$ in $\mathbb{Z}$ lösbar $\Leftrightarrow c \in H_{a,b}$ (d.h. $\mathrm{ggT}(a,b) \mid c$)

- falls Lösbarkeit gegeben und $d = \mathrm{ggT}(a,b)$

$$a \cdot x + b \cdot y = c \Leftrightarrow \frac{a}{d} \cdot x + \frac{b}{d} \cdot y = \frac{c}{d}$$

  wobei nun $\mathrm{ggT}(a/d, b/d) = 1$, d.h. es genügt, den Fall $\mathrm{ggT}(a,b) = 1$ zu betrachten

- mit Bézout-Koeffizienten s, t, d.h. $a \cdot s + b \cdot t = 1$ gilt

$$a \cdot (c \cdot s) + b \cdot (c \cdot t) = c$$

  d.h. $(x_0, y_0) = (c \cdot s, c \cdot t)$ ist spezielle Lösung von $a \cdot x + b \cdot y = c$

- $(x, y) = (x_0 + X, y_0 + Y)$ ist genau dann Lösung von $a \cdot x + b \cdot y = c$, wenn (X, Y) Lösung von $a \cdot X + b \cdot Y = 0$

- die Lösungen von $a \cdot X + b \cdot Y = 0$ sind genau die $(k \cdot b, -k \cdot a)$ mit $k \in \mathbb{Z}$ — hierfür benötigt man $\mathrm{ggT}(a,b) = 1$

Beispiel

- $57 \cdot x + 33 \cdot y = -9$

  Lösungsmenge:

$$\begin{aligned}(x, y) &= \frac{-9}{3} \cdot (-4, 7) + \frac{k}{3} \cdot (-33, 57) \\ &= (12 - 11 \cdot k, -21 + 19 \cdot k) \quad (k \in \mathbb{Z})\end{aligned}$$

- $57 \cdot x + 33 \cdot y = -8$

  Lösungsmenge: $\emptyset$

Wichtige Bemerkung:
man kann diese Idee ausbauen zu einem Verfahren, um die Menge der ganzzahligen Lösungen von linearen Gleichungssysteme mit ganzzahligen Koeffizienten zu berechnen.

(1)	$57x + 33y$	$=$	-9	die zu lösende Gleichung
(2)	$x + y$	$=$	z	Division mit $33 = \min\{57, 33\}$, neue Variable z
(3)	$24x + 33z$	$=$	-9	$(1) - 33 \cdot (2)$
(4)	$x + z$	$=$	u	Division mit $24 = \min\{24, 33\}$, neue Variable u
(5)	$24u + 9z$	$=$	-9	$(3) - 24 \cdot (4)$
(6)	$2u + z$	$=$	v	Division mit $9 = \min\{24, 9\}$, neue Variable v
(7)	$6u + 9v$	$=$	-9	$(5) - 9 \cdot (6)$
(8)	$u + v$	$=$	w	Division mit $6 = \min\{6, 9\}$, neue Variable w
(9)	$6w + 3v$	$=$	-9	$(7) - 6 \cdot (8)$
(10)	$2w + v$	$=$	-3	Division mit $\min\{3, 6\} = 3$
(11)	v	$=$	$-3 - 2w$	aus (10)
(12)	u	$=$	$3w + 3$	aus (11) und (8)
(13)	z	$=$	$-9 - 8w$	aus (12) und (6)
(14)	x	$=$	$11w + 12$	aus (13) und (4)
(15)	y	$=$	$-21 - 19w$	aus (14) und (2)

Die ganzzahlige Lösungsmenge der Gleichung $57x + 33y = -9$ ist also

$$(x, y) = (12, -21) + w \cdot (11, -19) \quad \text{mit } w \in \mathbb{Z}$$

Das Verfahren funktioniert auch für Gleichungen mit mehreren Variablen

(1)	$8x - 7y - 5z$	$=$	2	zu lösende Gleichung
(2)	$x - y - z$	$=$	u	Division mit 5, neue Variable u
(3)	$3x - 2y + 5u$	$=$	2	$(1) - 5 \cdot (2)$
(4)	$x - y + 2u$	$=$	v	Division mit 2, neue Variable v
(5)	$x + 2v + u$	$=$	2	$(3) - 2 \cdot (4)$
(6)	x	$=$	$2 - u - 2v$	aus (5)
(7)	y	$=$	$2 + u - 3v$	aus (6) und (4)
(8)	z	$=$	$-3u + v$	aus (7) und (2)

Die ganzzahlige Lösungsmenge der Gleichung $8x - 7y - 5z = 2$ ist also

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1 & -2 \\ 1 & -3 \\ -3 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} \quad u, v \in \mathbb{Z}$$

Zur Lösung linearer Gleichungen über $\mathbb{Z}$

Gesucht ganzzahlige Lösungsmenge von

$$(*) \quad a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_n \cdot x_n = b$$

wobei $a_1, \cdots, a_n, b \in \mathbb{Z}$, nicht alle $a_i = 0$

- ohne Einschränkung sei $a_1 > 0$ und $a_1 = \min_{1 \le i \le n}\{|a_i| \,;\, a_i \neq 0\}$
- Fall 1: $a_1 = 1$
  dann ist die allgemeine Lösung von (*) gegeben durch
  $$(b - a_2 \cdot x_2 - \cdots - a_n \cdot x_n, x_2, \ldots, x_n) \quad \text{mit} \quad (x_2, \ldots, x_n) \in \mathbb{Z}^{n-1}$$
- Fall 2a: $a_1 > 1$, alle a_i durch a_1 teilbar
  dann ist (*) lösbar $\Leftrightarrow a_1 \mid b$ und Lösung ergibt sich aus
  $$\frac{a_1}{a_1} \cdot x_1 + \frac{a_2}{a_1} \cdot x_2 + \cdots \frac{a_n}{a_1} \cdot x_n = \frac{b}{a_1}$$
  $\rightarrow$ Fall 1.

– Fall 2a: $a_1 > 1$, nicht alle a_i durch a_1 teilbar

Bezeichnungen ($\div$ = ganzzahlige Division)

$$\begin{aligned}
\boldsymbol{a} &= \langle a_1, a_2, \ldots, a_n \rangle \\
\boldsymbol{x} &= \langle x_1, x_2, \ldots, x_n \rangle \\
(\boldsymbol{a} \div a_1) &= \langle \underbrace{a_1 \div a_1}_{=1}, a_2 \div a_1, \ldots, a_n \div a_1 \rangle \\
(\boldsymbol{a} \bmod a_1) &= \langle \underbrace{a_1 \bmod a_1}_{=0}, \underbrace{a_2 \bmod a_1, \ldots, a_n \bmod a_1}_{\text{nicht alle } =0} \rangle
\end{aligned}$$

Zu lösende Gleichung ($\bullet$ = Skalarprodukt)

$$(*) \qquad \boldsymbol{a} \bullet \boldsymbol{x} = b$$

Divisionsbeziehung

$$\boldsymbol{a} = a_1 \cdot (\boldsymbol{a} \div a_1) + (\boldsymbol{a} \bmod a_1)$$

Daher

$$\boldsymbol{a} \bullet \boldsymbol{x} = a_1 \cdot \underbrace{(\boldsymbol{a} \div a_1) \bullet \boldsymbol{x}}_{y} + (\boldsymbol{a} \bmod a_1) \bullet \boldsymbol{x}$$

Betrachte y als neue Variable mit

$$(**) \quad y = x_1 + \lfloor \frac{a_2}{a_1} \rfloor \cdot x_2 + \cdots + \lfloor \frac{a_n}{a_1} \rfloor \cdot x_n$$

Mit

$$\begin{aligned}
\widetilde{\boldsymbol{a}} &= \langle a_1, a_2 \bmod a_1, \ldots, a_n \bmod a_1 \rangle \\
\widetilde{\boldsymbol{x}} &= \langle y, x_2, \ldots, x_n \rangle
\end{aligned}$$

gilt: die Lösungen $\langle x_1, x_2, \ldots, x_n \rangle$ von $(*)$ entsprechen ein-eindeutig den Lösungen $\langle y, x_2, \ldots, x_n \rangle$ von

$$\widetilde{\boldsymbol{a}} \bullet \widetilde{\boldsymbol{x}} = b$$

vermöge $(**)$.

Wichtig (für die Terminierung):

$$\min\{a_1, a_2 \bmod a_1, \ldots, a_n \bmod a_1\} < a_1$$

Das Verfahren lässt sich für die Bestimmung der ganzzahligen Lösungen ganzzahliger Gleichungssysteme verwenden.

Beispiel (vgl. KNUTH, TAOCP, Abschnitt 4.5.2)

1. zu lösendes System

$$\begin{array}{rcrcrcrcl}
10w & + & 3x & + & 3y & + & 8z & = & 1 \\
6w & - & 7x & & & - & 5z & = & 2
\end{array}$$

2. Elimination von y mittels der ersten Gleichung und Einführung einer neuen Variablen t_1

$$\begin{array}{rcrcrcrcl}
3w & + & x & + & y & + & 2z & = & t_1 \\
w & & & + & 3t_1 & + & 2z & = & 1
\end{array}$$

3. Elimination von w aus zweiter Gleichung

$$\begin{array}{rcrcrcrcl}
6(1-3t_1-2z) & - & 7x & - & 5z & = & 2 \\
18t_1 & + & 7x & + & 17z & = & 4
\end{array}$$

4. Elimination von x durch Einführung einer neuen Variablen t_2

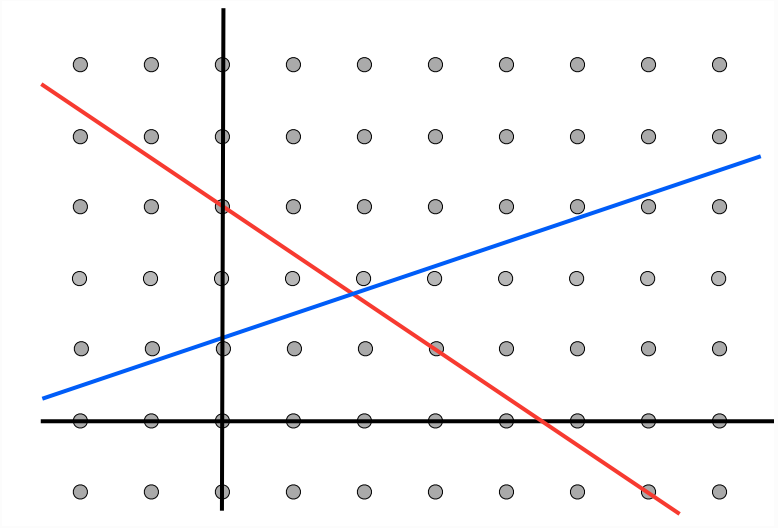
$$\begin{array}{rcrcrcrcl}
2t_1 & + & x & + & 2z & = & t_2 \\
4t_1 & + & 7t_2 & + & 3z & = & 4
\end{array}$$

5. Elimination von z durch Einführung einer neuen Variablen t_3

$$\begin{array}{rcrcrcrcl}
t_1 & + & 2t_2 & + & z & = & t_3 \\
t_1 & + & t_2 & + & 3t_3 & = & 4
\end{array}$$

6. Auflösen nach t_2 und Rückwaärtseinsetzen liefert

$$\begin{array}{rcrcrcrcl}
w & = & 17 & - & 5t_1 & - & 14t_3 \\
x & = & 20 & - & 5t_1 & - & 17t_3 \\
y & = & -55 & + & 19t_1 & + & 45t_3 \\
z & = & -8 & + & t_1 & + & 7t_3
\end{array}$$



Der binäre Euklidische Algorithmus

macht sich die Binärdarstellung der Zahlen zu Nutze und realisiert die Berechnung des ggT mittels Subtraktion und Division durch Zweierpotenzen (= shift).
Er benützt die (offensichtlichen) Beziehungen

$$\begin{aligned} \mathsf{ggT}(2a, 2b) &= 2 \cdot \mathsf{ggT}(a, b) && (a, b \in \mathbb{N}) \\ \mathsf{ggT}(2a, b) &= \mathsf{ggT}(a, b) && (a, b \in \mathbb{N}, b \text{ ungerade}) \\ \mathsf{ggT}(a, b) &= \mathsf{ggT}(a - b, b) && (a, b \in \mathbb{N}, a \geq b, \text{beide ungerade}) \end{aligned}$$


```
BINARY_EUCLID (int a, int b)
{ /* returns gcd(a,b) provided that a,b ∈ ℕ */
    int d = 1;
    while ((a mod 2 == 0) && (b mod 2 == 0))
    {
        a = a/2; b = b/2; d = 2 · d;
    }
    while (a ≠ 0)
    {
    /* a or b is odd */
        while (a mod 2 == 0) a = a/2;
        while (b mod 2 == 0) b = b/2;
        /* a and b are odd */
        if (a < b) swap(a, b)
        a = a − b;
    }
    return (b · d);
}
```


Beispiel: $a = 40902, b = 24140$

a	b	$\rightarrow$	a	b	
40902	24140	$\rightarrow$	20451	12770	$\Rightarrow d = 2$
20451	12770	$\rightarrow$	20451	6035	
14416	6035	$\rightarrow$	901	6035	
5134	901	$\rightarrow$	2567	901	
1666	901	$\rightarrow$	833	901	
68	833	$\rightarrow$	17	833	
816	17	$\rightarrow$	51	17	
34	17	$\rightarrow$	17	17	
0	17	$\rightarrow$			

$\Rightarrow \mathsf{ggT}(40902, 24140) = 2 \cdot 17 = 34$

- Termination und Korrektheit von BINARY_EUCLID ergeben sich aus den drei verwendeten Gleichungen für den ggT
- Bei jedem Durchlauf der äusseren while-Schleife wird mindestens eines der beiden Argumente halbiert $\Rightarrow O(\log(a+b))$ Schleifendurchläufe
- Jeder Schleifendurchlauf erfordert Operationen (Vergleiche, Zuweisungen, Divisionen durch 2, Subtraktion), die sich insgesamt in $O(\log(a+b))$ Bit-Operationen ausführen lassen
- BINARY_EUCLID berechnet den ggT von natürlichen Zahlen mit $O(\log^2(a+b))$ Bit-Operationen
  (HEUN, *Grundlegende Algorithmen*, Abschnitt 7.1; siehe auch KNUTH, *The Art of Computer Programing*, vol. 2, *Seminumerical Algorihtms*, Abschnitt 4.5.2, und BACH/SHALLIT, *Algorithmic Number Theory*, vol.1 *Efficient Algorithms*, Kapitel 4)

BINARY_EXTENDED_EUCLID (int a, int b)

```
{ /* returns (d,s,t), where d = gcd(a,b) = a·s + b·t */
    (d,s,t) = (a,1,0);
    (d',s',t') = (b,0,1);
    while (d' ≠ 0)
    {
        if (d' < d) swap((d,s,t),(d',s',t'))
        c = 1 << max(ℓ(d') − ℓ(d) − 1, 0);   /*c = 2^max(ℓ(d')−ℓ(d)−1,0) */
        (d',s',t') = (d',s',t') − c(d,s,t);
    }
    return (d,s,t);
}
```


- die wesentliche Änderung gegenüber der iterativen Version der Euklidischen Algorithmus
  - an Stelle der exakten Quotienten $\lfloor d'/d \rfloor$ wird mit den angenäherten Quotienten
    $$c = 2^{\max(\ell(d')-\ell(d)-1,0)}$$
    gerechnet ($\rightarrow$ shift-Operationen). Dabei gilt
    $$\frac{1}{4} \cdot \lfloor d'/d \rfloor \leq 2^{\max(\ell(d')-\ell(d)-1,0)} \leq \lfloor d'/d \rfloor$$
- Konsequenz für die Bit-Komplexität: auch BINARY_EXTENDED_EUCLID verwendet nur $O(\log^2(a+b))$ Bit-Operationen.

a	1023	$\ell(a)=10$	1	1	1	1	1	1	1	1	1	1
b	15	$\ell(b)=4$	0	0	0	0	0	0	1	1	1	1
$x = b * 2^{\ell(a)-\ell(b)-1}$	480		0	1	1	1	1	0	0	0	0	0
$c = a - x$	543	$\ell(c)=10$	1	0	0	0	0	1	1	1	1	1
$y = b * 2^{\ell(c)-\ell(b)-1}$	480		0	1	1	1	1	0	0	0	0	0
$d = c - y$	63	$\ell(d)=6$	0	0	0	0	1	1	1	1	1	1
$z = b * 2^{\ell(d)-\ell(b)-1}$	30		0	0	0	0	0	1	1	1	1	0
$e = d - z$	33	$\ell(e)=6$	0	0	0	0	1	0	0	0	0	1
$u = b * 2^{\ell(e)-\ell(b)-1}$	30		0	0	0	0	0	1	1	1	1	0
$f = e - u$	3	$\ell(f)=2$	0	0	0	0	0	0	0	0	1	1
$v = f * 2^{\ell(b)-\ell(f)-1}$	6		0	0	0	0	0	0	0	1	1	0
$g = b - v$	9	$\ell(g)=4$	0	0	0	0	0	0	1	0	0	1
$w = f * 2^{\ell(g)-\ell(f)-1}$	6		0	0	0	0	0	0	0	1	1	0
$h = g - w$	3	$\ell(h)=2$	0	0	0	0	0	0	0	0	1	1

Die Ringe $\mathbb{Z}_n$

- für $n > 0$ wird auf $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ definiert:

$$+_n : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n : (a, b) \mapsto (a + b) \bmod n$$
$$*_n : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n : (a, b) \mapsto (a * b) \bmod n$$

  Beispiel $n = 15$

$$9 +_{15} 11 = 5 \qquad 9 *_{15} 11 = 9$$
$$9 *_{15} 5 = 0 \qquad 9 \wedge_{15} 11 = 9$$

- $\langle \mathbb{Z}_n; +_n, *_n \rangle$ ist ein Ring mit kommutativer Addition und Multiplikation
- $\langle \mathbb{Z}_n; +_n, *_n \rangle$ ist genau dann nullteilerfrei, wenn n eine Primzahl ist

Algebraische Bemerkung (äquivalente Definition)

Der Ring $\langle \mathbb{Z}_n; +_n, *_n \rangle$ entsteht aus dem Ring $\langle \mathbb{Z}; +, * \rangle$, indem man Restklassen bezüglich der Untergruppe (sogar: Ideal) $n\mathbb{Z} = \{n \cdot k \,;\, k \in \mathbb{Z}\}$ betrachtet:

- Die n verschiedenen Restklassen

$$[a]_n = a + n\mathbb{Z} = \{a + n \cdot k \,;\, k \in \mathbb{Z}\}$$

  bilden die Elemente des Ringes $\langle \mathbb{Z}/(n\mathbb{Z}); \oplus_n, \odot_n \rangle$ mit

$$[a]_n \oplus_n [b]_n = [a + b]_n \quad (a, b \in \mathbb{Z})$$
$$[a]_n \odot_n [b]_n = [a \cdot b]_n \quad (a, b \in \mathbb{Z})$$

- $[a]_n = [b]_n \;\Leftrightarrow\; a \equiv b \mod n \;\Leftrightarrow\; n|(a - b)$
- Die Ringe $\langle \mathbb{Z}/(n\mathbb{Z}); \oplus_n, \odot_n \rangle$ und $\langle \mathbb{Z}_n; +_n, *_n \rangle$ sind isomorph:

$$(\mathbb{Z}/n\mathbb{Z}) \ni [a]_n \iff a \mod n \in \mathbb{Z}_n$$

Invertierbare Elemente ("Einheiten")

- beachte: für $a, n \in \mathbb{Z}$ gilt (Bézout!!)

$$\mathrm{ggT}(a, n) = 1 \quad \Leftrightarrow \quad \exists b \exists k \in \mathbb{Z} \,:\, a * b + n * k = 1$$

- Dabei gilt

$$\mathrm{ggT}(a, n) = 1 \quad \Rightarrow \quad \mathrm{ggT}(b, n) = 1$$

- b ist bis auf Vielfache von n eindeutig bestimmt, ist also ein Element von $\mathbb{Z}_n$!
- Man berechnet b (bzw. $b \bmod n$) mit Hilfe des erweiterten euklidischen Algorithmus.

- für $a \in \mathbb{Z}_n$

$$\exists b \in \mathbb{Z}_n \,:\, a *_n b = 1 \quad \Leftrightarrow \quad \exists b \in \mathbb{Z} \exists k \in \mathbb{Z} \,:\, a * b + n * k = 1$$
$$\Leftrightarrow \quad \mathrm{ggT}(a, n) = 1$$

- dieses $b \in \mathbb{Z}_n$ ist das Inverse von $a \in \mathbb{Z}_n$ bezüglich der Multiplikation: $b = a^{-1}$
- Menge der invertierbaren Elemente von $\mathbb{Z}_n$:

$$\mathbb{Z}_n^* = \left\{a \in \{1, 2, \ldots, n-1\} \,;\, \mathrm{ggT}(a, n) = 1\right\}$$

- Beispiel:

  $EEA(11, 19) \;\Rightarrow\; 11 \cdot 7 + 19 * (-4) = 1 \;\Rightarrow\; 11^{-1} = 7$ in $\mathbb{Z}_{19}$

- Beispiel $\mathbb{Z}_{18}$

a	1	5	7	11	13	17
a^{-1}	1	11	13	5	7	17

- $\mathbb{Z}_n^*$ hat multiplikative Gruppenstruktur, denn wegen

$$\mathrm{ggT}(a,n)=1, \mathrm{ggT}(b,n)=1 \Rightarrow \mathrm{ggT}(a*b,n)=1$$

gilt

- $\forall a,b \in \mathbb{Z}_n^* : a *_n b \in \mathbb{Z}_n^*$
- $\forall a \in \mathbb{Z}_n^* \; \exists b \in \mathbb{Z}_n^* : a *_n b = 1$ (eeA!!!)

und daher:

$$U_n = \langle \mathbb{Z}_n^* ; *_n \rangle \text{ ist eine kommutative Gruppe}$$

("Einheitengruppe modulo n")

Beispiel $\mathbb{Z}_{18}^* = U_{18}$:

$*_{18}$	1	5	7	11	13	17
1	1	5	7	11	13	17
5	5	7	17	1	11	13
7	7	17	13	5	1	11
11	11	1	5	13	17	7
13	13	11	1	17	7	5
17	17	13	11	7	5	1

- Folgerung (Gleichungslösen in $\mathbb{Z}_n$)

$$\forall a \in \mathbb{Z}_n^* \; \forall b \in \mathbb{Z}_n \; \exists_1 x \in \mathbb{Z}_n : a *_n x = b$$

nämlich $x = a^{-1} *_n b$

- Aus dem Lösungsverfahren für $\mathbb{Z}$ folgt:
  - Für $a,b \in \mathbb{Z}_n$ gilt:

$$a *_n x = b \text{ lösbar} \iff \mathrm{ggT}(a,n) \mid b$$

  - Falls $\mathrm{ggT}(a,n)=d$, so gibt es genau d verschiedene Lösungen

$$x_0 + k \cdot \frac{n}{d} \; (0 \le k < d) \text{ wobei } \frac{a}{d} \cdot x_0 = \frac{n}{d} \text{ in } \mathbb{Z}_{n/d} \text{ (\textit{eindeutig})}$$

Eulers φ-Funktion

- Definition: $\sharp U_n = \sharp \mathbb{Z}_n^* = \varphi(n)$
- erste Werte:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\varphi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8

- φ ist *multiplikativ* in folgendem Sinne:

$$\mathrm{ggT}(a,b)=1 \quad \Rightarrow \quad \varphi(a*b)=\varphi(a)*\varphi(b)$$

- Multiplikativität ist äquivalent zu

$$n = \sum_{d \mid n} \varphi(d) \quad (n>0)$$

$\sum_{d \mid n}$: Summe über alle positiven Teiler d von n

- Beispiel: $18 = \varphi(1)+\varphi(2)+\varphi(3)+\varphi(6)+\varphi(9)+\varphi(18) = 1+1+2+2+6+6$

Folgerung aus der Multiplikativität

- φ ist durch seine Werte auf Primzahlpotenzen bestimmt:

$$p \text{ prim} \quad \Rightarrow \quad \varphi(p^e) = p^{e-1}(p-1) \quad (e > 0)$$

- Speziell: $\langle \mathbb{Z}_n; +_n, *_n \rangle$ Körper $\Leftrightarrow$ n Primzahl $\Leftrightarrow \varphi(n) = n - 1$
- Folgerung: falls $n = \prod_{i=1}^{k} p_i^{e_i}$

$$\varphi(n) = \prod_{i=1}^{k} p_i^{e_i - 1}(p_i - 1) = n \cdot \prod_{\substack{p \mid n \\ p \text{ prim}}} \left(1 - \frac{1}{p}\right)$$

- Also: Berechnung von $\varphi(n)$ einfach, falls Primfaktorisierung von n bekannt
- Aber: bis heute ist kein effizientes Verfahren zur Berechnung von $\phi(n)$ (z.B. ohne Kenntnis der Primfaktorisierung) bekannt!

Beispiele

- $n = 12$

$$\mathbb{Z}_{12}^* = \{1, 5, 7, 11\} \qquad \varphi(12) = \sharp\mathbb{Z}_{12}^* = 4$$
$$12 = 2^2 * 3 \qquad \varphi(12) = 12 \cdot (1 - \frac{1}{2})(1 - \frac{1}{3}) = 4$$

- $n = 13$

$$\mathbb{Z}_{13}^* = \{1, 2, 3, \ldots, 12\} \qquad \varphi(13) = \sharp\mathbb{Z}_{13}^* = 12$$
$$13 = 13 \qquad \varphi(13) = 13 \cdot (1 - \frac{1}{13}) = 12$$

- $n = 67914$

$$67914 = 2 * 3^2 * 7^3 * 11$$
$$\varphi(67914) = n \cdot (1 - \frac{1}{2}) \cdot (1 - \frac{1}{3}) \cdot (1 - \frac{1}{7}) \cdot (1 - \frac{1}{11}) = 17640$$

Beispiel: Zerlegung von $\mathbb{Z}_{18}$

$$\begin{aligned}
\mathbb{Z}_{18} &= \{0, 1, 2, 3, \ldots, 17\} \\
&= \underbrace{\{1, 5, 7, 11, 13, 17\}}_{ggT(a,18)=1} \uplus \underbrace{\{2, 4, 8, 10, 14, 16\}}_{ggT(a,18)=2} \\
&\quad \uplus \underbrace{\{3, 15\}}_{ggT(a,18)=3} \uplus \underbrace{\{6, 12\}}_{ggT(a,18)=6} \uplus \underbrace{\{9\}}_{ggT(a,18)=9} \uplus \underbrace{\{0\}}_{ggT(a,18)=18} \\
&= 1 \cdot \{1, 5, 7, 11, 13, 17\} \uplus 2 \cdot \{1, 2, 4, 5, 7, 8\} \\
&\quad \uplus 3 \cdot \{1, 5\} \uplus 6 \cdot \{1, 2\} \uplus 9 \cdot \{1\} \uplus 18 \cdot \{0\} \\
&= 1 \cdot \mathbb{Z}_{18}^* \uplus 2 \cdot \mathbb{Z}_9^* \uplus 3 \cdot \mathbb{Z}_6^* \uplus 6 \cdot \mathbb{Z}_3^* \uplus 9 \cdot \mathbb{Z}_2^* \uplus 18 \cdot \mathbb{Z}_1^* \\
&= \biguplus_{d \mid 18} d \cdot \mathbb{Z}_{18/d}^*
\end{aligned}$$

$$\Rightarrow 18 = \sum_{d \mid 18} \sharp\mathbb{Z}_{18/d}^* = \sum_{d \mid 18} \varphi(\frac{18}{d}) = \sum_{d \mid 18} \varphi(d)$$

Nachweis der Multiplikativität

- Für $n \in \mathbb{N}$ und $d \mid n$ ist

$$A_{n,d} := \{\, a \in \mathbb{Z}_n \,;\, \text{ggT}(a, n) = d \,\} = d \cdot \mathbb{Z}_{n/d}^*$$

- Wegen $\mathbb{Z}_n = \biguplus_{d \mid n} A_{n,d}$ folgt

$$n = \sharp\mathbb{Z}_n = \sum_{d \mid n} \sharp A_{n,d} = \sum_{d \mid n} \mathbb{Z}_{n/d}^* = \sum_{d \mid n} \varphi(n/d) = \sum_{d \mid n} \varphi(d)$$

- Induktion über Teilerstruktur für a, b mit $\text{ggT}(a, b) = 1$:
  - Angenommen: für e, f mit $e|a$, $f|b$ und $e \cdot f \neq a \cdot b$ sei $\varphi(e) \cdot \varphi(f) = \varphi(e \cdot f)$ bereits gezeigt, dann folgt $\varphi(a) \cdot \varphi(b) = \varphi(a \cdot b)$ aus

$$\sum_{e|a} \sum_{f|b} \varphi(e \cdot f) = \sum_{d|a \cdot b} \varphi(d) = a \cdot b = \sum_{e|a} \varphi(e) \cdot \sum_{f|b} \varphi(f)$$

Exkurs: Möbius-Inversion

(A.F. Möbius: *Über eine besondere Art der Umkehrung von Reihen*, 1831)

- Definiere eine multiplikative Funktion $\mu : \mathbb{N}_{>0} \to \{0, \pm 1\}$ (die *Möbiusfunktion*) durch

$$\mu(1) = 1, \quad \mu(p^e) = \begin{cases} -1 & \text{für } e = 1 \\ 0 & \text{für } e > 1 \end{cases} \quad (p \text{ Primzahl})$$

also für $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$

$$\mu(n) = \begin{cases} (-1)^k & \text{falls } e_1 = e_2 = \ldots = e_k = 1 \quad (\textit{quadratfrei}) \\ 0 & \text{sonst} \end{cases}$$

- erste Werte:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\mu(n)$	1	−1	−1	0	−1	1	−1	0	0	1	−1	0	−1	1

- μ hat die Eigenschaft

$$\sum_{d \mid n} \mu(d) = \delta_{n,1} = \begin{cases} 1 & n = 1 \\ 0 & n > 1 \end{cases}$$

- Beweis mittels Binomialformel. Für $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$ ist

$$\begin{aligned} \sum_{d \mid n} \mu(d) &= \sum \{ \mu(d) ; d \mid n, d \text{ quadratfrei} \} \\ &= \sum_{0 \le j \le k} \sum_{1 \le i_1 < i_2 < \ldots < i_j \le k} \mu(p_{i_1} \cdot p_{i_2} \cdots p_{i_j}) \\ &= \sum_{0 \le j \le k} \binom{k}{j} (-1)^j = (1-1)^k = \delta_{k,0} = \begin{cases} 0 & k > 0 \\ 1 & k = 0 \end{cases} \end{aligned}$$

- Sind $f, g : \mathbb{N}_{>0} \to \mathbb{C}$ Funktionen, so sind die beiden Aussagen

$$g(n) = \sum_{d \mid n} f(d) \quad (n \ge 1))$$

$$f(n) = \sum_{d \mid n} \mu(d) \cdot g(\frac{n}{d}) = \sum_{d \mid n} \mu(\frac{n}{d}) \cdot g(d) \quad (n \ge 1)$$

äquivalent zueinander. Insbesondere ist dann jede der beiden Funktionen durch die andere eindeutig bestimmt

- Beweis (durch Vertauschen von Summationen):

$$\begin{aligned} \sum_{d \mid n} \mu(d) \cdot g(\frac{n}{d}) &= \sum_{d \mid n} \mu(d) \cdot \sum_{e \mid \frac{n}{d}} f(e) = \sum_{e \mid n} f(e) \cdot \sum_{d \mid \frac{n}{e}} \mu(d) \\ &= \sum_{e \mid n} f(e) \cdot \delta_{e,n} = f(n) \end{aligned}$$

Das spezielle Beispiel $g(n) = n, f(n) = \varphi(n)$

$$g(n) = n = \sum_{d \mid n} \varphi(d) = \sum_{d \mid n} f(d)$$

$$f(n) = \varphi(n) = \sum_{d \mid n} \mu(d) \cdot g(\frac{n}{d}) = \sum_{d \mid n} \mu(d) \cdot \frac{n}{d}$$

Beachte nun für $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$

$$\begin{aligned} \sum_{d \mid n} \mu(d) \cdot \frac{n}{d} &= \sum_{0 \le j \le k} \sum_{1 \le i_1 < i_2 < \ldots < i_j \le k} \mu(p_{i_1} \cdot p_{i_2} \cdots p_{i_j}) \cdot \frac{n}{p_{i_1} \cdot p_{i_2} \cdots p_{i_j}} \\ &= n \cdot \sum_{0 \le j \le k} \sum_{1 \le i_1 < i_2 < \ldots < i_j \le k} \cdot \frac{(-1)^j}{p_{i_1} \cdot p_{i_2} \cdots p_{i_j}} \\ &= n \cdot \prod_{\substack{p \mid n \\ p \text{ prim}}} \left(1 - \frac{1}{p}\right) \end{aligned}$$

Gruppentheoretisches Intermezzo (additiv)

G kommutative Gruppe , H Untergruppe von G

- Definition der H-Äquivalenz:

$$\forall a, b \in G \ : \ a \sim_H b \quad \leftrightarrow \quad b - a \in H$$

$$\begin{aligned} 0 \in H \quad &\Rightarrow \quad \sim_H \text{ ist reflexiv} \\ a \in H \rightarrow -a \in H \quad &\Rightarrow \quad \sim_H \text{ ist symmetrisch} \\ a, b \in H \rightarrow a + b \in H \quad &\Rightarrow \quad \sim_H \text{ ist transitiv} \end{aligned}$$

- Folgerung: $\sim_H$ ist eine Äquivalenzrelation auf G
  Die Äquivalenzklassen ("Nebenklassen" (*cosets*) von H)

$$[a]_H = \{b \in G \,;\, a \sim_H b\} = \{a + h \,;\, h \in H\} = a + H$$

partitionieren G in "gleich grosse" Teile, denn

$H \rightarrow [a]_H \ : \ h \mapsto a+h$ ist bijektiv, also $\forall a \in G \ : \ \sharp[a]_H = \sharp H$

Gruppentheoretisches Intermezzo (multiplikativ)

G Gruppe, H Untergruppe von G

- Definition der H-Äquivalenz:

$$\forall a, b \in G \ : \ a \sim_H b \quad \leftrightarrow \quad a^{-1} * b \in H$$

$$\begin{aligned} e \in H \quad &\Rightarrow \quad \sim_H \text{ ist reflexiv} \\ a \in H \rightarrow a^{-1} \in H \quad &\Rightarrow \quad \sim_H \text{ ist symmetrisch} \\ a, b \in H \rightarrow a * b \in H \quad &\Rightarrow \quad \sim_H \text{ ist transitiv} \end{aligned}$$

- Folgerung: $\sim_H$ ist eine Äquivalenzrelation auf G
  Die Äquivalenzklassen ("Nebenklassen" (*cosets*) von H)

$$[a]_H = \{b \in G \,;\, a \sim_H b\} = \{a * h \,;\, h \in H\} = a * H$$

partitionieren G in "gleich grosse" Teile, denn

$H \rightarrow [a]_H \ : \ h \mapsto a*h$ ist bijektiv, also $\forall a \in G \ : \ \sharp[a]_H = \sharp H$

Bemerkung für die Definitionen der H-Äquivalenz bezüglich einer Untergruppe H in einer nicht-kommutativen Gruppe G:

- Es kommt bei der Multiplikation auf die Reihenfolge der Faktoren an. Die Aussagen $a^{-1} * b \in H$ (d.h. $b \in a * H$) und $b * a^{-1} \in H$ (d.h. $b \in H * a$) sind i.a. nicht gleichwertig. Entsprechend muss man "Linksnebenklassen" $a * H$ und "Rechtsnebenklassen" $H * a$ unterschieden. Es kann vorkommen, dass $a * H \neq H * a$
- Ist H eine Untergruppe von G und gilt $a * H = H * a$ für alle $a \in G$, so spricht man von H als einer normalen Untergruppe oder einem Normalteiler von G.
  Normalteiler sind besonders angenehme Untergruppen, weil man auf der Menge der Nebenklassen wieder eine Gruppenoperation definieren kann

$$\forall a, b \in G \ : \ [a]_H * [b]_H := [a * b]_H$$

Für Nicht-Normalteiler funktioniert das nicht!

Zyklische Gruppen

- G Gruppe (multiplikativ), $a \in G$

$$\langle a \rangle = \{\ldots, a^{-2}, a^{-1}, e = a^0, a = a^1, a^2, a^3, \ldots\} = \{a^k \,;\, k \in \mathbb{Z}\}$$

ist eine Untergruppe von G: die *von a erzeugte Untergruppe*

- Falls $\sharp\langle a \rangle = \infty$: $\langle a \rangle$ hat die gleiche Struktur wie $(\mathbb{Z}, +)$:
  - a hat *unendliche Ordnung*, $ord_G(a) = \infty$,
  - $\langle a \rangle$ ist eine *unendliche zyklische Gruppe*.
- Falls $\sharp\langle a \rangle = n < \infty$: $\langle a \rangle$ hat die gleiche Struktur wie $(\mathbb{Z}_n, +_n)$:
  - a hat *endliche Ordnung*, $ord_G(a) = n$,
  - $\langle a \rangle$ ist eine *endliche zyklische Gruppe* der Ordnung n.
- Eine Gruppe G heisst *zyklische Gruppe*, wenn es ein $a \in G$ gibt mit $G = \langle a \rangle$.
- Die Gruppen $(\mathbb{Z}, +)$ und $(\mathbb{Z}_n, +_n)$ (für $n \geq 1$) sind — bis auf Isomorphie – die einzigen zyklischen Gruppen.

Beispiele

Betrachten Einheitengruppen $U_n = \mathbb{Z}_n^*$:
$\sharp\langle a\rangle = ord_n(a)$: die *Ordnung (Periode) von a modulo n*

- $n = 7$

$$\begin{aligned}
\langle 1\rangle &= \{1\} && ord_7(1) = 1\\
\langle 2\rangle &= \{2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 1, \ldots\} = \{1, 2, 4\} && ord_7(2) = 3\\
\langle 3\rangle &= \{1, 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5, 3^6 = 1, \ldots\}\\
&= \{1, 2, 3, 4, 5, 6\} && ord_7(3) = 6\\
\langle 4\rangle &= \{1, 4, 2, 1, \ldots\} = \{1, 2, 4\} && ord_7(4) = 3\\
\langle 5\rangle &= \{1, 5, 4, 6, 2, 31, \ldots\} = \{1, 2, 3, 4, 5, 6\} && ord_7(5) = 6\\
\langle 6\rangle &= \{1, 6, 1, \ldots\} = \{1, 6\} && ord_7(6) = 2
\end{aligned}$$

$U_7 = \mathbb{Z}_7^* = \langle 3\rangle = \langle 5\rangle$ ist also eine zyklische Gruppe

Beispiele

- $n = 8$

$$\begin{aligned}
\langle 1\rangle &= \{1\} && ord_8(1) = 1\\
\langle 3\rangle &= \{1, 3, 3^2 = 1, \ldots\} = \{1, 3\} && ord_8(3) = 2\\
\langle 5\rangle &= \{1, 5, 5^2 = 1, \ldots\} = \{1, 5\} && ord_8(5) = 2\\
\langle 7\rangle &= \{1, 7, 7^2 = 1, \ldots\} = \{1, 7\} && ord_8(7) = 2
\end{aligned}$$

$U_8 = \mathbb{Z}_8^* = \{1, 3, 5, 7\} \neq \langle 1\rangle, \langle 3\rangle, \langle 5\rangle, \langle 7\rangle$.
U_8 ist also keine zyklische Gruppe.

Beispiele

- $n = 9$

$$\begin{aligned}
\langle 1\rangle &= \{1\} && ord_9(1) = 1\\
\langle 2\rangle &= \{1, 2, 2^2 = 4, 2^3 = 8, 2^4 = 7, 2^5 = 5, 2^6 = 1, \ldots\}\\
&= \{1, 2, 4, 5, 7, 8\} && ord_9(2) = 6\\
\langle 4\rangle &= \{1, 4, 7, 1, \ldots\} = \{1, 4, 7\} && ord_9(4) = 3\\
\langle 5\rangle &= \{1, 5, 7, 8, 4, 2, 1, \ldots\} = \langle 2\rangle && ord_9(5) = 6\\
\langle 7\rangle &= \langle 4\rangle && ord_9(7) = 3\\
\langle 8\rangle &= \{1, 8\} && ord_9(8) = 2
\end{aligned}$$

$U_9 = \mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\} = \langle 2\rangle = \langle 5\rangle$
U_9 ist also eine zyklische Gruppe.

Beispiele

- $n = 10$

$$\begin{aligned}
\langle 1\rangle &= \{1\} && ord_{10}(1) = 1\\
\langle 3\rangle &= \{1, 3, 9, 7, 1, \ldots\} && ord_{10}(3) = 4\\
\langle 7\rangle &= \{1, 7, 9, 3, 1, \ldots\} && ord_{10}(7) = 4\\
\langle 9\rangle &= \{1, 9, 1, \ldots\} && ord_{10}(9) = 2
\end{aligned}$$

$U_{10} = \mathbb{Z}_{10}^* = \{1, 3, 7, 9\} = \langle 3\rangle = \langle 7\rangle$
U_{10} ist also eine zyklische Gruppe.

- $n = 11$

$$\begin{aligned}
\langle 1\rangle &= \{1\} & ord_{11}(1) &= 1\\
\langle 2\rangle &= \{1,2,4,8,5,10,9,7,3,6,1,\ldots\} & ord_{11}(2) &= 10\\
\langle 3\rangle &= \{1,3,9,5,4,1,\ldots\} & ord_{11}(3) &= 5\\
\langle 4\rangle &= \{1,4,5,9,3,1,\ldots\} & ord_{11}(4) &= 5\\
\langle 5\rangle &= \{1,5,3,4,9,1,\ldots\} & ord_{11}(5) &= 5\\
\langle 6\rangle &= \{1,6,3,7,9,10,5,8,4,2,1,\ldots\} & ord_{11}(6) &= 10\\
\langle 7\rangle &= \{1,7,5,2,3,10,4,6,9,8,1,\ldots\} & ord_{11}(7) &= 10\\
\langle 8\rangle &= \{1,8,9,6,4,10,3,2,5,7,1,\ldots\} & ord_{11}(8) &= 10\\
\langle 9\rangle &= \{1,9,4,3,5,1,\ldots\} & ord_{11}(9) &= 5\\
\langle 10\rangle &= \{1,10,1,\ldots\} & ord_{11}(10) &= 2
\end{aligned}$$

$U_{11} = \mathbb{Z}_{11}^* = \{1,2,3,\ldots,10\} = \langle 2\rangle = \langle 6\rangle = \langle 7\rangle = \langle 8\rangle$
U_{11} ist also eine zyklische Gruppe.

- $n = 12$

$$\begin{aligned}
\langle 1\rangle &= \{1\} & ord_{12}(1) &= 1\\
\langle 5\rangle &= \{1,5,1,\ldots\} & ord_{12}(5) &= 2\\
\langle 7\rangle &= \{1,7,1,\ldots\} & ord_{12}(7) &= 2\\
\langle 11\rangle &= \{1,11,1,\ldots\} & ord_{12}(10) &= 2
\end{aligned}$$

$U_{12} = \mathbb{Z}_{12}^* = \{1,5,7,11\} \neq \langle 1\rangle, \langle 5\rangle, \langle 7\rangle, \langle 11\rangle$
U_{12} ist also keine zyklische Gruppe.

Die Sätze von Lagrange, Fermat und Euler

- Satz von Lagrange
  G endliche Gruppe, $H \subseteq G$ Untergruppe $\Rightarrow \sharp H \,|\, \sharp G$.
- Folgerung:
  - G endliche (multiplikative) Gruppe mit neutralem Element e.
    Für jedes $a \in G$ gilt:
    $$ord_G(a) \,|\, \sharp G$$
    und somit
    $$a^{\sharp G} = a^{ord_G(a)} = e$$
  - $ord_G(a)$ ist die kleinste positive Zahl t mit $a^t = e$

Nützliche Regeln für das Rechnen mit Ordnungen:

- Wegen der Divisionseigenschaft gilt für $k \in \mathbb{Z}$:
  $$a^k = a^{k \bmod ord_G(a)} = a^{k \bmod \sharp G}$$
  also
  $$a^k = e \quad \Leftrightarrow \quad ord_G(a) \,|\, k$$
- Für $k \in \mathbb{Z}$ gilt:
  $$ord_G(a^k) = \frac{ord_G(a)}{ggT(k, ord_G(a))} = \frac{kgV(k, ord_G(a))}{k}$$
  Begründung: für $a, b, c \in \mathbb{Z}$ gilt
  $$a \,|\, (b \cdot c) \quad \Leftrightarrow \quad \frac{a}{ggT(a,b)} \,|\, c \quad \Leftrightarrow \quad \frac{kgV(a,b)}{b} \,|\, c$$

- Spezielle Situation der Einheitengruppen $U_n = \mathbb{Z}_n^*$ modulo n:

$$\forall a \in U_n : ord_n(a) = \sharp\langle a\rangle \mid \sharp G = \varphi(n)$$

also

$$\forall a \in U_n : a^{\varphi(n)} = a^{ord_n(a)} = 1$$

- Satz von Euler: Für alle $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

- Satz von Fermat: Für jede Primzahl p und beliebiges $a \in \mathbb{Z}$ mit $p \not| a$ gilt

$$a^{p-1} \equiv 1 \pmod{p}$$

und damit auch $a^p \equiv a \pmod{p}$ für alle $a \in \mathbb{Z}$.

- Für alle $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$ und $k \in \mathbb{Z}$ gilt:

$$a^k = a^{k \bmod \varphi(n)} = a^{k \bmod ord_n(a)} \bmod n$$

Zur Illustration: $\mathbb{Z}_{36}^*$

- $\mathbb{Z}_{36}^* = \{1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35\}, \varphi(36) = 12$
- $\varphi(36) = 36 \cdot (1 - \frac{1}{2})(1 - \frac{1}{3}) = 12$
- $\langle 5\rangle = \{1, 5, 25, 17, 13, 29\}$, $ord_{36}(5) = 6$
- $5^{117} \equiv 5^9 \equiv 5^3 \equiv 17 \bmod 36$
- $\langle 13\rangle = \{1, 13, 25\}$, $ord_{36}(13) = 3$
- $13^{116} \equiv 13^8 \equiv 13^2 \equiv 25 \bmod 36$

Zur Illustration: $\mathbb{Z}_{17}^*$

- $\mathbb{Z}_{17}^* = \{1, 2, 3, \ldots, 16\}, \varphi(17) = 16$
- $\varphi(17) = 17 \cdot (1 - \frac{1}{17}) = 16$
- $\langle 3\rangle = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$, $ord_{17}(3) = 18$
- $3^{99} \equiv 3^3 \equiv 10 \bmod 17$
- $\langle 4\rangle = \{1, 4, 16, 13\}$, $ord_{17}(4) = 4$
- $4^{106} \equiv 4^{10} \equiv 4^2 \equiv 16 \bmod 17$

Ein Blick in Richtung Faktorisierung

- Problem: eine grosse Zahl $N \in \mathbb{N}$ soll faktorisiert werden.
- Annahme: man kann für a mit $1 < a < N$ die Ordnung (Periode) $ord_N(a)$ bestimmen.
- Man wählt sich ein a und berechnet $\mathrm{ggT}(a, N)$.
  - Falls $\mathrm{ggT}(a, N) > 1$ hat man einen Faktor von N gefunden!
  - Andernfalls ist $a \in \mathbb{Z}_N^*$ und $ord_N(a)$ ist definiert.
- Falls $ord_N(a)$ ungerade, ist a nicht weiter brauchbar. (Der Fall tritt nur selten ein)
- Falls $ord_N(a) = 2t$, so gilt $N|(a^{2t} - 1)$ und $N \not| (a^t - 1)$, also

$$N \mid (a^{2t} - 1) = (a^t - 1)(a^t + 1)$$

  - Falls $N \mid (a^t + 1)$, ist a nicht weiter brauchbar. (Der Fall tritt nur selten ein)
  - Falls $N \not| (a^t + 1)$, müssen $\mathrm{ggT}(N, a^t + 1)$ und $\mathrm{ggT}(N, a^t - 1)$ echte Faktoren von N liefern!

Beispiel: die Faktorisierung von $N = 15$

- Für $a \in \{3, 5, 6, 9, 10, 12\}$ ist $\text{ggT}(a, 15) > 1$
- Für $a \in U_{15}$:

a	$ord_{15}(a)$	t	a^t+1	$\text{ggT}(15, a^t+1)$	a^t-1	$\text{ggT}(15, a^t-1)$
2	4	2	5	5	3	3
4	2	1	5	5	3	3
7	4	2	50	5	48	3
8	4	2	65	5	63	3
11	2	1	12	3	19	5
13	4	2	170	5	168	3
14	2	1	15	15	13	1

- Ausser $a = 14$ (und $a = 1$, natürlich!) liefern alle $a \in U_{14}$ eine Faktorisierung, falls man $ord_{15}(a)$ kennt.

Die Restklassenringe $\mathbb{Z}_n$
Ordnung und Faktorisierung

- $N = 77$:
  - $\varphi(77) = 77 \cdot (1 - \frac{1}{7}) \cdot (1 - \frac{1}{11}) = 60$
  - Von den 60 Elementen $a \in U_{77}$ haben 15 eine ungerade Ordnung $ord_{77}(a)$.
  - Von den 45 Elementen $a \in U_{77}$ mit gerader Ordnung $ord_{77}(a) = 2t$ liefern 30 mittels $\text{ggT}(a^t + 1, 77)$ einen echten Teiler von 77.
- $N = 119$
  - $\varphi(119) = 119 \cdot (1 - \frac{1}{7}) \cdot (1 - \frac{1}{17}) = 96$
  - Von den 96 Elementen $a \in U_{119}$ haben drei (1,18,86) eine ungerade Ordnung $ord_{119}(a)$.
  - Von den 93 Elementen $a \in U_{119}$ mit gerader Ordnung $ord_{119}(a) = 2t$ liefern 90 (ausgenommen 33, 101, 118) , mittels $\text{ggT}(a^t + 1, 119)$ einen echten Teiler von 119.

- Exponentiation: das Problem
  - Gegeben:
    (multiplikative) Halbgruppe $(H, *)$, Element $a \in H$, $n \in \mathbb{N}$
  - Aufgabe: berechne das Element

$$a^{*n} = \underbrace{a * a * a * \cdots * a}_{n} \in H$$

    (schreiben ab jetzt a^n statt a^{*n})
  - Hinweis: der wesentliche "Trick" funktioniert immer, wenn $*$ eine assoziative (aber nicht notwendig kommutative) Operation ist — also nicht nur für Zahlen, sondern z.B. auch für Polynome, Matrizen, Funktionen, ...

- Banale Idee: iterierte Multiplikation
- Algorithmus:

  Require: $a \in H, n \in \mathbb{N}$
  Ensure: $x = a^n$
  if $n = 0$ **then**
  Return(e)
  end if
  $x \leftarrow a$
  for $i = 1$ to $n - 1$ **do**
  $x \leftarrow x * a$
  end for
  Return(x)
- Dies erfordert $n - 1$ Multiplikationen in H.

Bessere Idee:

- In jeder Halbgruppe $(H, *)$ kann man zur Berechnung der Exponentiation

$$H \times \mathbb{N} \to H \ : \ (a, n) \mapsto a^n$$

  die Binärdarstellung des Exponenten n verwenden
- Rekursive Struktur:

$$a^n = \left\{ \begin{array}{ll} \left(a^{\frac{n}{2}}\right)^2 & \text{falls } n \text{ gerade} \\ a * \left(a^{\frac{n-1}{2}}\right)^2 & \text{falls } n \text{ ungerade} \end{array} \right\} = a^{n \bmod 2} \cdot \left(a^{n \div 2}\right)^2$$

  $\to$ "schnelle" oder "binäre" Exponentiation, Exponentiation mittels "square-and-multiply"
- Man kann die Binärdarstellung von n von links nach rechts (LR) oder von rechts nach links (RL) abarbeiten

Notation:

- $(n)_2$: Binärdarstellung von n (ohne führende Nullen)
- $\ell(n)$: Länge der Binärdarstellung von $n = \lfloor \log(n) \rfloor + 1$
- $\nu(n) = \sharp_1(n)_2$: Anzahl der Einsen in $(n)_2$

Rekursionen:

- $\ell(2n) = \ell(n) + 1$
  $\ell(2n+1) = \ell(n) + 1$
  $\ell(1) = 1$
- $\nu(2n) = \nu(n)$
  $\nu(2n+1) = \nu(n) + 1$
  $\nu(1) = 1$

Beispiel $n = 155$ mit $(155)_2 = 10011011$, $\ell(155) = 8$, $\nu(155) = 5$

- LR-Methode

$(1)_2$	$=$	1	$a^1 = a$
$(2)_2$	$=$	10	$a^2 = (a^1)^2$
$(4)_2$	$=$	100	$a^4 = (a^2)^2$
$(9)_2$	$=$	1001	$a^9 = (a^4)^2 * a$
$(19)_2$	$=$	10011	$a^{19} = (a^9)^2 * a$
$(38)_2$	$=$	100110	$a^{38} = (a^{19})^2$
$(77)_2$	$=$	1001101	$a^{77} = (a^{38})^2 * a$
$(155)_2$	$=$	10011011	$a^{155} = (a^{77})^2 * a$

erfordert $7 = \ell(155) - 1$ Quadrierungen und $4 = \nu(155) - 1$ zusätzliche Multiplikationen, also insgesamt $\ell(155) + \nu(n) - 2$ Multiplikationen

- RL-Methode

$(1)_2$	$=$	1	$a^1 = a$
			$a^2 = (a^1)^2$
$(3)_2$	$=$	11	$a^3 = a^2 * a$
			$a^4 = (a^2)^2$
			$a^8 = (a^4)^2$
$(11)_2$	$=$	1011	$a^{11} = a^8 * a^3$
			$a^{16} = (a^8)^2$
$(27)_2$	$=$	11011	$a^{27} = a^{16} * a^{11}$
			$a^{32} = (a^{16})^2$
			$a^{64} = (a^{32})^2$
			$a^{128} = (a^{64})^2$
$(155)_2$	$=$	10011011	$a^{155} = a^{128} * a^{27}$

erfordert $7 = \ell(155) - 1$ Quadrierungen und $4 = \nu(155) - 1$ zusätzliche Multiplikationen, also insgesamt $\ell(155) + \nu(n) - 2$ Multiplikationen

Aufwand:

- $M(n)$: Anzahl der Multiplikationen (incl. Quadrierungen) in H zur Berechung von a^n mittels LR- bzw. RL-Methode
- Rekursion:

$$\begin{array}{rcl} M(2n) & = & M(n) + 1 \\ M(2n+1) & = & M(n) + 2 \end{array} \qquad M(1) = 0$$

- Folgerung: $M(n) = \ell(n) + \nu(n) - 2$
- Beweis (Induktion)

$$\begin{aligned} M(1) &= 0 = && 1 + 1 - 2 = \ell(1) + \nu(1) - 2 \\ M(2n) &= M(n) + 1 = && \ell(n) + \nu(n) - 1 = \ell(2n) + \nu(2n) - 2 \\ M(2n+1) &= M(n) + 2 = && \ell(n) + \nu(n) = \ell(2n+1) + \nu(2n+1) - 2 \end{aligned}$$

- Also gilt insbesondere:

$$\lfloor \log n \rfloor \leq M(n) \leq 2\lfloor \log n \rfloor \quad \text{d.h.} \quad M(n) \in \Theta(\log n)$$

This binary method is easily justified by a consideration of the sequence of exponents in the calculation: If we reinterpret "S" as the operation of multiplying by 2 and "X" as the operation of adding 1, and if we start with 1 instead of x, the rule will lead to a computation of n because of the properties of the binary number system. The method is quite ancient; it appeared before 200 B.C. in Piṅgala's Hindu classic *Chandaḥ-sûtra* [see B. Datta and A. N. Singh, *History of Hindu Mathematics* **2** (Lahore: Motilal Banarsi Das, 1935), 76]. There seem to be no other references to this method outside of India during the next 1000 years, but a clear discussion of how to compute 2^n efficiently for arbitrary n was given by al-Uqlīdisī of Damascus in A.D. 952; see *The Arithmetic of al-Uqlīdisī* by A. S. Saidan (Dordrecht: D. Reidel, 1975), 341–342, where the general ideas are illustrated for $n = 51$. See also al-Bīrūnī's *Chronology of Ancient Nations*, edited and translated by E. Sachau (London: 1879), 132–136; this eleventh-century Arabic work had great influence.

Abbildung: Knuth, TAOCP vol.2, ch. 4.6.3 (LR-Methode)

The S-and-X binary method for obtaining x^n requires no temporary storage except for x and the current partial result, so it is well suited for incorporation in the hardware of a binary computer. The method can also be readily programmed; but it requires that the binary representation of n be scanned from left to right. Computer programs generally prefer to go the other way, because the available operations of division by 2 and remainder mod 2 will deduce the binary representation from right to left. Therefore the following algorithm, based on a right-to-left scan of the number, is often more convenient:

Abbildung: Knuth, TAOCP vol.2, ch. 4.6.3 (LR vs. RL)

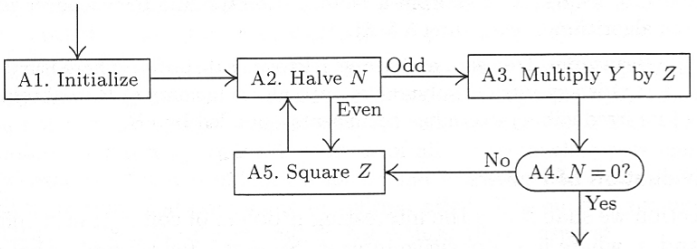


Fig. 13. Evaluation of x^n, based on a right-to-left scan of the binary notation for n.

Abbildung: Knuth, TAOCP vol.2, ch. 4.6.3 (RL)

Algorithm A (*Right-to-left binary method for exponentiation*). This algorithm evaluates x^n, where n is a positive integer. (Here x belongs to any algebraic system in which an associative multiplication, with identity element 1, has been defined.)

A1. [Initialize.] Set $N \leftarrow n$, $Y \leftarrow 1$, $Z \leftarrow x$.

A2. [Halve N.] (At this point, $x^n = Y\,Z^N$.) Set $N \leftarrow \lfloor N/2 \rfloor$, and at the same time determine whether N was even or odd. If N was even, skip to step A5.

A3. [Multiply Y by Z.] Set $Y \leftarrow Z$ times Y.

A4. [$N = 0$?] If $N = 0$, the algorithm terminates, with Y as the answer.

A5. [Square Z.] Set $Z \leftarrow Z$ times Z, and return to step A2. ∎

Abbildung: Knuth, TAOCP vol.2, ch. 4.6.3 (RL)

The great calculator al-Kāshī stated Algorithm A in A.D. 1427 [*Istoriko-Mat. Issledovaniiâ* **7** (1954), 256–257]. The method is closely related to a procedure for multiplication that was actually used by Egyptian mathematicians as early as 2000 B.C.; for if we change step A3 to "$Y \leftarrow Y+Z$" and step A5 to "$Z \leftarrow Z+Z$", and if we set Y to zero instead of unity in step A1, the algorithm terminates with $Y = nx$. [See A. B. Chace, *The Rhind Mathematical Papyrus* (1927); W. W. Struve, *Quellen und Studien zur Geschichte der Mathematik* **A1** (1930).] This is a practical method for multiplication by hand, since it involves only the simple operations of doubling, halving, and adding. It is often called the "Russian peasant method" of multiplication, since Western visitors to Russia in the nineteenth century found the method in wide use there.

Abbildung: Knuth, TAOCP vol.2, ch. 4.6.3 (RL)

Require: $a \in H, n \in \mathbb{N}$
Ensure: $x = a^n$
  $x \leftarrow e$
  $A \leftarrow a$
  $N \leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $A \leftarrow A * A$
      $N \leftarrow N/2$
    else $\{N$ is odd$\}$
      $x \leftarrow x * A$
      $N \leftarrow N - 1$
    end if
  end while
  Return(x)

- Maple-Programm (rekursiv)

```
power := proc (a,n::nonnegint)
if n=0 then RETURN(1)
   else t := power(a,iquo(n/2))^2;      (Q)
   if odd(e) then t := t*a fi;          (M)
   RETURN(t)
   fi;
end;
```


Zur Anwendung:

- Knuth (TAOCP, vol2., ch. 4.6.3) diskutiert, wann man "binäre" ("schnelle") Exponentiation verwenden sollte und wann nicht!
- Zitat: *The point of these remarks is that binary methods are nice, but not a panacea. They are most applicable when the time to multiply $x^j \cdot x^k$ is essentially independent of j and k (for example, when we are doing floting point multiplication, or multiplication modulo m); in such cases the running time is reduced from order n to order* $\log n$.
- Vorsicht! wenn der Aufwand für die Multiplikation $x^j \cdot x^k$ proportional zu $j \cdot k$, als ist (Integer- und Polynommultiplikation!), also quadratisch mit der Anzahl der Ziffern- bzw. Koeffizientenoperationen wächst, kann der Aufwand für die "banale" Methode von der gleichen Grössenordnung wir für die "schnelle" Methode sein — oder sogar schlechter!

- Anwendungszenario: Exponentiation in $\mathbb{Z}_m$

$$(a, n, m) \mapsto a^n \bmod m$$

- Maple-Programm (rekursiv)

```
modpower := proc (a,n,m)
if n=0 then RETURN(1)
   else t := modpower(a,iquo(n/2),m)^2; (Q)
   if odd(n) then t := t*a mod m fi;    (M)
   RETURN(t mod n)
   fi;
end;
```

- benötigt (etwa) $\log n$-maliges Quadrieren und höchstens $\log n$ weitere Multiplikationen von $\log m$-bit Zahlen und $\log n$ Reduktionen modulo m von $2 \log m$-bit-Zahlen, also insgesamt einen Aufwand $O\left(\log n \cdot (\log m)^2\right)$ gemessen in bit-Operationen.

- Umkehrabbildung: *diskreter Logarithmus*

$$(a, a^n \bmod m, m) \mapsto n$$

- Hierfür ist bis heute kein effizienter Algorithmus bekannt! Die besten bekannten Algorithmen haben gleiche Komplexität wie die besten Algorithmen für die Faktorisierung von m
- Zahlenbeispiel: a, n, m in der Grössenordnung 10^{200}
  - schnelle Exponentiation erfordert etwa 3000 Multiplikationen von 200-digit-Zahlen und 3000 Reduktionen modulo m von 400-digit-Zahlen
  - die Berechnung des diskreten Logarithmus "brute-force" würde etwa 10^{200} Multiplikationen und Reduktionen modulo m erfordern

- Anwendungsszenario: schnelle Berechung C-rekursiver Folgen
- Kanonisches Beispiel: Fibonacci-Zahlen
  - Idee: mit $F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ gilt

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n-1} + f_{n-2} \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = F \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}$$

  also

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = F^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} = F^{n-2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

  - Die Berechnung von f_n mittels iteriertem Quadrieren der Matrix F benötigt $13 \cdot \lfloor \log(n-2) \rfloor + 12 \cdot \nu(n-2) - 10$ arithmetische Operationen (Additionen, Subtraktionen, Multiplikationen, Divisionen durch 2 , siehe HEUN, GA)
  - Dies kann noch etwas verbessert werden, wenn man ausnützt, dass die Potenzen von F symmetrische Matrizen sind

- Gofer-Programm zur Berechnung der Fibonacci-Zahlen mittels schneller Exponentiation von Matrizen (HEUN, GA)

```
h :: Int -> (Int,Int,Int,Int)
h n | n==1   = (1, 1,
                1, 0)
    | even n = let (a,b,c,d)=h(n/2)
               in ((a*a)+(b*c), (a*b)+(b*d),
                   (c*a)+(d*c), (c*b)+(d*d))
    | odd  n = let (a,b,c,d)=h(n/2)
               in ((a*a)+(b*c)+(a*b)+(b*d), (a*a)+(b*c),
                   (c*a)+(d*c)+(c*b)+(d*d), (c*a)+(d*c))
fib3 :: Int -> Int
fib3 n | n==1 || n==2 =1
       | n>2          = a+c where (a,b,c,d) = h (n-2)
```

  - Siehe Materialien zur Vorlesung für Laufzeitanalysen und gemessene Laufzeiten verschiedener Algorithmen zur Berechnung von Fibonacci-Zahlen
  - Die Methode lässt sich generell für C-rekursive Folgen anwenden und liefert Verfahren logarithmischer (in n) Komplexität

Hinweis:

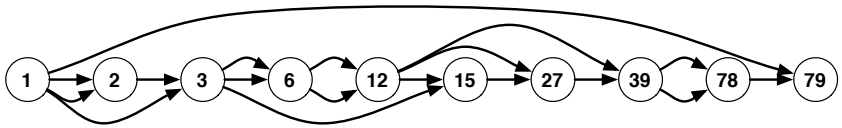
- Das Problem, a^n in einer Halbgruppe möglichst effizient zu berechnen, hat etwas mit dem Problem der sog. *Additionsketten* zu tun.
- Eine Additionskette für $n \in \mathbb{N}$ ist eine Folge

$$1 = a_0, a_1, a_2, \ldots, a_r = n$$

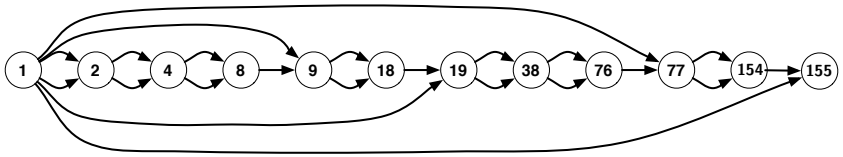
  von ganzen Zahlen mit der Eigenschaft

$$a_i = a_j + a_k \quad \text{wobei } j \leq k < i \quad (1 \leq i \leq r)$$

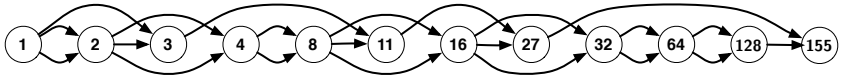
  (straight-line-Programm mit Addition als einziger Operation)



- LR-Additionskette für $n = 155$

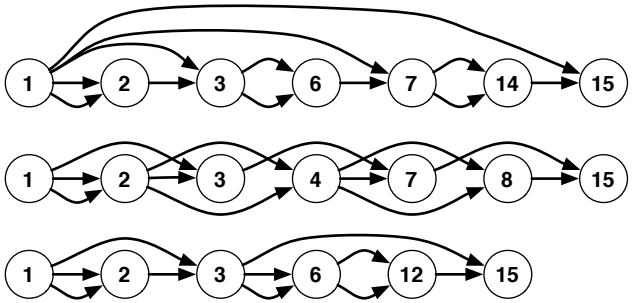


- RL-Additionskette für $n = 155$



Exponentiation
Additionsketten

- Das Problem, die Länge $L(n)$ *kürzester* Additionsketten für gegebenes n zu bestimmen, ist sehr schwierig!
- LR- und RL-Additionsketten sind nicht immer optimal!



- Berühmte Vermutung (Scholz, Brauer, 1937,1939):

$$L(2^n - 1) \leq n - 1 + L(n)$$

- Mehr darüber in ch. 4.6.3 von TAOCP!

Rekursive Definition der Fibonacci-Zahlen

$$f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2} \quad (n \geq 2)$$

Erste Werte

n	0	1	2	3	4	5	6	7	8	9	10	...	25	...
f_n	0	1	1	2	3	5	8	13	21	34	55	...	75025	...

Exakte Formel (de Moivre, 1718)

$$f_n = \frac{\varphi^n - \hat{\varphi}^n}{\sqrt{5}} \quad \text{mit} \begin{cases} \varphi = \frac{1+\sqrt{5}}{2} & \approx 1.61803\ldots \text{ “goldener Schnitt”} \\ \hat{\varphi} = \frac{1-\sqrt{5}}{2} & \approx -0.61803\ldots \end{cases}$$

Bemerkung: der “goldene Schnitt” φ ist die positive Lösung der Gleichung:

$$x^2 = 1 + x$$

$\hat{\varphi} = 1 - \varphi$ ist die negative Lösung dieser Gleichung.

Die Folge $(f_{n+1}/f_n)_{n\geq 1}$ von Quotienten aufeinanderfolgender Fibonacci-Zahlen konvergiert und es gilt

$$\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \varphi$$

Abschätzung mittels goldenem Schnitt

$$\varphi^{n-2} \leq f_n \leq \varphi^{n-1}$$

Alternative Abschätzung (GA:Lemma 1.3)

$$2^{\lfloor \frac{n-1}{2} \rfloor} \leq f_n \leq 2^{n-2} \quad (n > 2)$$

andere interessante Eigenschaften der Fibonacci-Zahlen

- $\mathsf{ggT}(f_m, f_n) = f_{\mathsf{ggT}(m,n)}$
- $f_{n+1} f_{n-1} - f_n^2 = (-1)^n$
- $f_{n+m} = f_m f_{n+1} + f_{m-1} f_n$
- $\varphi^n = f_{n+1} - \hat{\varphi} \cdot f_n$, $\varphi^{n+1} = \varphi \cdot f_{n+1} + f_n$
- $\varphi^n + \hat{\varphi}^n = 2 f_{n+1} - f_n$

Die Fibonacci-Zahlen treten in der Mathematik und Algorithmik (in der Kunst und in der Natur...) an vielen verschiedenen Stellen auf. Wichtig ist der Zusammenhang mit dem Algorithmus von Euklid.

Satz von Lamé (1844):

> Sind $m, n \in \mathbb{N}$ mit $m, n \leq f_k$, so benötigt der Algorithmus von Euklid zur Berechnung von $\mathsf{ggT}(m, n)$ maximal $k + 1$ Divisionsschritte.

Dieses Resultat kann man als die (vermutlich) historisch früheste quantitative Analyse des Laufzeitverhaltens eines Algorithmus ansehen.

Rekursive Berechnung der Fibonacci-Zahlen (gofer)

```
fib1 :: Int -> Int
fib1 1 = 1
fib1 2 = 1
fib1 n = fib1 (n-1) + fib1 (n-2)
```

$C_{rek}(n) =$ Anzahl der arithmetischen Operationen (Additionen, Subtraktionen, Mutliplikationen, Divisionen) zur Berechnung von f_n mit dem Programm fib1

$$C_{rek}(1) = C_{rek}(2) = 0$$
$$C_{rek}(n) = 3 + C_{rek}(n-1) + C_{rek}(n-2) \qquad (n \geq 3)$$

Tatsache (GA Lemma 1.1): Die Rekursion

$$D(1) = d$$
$$D(2) = d$$
$$D(n) = D(n-1) + D(n-2) \ (n \geq 3)$$

hat die Lösung $D(n) = d \cdot f_n \ (n \geq 3)$.

Resultat (GA, Korollar 1.2):

$$C_{rek}(n) = 3\,(f_n - 1)$$

Zahlenbeispiel (GA): $n = 100 \Rightarrow C_{rek}(n) \approx 5 \cdot 10^{14}$

Iterative Berechnung der Fibonacci-Zahlen (gofer)

Idee: Iteration der Beziehung $(f_n, f_{n-1}) = (f_{n-1} + f_{n-2}, f_{n-1})$

```
fib2 n :: Int -> Int
fib2 n = x where (x,y) = g n

g :: Int -> (Int,Int)
g 1 = (1,undefined)
g 2 = (1,1)
g n = (x+y,x) where (x,y) = g (n-1)
```

$C_{iter}(n) =$ Anzahl der arithmetischen Operationen zur Berechnung von f_n mit dem Programm fib2

$$C_{iter}(1) = C_{iter}(2) = 0$$
$$C_{iter}(n) = 2 + C_{iter}(n-1) \qquad (n \geq 3)$$

Lösung der Rekursion (GA, Lemma 1.4): $C_{iter}(n) = 2(n-2) \quad (n \geq 2)$

Schnelle Exponentiation - die Idee:

um für eine Zahl $a = 3$ und einen Exponenten $n = 155$ die Potenz 3^{155} zu berechnen, kann man

- 154 Multiplikationen mit dem Faktor 3 ausführen, oder
- die Binärdarstellung $(n)_2 = 10011011$ geschickt ausnutzen, indem man
  - sukzessive
    3^2, $3^4 = \left(3^2\right)^2$, $3^8 = \left(3^4\right)^2$, $\ldots$, $3^{128} = \left(3^{64}\right)^2$ berechnet, sowie
    $3^3 = 3 \times 3^2$, $3^{11} = 3^3 \times 3^8$, $3^{27} = 3^{11} \times 3^{16}$, $3^{155} = 3^{27} \times 3^{128}$
    (macht 7 Quadrierungen und 4 Multiplikationen)
  - sukzessive
    3^2, $3^4 = \left(3^2\right)^2$, $3^9 = 3 \times \left(3^4\right)^2$, $3^{19} = 3 \times \left(3^9\right)^2$, $3^{38} = \left(3^{19}\right)^2$,
    $3^{77} = 3 \times \left(3^{38}\right)^2$, $3^{155} = 3 \times \left(3^{77}\right)^2$ berechnet (macht ebenfalls 7 Quadrierungen und 4 Multiplikationen)

Exponentiation mittels iteriertem Quadrieren:

Idee: in jeder Halbgruppe H kann man zur Berechnung von $(a,n) \mapsto a^n$ für $a \in H, n \in \mathbb{N}$ die Binärdarstellung des Exponenten n verwenden

$$a^n = \left\{ \begin{array}{ll} \left(a^{\frac{n}{2}}\right)^2 & \text{falls } n \text{ gerade} \\ a \cdot \left(a^{\frac{n-1}{2}}\right)^2 & \text{falls } n \text{ ungerade} \end{array} \right\} = a^{n \pmod 2} \cdot \left(a^{n \div 2}\right)^2$$

Bezeichnungen für $n \in \mathbb{N}$:

- $(n)_2$: Binärdarstellung von n (ohne führende Nullen)
- $\ell(n)$: Länge der Binärdarstellung von $n = \lfloor \log(n) \rfloor + 1$
- $\sharp_1(n)_2$: Anzahl der Einsen in $(n)_2$

rekursives Programm zur Berechnung von $pot : (a,n) \mapsto a^n$ (gofer)

```
pot :: Int -> Int -> Int
pot a n | n == 1 = a
        | even n = p*p   where p = pot a (n/2)
        | odd  n = a*p*p where p = pot a (n/2)
```

Resultat (GA, Theorem 1.7): Die Berechnung von $pot(a,n)$ benötigt $\ell(n) + \sharp_1(n)_2 - 2$ Multiplikationen und $\ell(n) - 1$ Divisionen durch 2

Berechnung der Fibonacci-Zahlen mittels iteriertem Quadrieren

Idee: mit $F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ gilt

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n-1} + f_{n-2} \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = F \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}$$

also ist (Induktion, GA, Lemma 1.5)

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = F^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} = F^{n-2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Folgerung (GA, Theorem 1.8):

Die Berechnung von f_n mittels iteriertem Quadrieren der Matrix F benötigt $13\lfloor \log(n-2) \rfloor + 12\sharp_1(n-2)_2 - 10$ arithmetische Operationen (Additionen, Subtraktionen, Multiplikationen, Divisionen durch 2)

Dies kann noch etwas verbessert werden, wenn man ausnützt, dass die Potenzen von F symmetrische Matrizen sind

```
h :: Int -> (Int,Int,Int,Int)
h n | n==1   = (1, 1,
                1, 0)
    | even n = let (a,b,c,d)=h(n/2)
               in ((a*a)+(b*c), (a*b)+(b*d),
                   (c*a)+(d*c), (c*b)+(d*d))
    | odd  n = let (a,b,c,d)=h(n/2)
               in ((a*a)+(b*c)+(a*b)+(b*d), (a*a)+(b*c),
                   (c*a)+(d*c)+(c*b)+(d*d), (c*a)+(d*c))
fib3 :: Int -> Int
fib3 n | n==1 || n==2 =1
       | n>2          = a+c where (a,b,c,d) = h (n-2)
```


Zur Dauer der Berechnung von Fibonacci-Zahlen

Annahme: Dauer einer arithmetischen Operation = 1 Mikrosekunde

Variante/Zeitbedarf	1ms	1s	1m	1h
rekursiv	≈ 14	≈ 28	≈ 37	≈ 45
iterativ	≈ 500	$\approx 5 \cdot 10^5$	$\approx 3 \cdot 10^7$	$\approx 2 \cdot 10^9$
iteriertes Quadrieren	$\approx 10^{12}$	$\approx 10^{12.000}$	$\approx 10^{700.000}$	$\approx 10^{10^6}$

Da die beteiligten Operanden sehr schnell wachsen, ist obige Annahmen nicht realistisch!

Laufzeitvergleich für drei Methoden zur Berechnung der Fibonacci-Zahlen

(Quelle: Brassard, Bratley, *Algorithmics, Theory and Practice*, Prentice-Hall 1988, Sec. 1.7)

A: Berechnung modulo 7, gemessene Werte, Zeiten $>$ 2 min geschätzt

n	10	20	30	50	10^2	10^4	10^6	10^8
fib1	8ms	1s	2m	21d	10^9y	–	–	–
fib2	1/6ms	1/3ms	1/2ms	3/4ms	3/2ms	150ms	15s	25m
fib3	1/3ms	2/5ms	1/2ms	1/2ms	1/2ms	1ms	3/2ms	2ms

B: Laufzeitverhalten

fib1 : $t_1(n) = \varphi^{n-20}$ sec

fib2 : $t_2(n) = 15\,n \times 10^{-6}$ sec

fib3 : $t_3(n) = \frac{1}{4} \log n \times 10^{-3}$ sec

C: gemessene Werte (in Sekunden) für exakte Berechnung (long integer)

n	5	10	15	20	25	100	500	10^3	$5 \cdot 10^3$	10^4
fib1	0.007	0.087	0.941	10.766	118.457	–	–	–	–	–
fib2	0.005	0.009	0.011	0.017	0.021	0.109	1.177	3.581	76.107	298.892
fib3	0.013	0.017	0.019	0.020	0.021	0.041	0.132	0.348	7.664	29.553

Vorbemerkung: Homorphieprinzip für Ringe

Ringe $\langle R, +_R, *_R, 0_R, 1_R\rangle$ und $\langle S, +_S, *_S, 0_S, 1_S\rangle$
Abbbildung $\Phi : R \to S$ ist *Homomorphismus*, falls $\forall a, b \in R$

$$\begin{aligned}\Phi(a +_R b) &= \Phi(a) +_S \Phi(b)\\ \Phi(a *_R b) &= \Phi(a) *_S \Phi(b)\\ \Phi(0_R) &= 0_S\\ \Phi(1_R) &= 1_S\end{aligned}$$

Dann gilt

$$\Phi(f_R(a, b \ldots, c)) = f_S(\Phi(a), \Phi(b), \ldots, \Phi(c))$$

für alle Funktionen f, die aus $+, *$ durch Komposition entstehen (Ringterme).
Φ ist *Isomorphismus*, falls bijektiv.

Schematische Darstellung (*kommutatives Diagramm*)
für $+$, $*$ und zweistelliges f:

$$\begin{array}{ccc} R \times R & \xrightarrow{\Phi\times\Phi} & S \times S\\ \downarrow{\scriptstyle +_R, *_R, f_R} & & \downarrow{\scriptstyle +_S, *_S, f_S}\\ R & \xrightarrow{\Phi} & S \end{array}$$

Im Folgenden geht es um Homomorphismen

$$\Phi_N : \mathbb{Z} \to \mathbb{Z}_N : a \mapsto a \bmod N$$

bzw.

$$\Phi_{N,M} : \mathbb{Z}_N \to \mathbb{Z}_M : a \mapsto a \bmod M \quad (\text{falls } M|N)$$

Das sind die einzigen Homomorphismen, die die Ringe $\mathbb{Z}$ bzw. $\mathbb{Z}_N$ überhaupt zulasssen!

Aus einem alten indischen Rechenbuch:

- Aus Früchten werden 63 gleich grosse Haufen gelegt, 7 Stück bleiben übrig. Es kommen 23 Reisende, unter denen die Früchte gleichässig verteilt werden, so dass keine übrig bleibt. Wieviele waren es?
- Gesucht ist eine (die kleinste?) natürliche Zahl x mit

$$\begin{aligned} x &\equiv 7 \mod 63\\ x &\equiv 0 \mod 23\end{aligned}$$

- Lösung: $x \equiv 322 \bmod 23 \cdot 63 \equiv 322 \bmod 1449$

Chinesischer Restesatz — einfachste Form

- $p, q \in \mathbb{Z}_{>0}$ mit $\mathrm{ggT}(p, q) = 1$
- Bézout-Koeffizienten $u, v \in \mathbb{Z} : p \cdot u + q \cdot v = 1$
- also $p \cdot u \equiv 1 \bmod q$ und $q \cdot v \equiv 1 \bmod p$
- für $b, c \in \mathbb{Z}$ sei $x = c \cdot p \cdot u + b \cdot q \cdot v$, dann gilt

$$\begin{aligned} x &\equiv b \mod p\\ x &\equiv c \mod q\end{aligned}$$

- für $y \in \mathbb{Z}$ gilt

$$\left\{\begin{aligned} y &\equiv b \bmod p\\ y &\equiv c \bmod q\end{aligned}\right\} \quad \Leftrightarrow \quad x \equiv y \bmod (p \cdot q)$$

d.h. es gibt in $\mathbb{Z}_{p\cdot q}$ genau eine Lösung y der simultanen Kongruenzen $y \equiv b \bmod p$ und $y \equiv c \bmod q$, nämlich $y = x \bmod (p \cdot q)$

Beispiel

- bestimme $x \in \mathbb{Z}$ mit

$$x \equiv 3 \bmod 5 \quad \text{und} \quad x \equiv 2 \bmod 7$$

- berechne mittels erweitertem euklidischen Algorithmus Bézout-Koeffizienten für $(5, 7)$

$$3 \cdot 5 - 2 \cdot 7 = 1$$

- es gilt also

$$3 = 5^{-1} \text{ in } \mathbb{Z}_7 \quad \text{und} \quad -2 = 7^{-1} \text{ in } \mathbb{Z}_5$$

- für

$$x = 3 \cdot 7 \cdot (-2) + 2 \cdot 5 \cdot 3 = -42 + 30 = -12$$

gilt dann

$$x \equiv 3 \bmod 5 \quad \text{und} \quad x \equiv 2 \bmod 7$$

und ebenso für jede Zahl y mit $y \equiv x \bmod (5 \cdot 7)$, also insbesondere auch für $y = 23 \in \mathbb{Z}_{35}$

Aus einem alten chinesischen Rechenbuch:

- Eine Bande von 17 Räubern stahl einen Sack mit Goldstücken. Als sie ihre Beute teilen wollten, blieben 3 Goldstücke übrig. Beim Streit darüber, wer ein Goldstück mehr erhalten sollte, wurde ein Räuber erschlagen. Jetzt blieben bei der Verteilung 10 Goldstücke übrig. Erneut kam es zum Streit, und wieder verlor ein Räuber sein Leben. Jetzt liess sich endlich die Beute gleichmässig verteilen. Wieviele Goldstücke waren mindestens in dem Sack?
- Gesucht ist eine (die kleinste?) natürliche Zahl x mit

$$\begin{aligned} x &\equiv 3 \bmod 17 \\ x &\equiv 10 \bmod 16 \\ x &\equiv 0 \bmod 15 \end{aligned}$$

- Lösung: $x \equiv 3930 \bmod 15 \cdot 16 \cdot 17 \equiv 3930 \bmod 4080$

Chinesischer Restesatz

- Theorem:
  - $m_1, m_2, \ldots, m_k$ paarweise teilerfremden natürlichen Zahlen ("Moduln"), $M = m_1 \cdot m_2 \cdots m_k$
  - Elemente $c_1 \in \mathbb{Z}_{m_1}, c_2 \in \mathbb{Z}_{m_2}, \ldots, c_k \in \mathbb{Z}_{m_k}$
  - Dann gibt es *genau ein* $c \in \mathbb{Z}_M$ mit

$$\begin{aligned} c &\equiv c_1 \bmod m_1 \\ c &\equiv c_2 \bmod m_2 \\ &\vdots \\ c &\equiv c_k \bmod m_k \end{aligned}$$

d.h.

$$c \equiv c_i \bmod m_i \quad (1 \leq i \leq k)$$

- Beweis
  - bestimme Bézout-Koeffizienten $u_i, v_i \in \mathbb{Z}$ für m_i und M/m_i $(1 \leq i \leq k)$:

$$u_i \cdot m_i + v_i \cdot (M/m_i) = 1$$

  - für $x = \sum_{1 \leq i \leq k} c_i \cdot v_i \cdot (M/m_i)$ gilt

$$x \equiv c_i \pmod{m_i} \quad (1 \leq i \leq k)$$

  - für $y \in \mathbb{Z}$ gilt

$$y \equiv c_i \pmod{m_i}\ (1 \leq i \leq k) \quad \Leftrightarrow \quad y \equiv x \pmod{M}$$

d.h. es gibt in $\mathbb{Z}_M$ genau eine Lösung y der simultanen Kongruenzen $y \equiv c_i \bmod m_i$ $(1 \leq i \leq k)$, nämlich $y = x \bmod M$

- Beispiel
  - bestimme $x \in \mathbb{Z}_{5\cdot 7\cdot 11}$ mit

$$x \equiv 3 \bmod 5, \quad x \equiv 1 \bmod 7, \quad x \equiv 7 \bmod 11$$

  - eeA liefert Bézout-Koeffizienten

$$31 \cdot 5 - 2 \cdot 77 = 1, \quad \text{also} \quad (-2) \cdot 77 \equiv \begin{cases} 1 & \bmod 5 \\ 0 & \bmod 7 \\ 0 & \bmod 11 \end{cases}$$

$$8 \cdot 7 - 1 \cdot 55, = 1, \quad \text{also} \quad (-1) \cdot 55 \equiv \begin{cases} 0 & \bmod 5 \\ 1 & \bmod 7 \\ 0 & \bmod 11 \end{cases}$$

$$16 \cdot 11 - 5 \cdot 35 = 1, \quad \text{also} \quad (-5) \cdot 35 \equiv \begin{cases} 0 & \bmod 5 \\ 0 & \bmod 7 \\ 1 & \bmod 11 \end{cases}$$

  - Lösung

$$x = 3\cdot(-2)\cdot 77 + 1\cdot(-1)\cdot 55 + 7\cdot(-5)\cdot 35 = -1742 \equiv 183 \bmod 5\cdot 7\cdot 11$$

Warum funktioniert das?

- die Zahlen $e_i = v_i \cdot (M/m_i)$ $(1 \le i \le k)$ haben die "Indikatoreigenschaft"

$$e_i \equiv \begin{cases} 1 & \bmod m_i \\ 0 & \bmod m_j \quad \text{für } j \neq i \end{cases} \qquad (1 \le i \le k)$$

  Damit kann man Zahlen mit vorgegebenen Kongruenzeigenschaften (= Werten an den Stellen m_i) mittels Linearkombination konstruieren
- für $x = \sum_{1\le i\le k} c_i \cdot e_i$ gilt

$$x \equiv c_i \bmod m_i \quad (1 \le i \le k)$$

Dieses Konstruktionsprinzip ist weit verbreitet. Andere Instanzen sind:

- Interpolationsformel von Lagrange
  - Zu vorgegebenen (paarweise verschiedenen) Stellen $m_1, m_2, \ldots, m_k \in \mathbb{C}$
  - und vorgegebenen Werten $c_1, c_2, \ldots, c_k \in \mathbb{C}$
  - gibt es genau ein Polynom

$$p(X) = \sum_{0\le i<k} p_i X^i \quad \text{mit Grad } \deg p(X) < k,$$

  das an den Stellen m_i die vorgegebenen Werte c_i annimmt:

$$p(m_i) = c_i \quad (1 \le i \le k)$$

- Dieses $p(X)$ kann man angeben

$$p(X) = \sum_{1\le i\le k} c_i \cdot \frac{\prod_{j\neq i}(X - m_j)}{\prod_{j\neq i}(m_i - m_j)}$$

- Für die Polynome

$$e_i(X) = \frac{\prod_{j\neq i}(X - m_j)}{\prod_{j\neq i}(m_i - m_j)}$$

  gilt die Indikatoreigenschaft

$$e_i(m_j) = \delta_{i,j} = \begin{cases} 1 & \text{für } i = j \\ 0 & \text{für } i \neq j \end{cases}$$

- Darstellung BOOLEscher Funktionen in disjunktiver Normalform
  - $\mathbb{B} = \{0,1\}^n$: BOOLEsche Algebra mit $\neg, \vee, \wedge$
  - Jede BOOLEsche Funktion $f : \mathbb{B}^n \to \mathbb{B}$ kann als BOOLEsches Polynom

$$f(X_1, \ldots, X_n) = \bigvee_{\mathbf{b} \in \mathbb{B}^n} f(\mathbf{b}) \cdot e_{\mathbf{b}}(X_1, \ldots, X_n)$$

  dargestellt werden.

- Dabei sind für $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{B}^n$ die "Minterme"

$$e_{\mathbf{b}}(X_1, \ldots, X_n) = \bigwedge_{b_i=1} X_i \wedge \bigwedge_{b_i=0} (\neg X_i)$$

  Funktionen mit

$$e_{\mathbf{b}}(a_1, \ldots, a_n) = \delta_{\mathbf{a},\mathbf{b}} = \begin{cases} 1 & \text{für } (b_1, \ldots, b_n) = (a_1, \ldots, a_n) \\ 0 & \text{für } (b_1, \ldots, b_n) \neq (a_1, \ldots, a_n) \end{cases}$$

  für $\mathbf{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{B}^n$

Algebraische Version des Chinesischen Restesatzes:

- Die Abbildung

$$\begin{aligned} \Psi : \mathbb{Z}_M &\to \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \cdots \times \mathbb{Z}_{m_k} \\ a &\mapsto (a \bmod m_1, a \bmod m_2, \ldots, a \bmod m_k) \end{aligned}$$

  ist ein Isomorphismus von Ringen
- Insbesondere gilt für die invertierbaren Elemente (Einheiten)

$$\Psi(\mathbb{Z}_M^*) = \mathbb{Z}_{m_1}^* \times \mathbb{Z}_{m_2}^* \times \cdots \times \mathbb{Z}_{m_k}^*$$

  (Isomorphismus von Gruppen)
- Bemerkung: aus der Isomorphie der Einheitengruppen folgt

$$\varphi(M) = \varphi(m_1) \cdot \varphi(m_2) \cdots \varphi(m_k)$$

  d.h. die Multiplikativität der EULERschen φ-Funktion

Beispiel
Isomorphismus der Ringe: $\mathbb{Z}_4 \times \mathbb{Z}_9 \simeq \mathbb{Z}_{36}$

	0	1	2	3	4	5	6	7	8	$\mathbb{Z}_9$
0	0	28	20	12	4	32	24	16	8	
1	9	1	29	21	13	5	33	25	17	
2	18	10	2	30	22	14	6	34	26	
3	27	19	11	3	31	23	15	7	35	
$\mathbb{Z}_4$										$\mathbb{Z}_{36}$

Isomorphismus der Einheitengruppen: $\mathbb{Z}_4^* \times \mathbb{Z}_9^* \simeq \mathbb{Z}_{36}^*$

	1	2	4	5	7	8	$\mathbb{Z}_9^*$
1	1	29	13	5	25	17	
3	19	11	31	23	7	35	
$\mathbb{Z}_4^*$							$\mathbb{Z}_{36}^*$

Schema der modularen Arithmetik

$$\begin{array}{ccccccccc} \mathbb{Z}_M & \longrightarrow_{\Psi} & \mathbb{Z}_{m_1} & \times & \mathbb{Z}_{m_2} & \times & \ldots & \times & \mathbb{Z}_{m_k} \\ \downarrow_{\mathrm{op}} & & \downarrow_{\mathrm{op}} & & \downarrow_{\mathrm{op}} & & \ldots & & \downarrow_{\mathrm{op}} \\ \mathbb{Z}_M & \longleftarrow_{\Psi^{-1}} & \mathbb{Z}_{m_1} & \times & \mathbb{Z}_{m_2} & \times & \ldots & \times & \mathbb{Z}_{m_k} \end{array}$$

wobei op $\in \{+, *, \texttt{inverse}\}$

- die Abbildung Ψ ist eine "Auswertungsabbildung"
- ihre Umkehrabbildung Ψ^{-1} ist eine "Interpolationsabbildung"

Beispiel: Addition in $\mathbb{Z}_{36} \simeq \mathbb{Z}_4 \times \mathbb{Z}_9$

$$\begin{array}{ccccc} \mathbb{Z}_{36} & & \mathbb{Z}_4 & \mathbb{Z}_9 \\ (22, 25) & \longrightarrow_{\Psi} & (2, 1) & (4, 7) \\ \downarrow_{+_{36}} & & \downarrow_{+_4} & \downarrow_{+_9} \\ 11 & \longleftarrow_{\Psi^{-1}} & 3 & 2 \end{array}$$

Beispiel: Multiplikation in $\mathbb{Z}_{36} \simeq \mathbb{Z}_4 \times \mathbb{Z}_9$

$$\begin{array}{ccccc} \mathbb{Z}_{36} & & \mathbb{Z}_4 & \mathbb{Z}_9 \\ (22, 25) & \longrightarrow_{\Psi} & (2, 1) & (4, 7) \\ \downarrow_{*_{36}} & & \downarrow_{*_4} & \downarrow_{*_9} \\ 10 & \longleftarrow_{\Psi^{-1}} & 2 & 1 \end{array}$$

Beispiel: Inverse in $\mathbb{Z}_{36}^* \simeq \mathbb{Z}_4^* \times \mathbb{Z}_9^*$

$$\begin{array}{ccccc} \mathbb{Z}_{36}^* & & \mathbb{Z}_4^* & \mathbb{Z}_9^* \\ 25 & \longrightarrow_{\Psi} & 1 & 7 \\ \downarrow_{\mathrm{inv}_{36}} & & \downarrow_{\mathrm{inv}_4} & \downarrow_{\mathrm{inv}_9} \\ 13 & \longleftarrow_{\Psi^{-1}} & 1 & 4 \end{array}$$

Das Schema der modularen Arithmetik

- $$\begin{array}{ccccccccc} \mathbb{Z}_M & \longrightarrow_{\Psi} & \mathbb{Z}_{m_1} & \times & \mathbb{Z}_{m_2} & \times & \ldots & \times & \mathbb{Z}_{m_k} \\ \downarrow_{\mathtt{f}} & & \downarrow_{\mathtt{f}} & & \downarrow_{\mathtt{f}} & & \ldots & & \downarrow_{\mathtt{f}} \\ \mathbb{Z}_M & \longleftarrow_{\Psi^{-1}} & \mathbb{Z}_{m_1} & \times & \mathbb{Z}_{m_2} & \times & \ldots & \times & \mathbb{Z}_{m_k} \end{array}$$
  gilt für alle Funktionen `f`, die mit den Ringoperationen verträglich sind ("Homomorphismen"), d.h. aus $\{+, *, \texttt{inverse}\}$ mittels Komposition entstehen.
- Man kann Berechungen in $\mathbb{Z}$ "modularisiert" ausführen, wenn man M gross genug macht (d.h. genügend viele "kleine" Moduln m_i, z.B. Primzahlen in Maschinenwortgrösse)
- Vorteil: Exaktes Rechnen in Bereichen kontrollierter Grösse, Parallelisierung

Beispiel: Lösung einer linearen Kongruenz

- Bestimme die Lösung von $1193 \cdot x \equiv 367 \bmod 31500$
- $31500 = 2^2 \cdot 3^2 \cdot 5^3 \cdot 7$
- Wähle Moduln $m_1 = 4, m_2 = 9, m_3 = 125, m_4 = 7$
- Die gegebene Kongruenz ist äquivalent zu

$$\begin{array}{rcrl} 1 \cdot x & \equiv & 3 & \bmod 4 \\ 5 \cdot x & \equiv & 7 & \bmod 9 \\ 68 \cdot x & \equiv & 117 & \bmod 125 \\ 3 \cdot x & \equiv & 3 & \bmod 7 \end{array}$$

- Wegen $5^{-1} = 2 \in \mathbb{Z}_9$ und $68^{-1} \equiv 57 \in \mathbb{Z}_{125}$:

$$\begin{array}{rcrcrl} x & \equiv & & & 3 & \bmod 4 \\ x & \equiv & 2 \cdot 7 & \equiv & 2 & \bmod 9 \\ x & \equiv & 57 \cdot 117 & \equiv & 44 & \bmod 125 \\ x & \equiv & & & 1 & \bmod 7 \end{array}$$

- Lösung: $x \equiv 22919 \bmod 31500$

- Beispiel: Berechnung einer Determinanten

$$A = \begin{bmatrix} -82 & -48 & -11 \\ 38 & -7 & 58 \\ -94 & -68 & 14 \end{bmatrix} \qquad \det A = ?$$

- Homomorphieprinzip: Determinanten sind 'Ringterme', deshalb gilt

$$(\det_{\mathbb{Z}} A) \bmod m = \det_{\mathbb{Z}_m}(A \bmod m)$$

und für $M = m_1 \cdot m_2 \cdot m_k$

$$(\det_{\mathbb{Z}_M}(A \bmod M)) \bmod m_i = \det_{\mathbb{Z}_{m_i}}(A \bmod m_i) \quad (1 \le i \le k)$$

Homomorphieprinzip: Determinanten sind Ringterme!

- $\mathbb{Z} \to \mathbb{Z}_m : a \mapsto a \bmod m$

$$\begin{array}{ccc} A & \xrightarrow{\bmod m} & A \bmod m \\ \det_{\mathbb{Z}} \downarrow & & \downarrow \det_{\mathbb{Z}_m} \\ \det(A) & \xrightarrow{\bmod m} & \det(A \bmod m) \end{array}$$

- $\mathbb{Z}_M \to \mathbb{Z}_{m_i} : a \mapsto a \bmod m_i \quad (M = m_1 \cdots m_k)$

$$\begin{array}{ccc} A & \xrightarrow{\bmod m_i} & A \bmod m_i \\ \det_{\mathbb{Z}_M} \downarrow & & \downarrow \det_{\mathbb{Z}_{m_i}} \\ \det(A) & \xrightarrow{\bmod m_i} & \det(A \bmod m_i) \end{array}$$

- Moduln :
  $m_1 = 29, m_2 = 31, m_3 = 37, m_4 = 41$
  $M = m_1 \cdot m_2 \cdot m_3 \cdot m_4 = 1363783$
- Berechungen in den einzelnen $\mathbb{Z}_{m_i}$ $(1 \le i \le 4)$
  - $m_1 = 29$

$$A \bmod 29 = \begin{bmatrix} 5 & 10 & 18 \\ 9 & 22 & 0 \\ 22 & 19 & 14 \end{bmatrix}$$

$$\det(A \bmod 29) = (\det A) \bmod 29 = 11$$

  - $m_2 = 31$

$$A \bmod 31 = \begin{bmatrix} 11 & 14 & 20 \\ 7 & 24 & 27 \\ 30 & 25 & 14 \end{bmatrix}$$

$$\det(A \bmod 31) = (\det A) \bmod 31 = 20$$

- $m_3 = 37$

$$A \bmod 37 = \begin{bmatrix} 29 & 26 & 26 \\ 1 & 30 & 21 \\ 17 & 6 & 14 \end{bmatrix}$$

$$\det(A \bmod 37) = (\det A) \bmod 37 = 11$$

- $m_4 = 41$

$$A \bmod 41 = \begin{bmatrix} 0 & 34 & 30 \\ 38 & 34 & 17 \\ 29 & 14 & 14 \end{bmatrix}$$

$$\det(A \bmod 41) = (\det A) \bmod 41 = 19$$

- Modulares Schema

$$\begin{array}{ccccccc} \mathbb{Z}_{29\cdot 31\cdot 37\cdot 41} & & & \mathbb{Z}_{29} & \mathbb{Z}_{31} & \mathbb{Z}_{37} & \mathbb{Z}_{41} \\ A & \longrightarrow_{\Psi} & & A \bmod 29 & A \bmod 31 & A \bmod 37 & A \bmod 41 \\ \downarrow_{\det_M} & & & \downarrow_{\det_{29}} & \downarrow_{\det_{31}} & \downarrow_{\det_{37}} & \downarrow_{\det_{41}} \\ 7522 & \longleftarrow_{\Psi^{-1}} & & 11 & 20 & 11 & 19 \end{array}$$

- Dieses Schema zeigt

$$\det A \equiv 7522 \bmod M$$

- Tatsächlich gilt sogar

$$\det A = 7522$$

Das kann man folgern, wenn man weiss, dass $0 \leq \det A < M$ oder $-(M/2) \leq \det A \leq M/2$ ist
(z.B. mittels der Ungleichung von HADAMARD für Determinanten)

Eine andere Sicht der modularen Arithmetik:

- Gegeben teilerfremde Moduln $m_1, m_2, \ldots, m_k$, $M := m_1 \cdot m_2 \cdots m_k$
- Aus den Bézout-Beziehungen

$$u_i \cdot m_i + v_i \cdot \frac{M}{m_i} = 1 \quad (1 \leq i \leq k)$$

hat man $v_i = (M/m_i)^{-1} \in \mathbb{Z}_{m_i}$ und $e_i \equiv v_i \cdot (M/m_i) \in \mathbb{Z}_M$.

- Seien nun $a \in \mathbb{Z}$ sowie $a_i \in \mathbb{Z}_{m_i} (1 \leq i \leq k)$ mit

$$a \equiv a_1 \cdot e_1 + a_2 \cdot e_2 + \cdots + a_k \cdot e_k \bmod M$$

- Anders formuliert: es gibt ein $a_0 \in \mathbb{Z}$ mit

$$a = a_0 \cdot M + a_1 \cdot v_1 \cdot \frac{M}{m_1} + a_2 \cdot v_2 \cdot \frac{M}{m_2} + \cdots + a_k \cdot v_k \cdot \frac{M}{m_k}$$

- Division durch M ergibt

$$\frac{a}{M} = a_0 + \frac{a_1 \cdot v_1}{m_1} + \frac{a_2 \cdot v_2}{m_2} + \cdots + \frac{a_k \cdot v_k}{m_k}$$

- Indem man jetzt noch Division mit Rest in den Brüchen macht, erhält man eine Darstellung

$$\frac{a}{M} = \alpha_0 + \frac{\alpha_1}{m_1} + \frac{\alpha_2}{m_2} + \cdots + \frac{\alpha_k}{m_k}$$

mit $\alpha_0 \in \mathbb{Z}$ und $\alpha_i \in \mathbb{Z}_{m_i}$ $(1 \leq i \leq k)$

- Diese "modulare" Darstellung (*Partialbruchdarstellung*) der rationalen Zahl a/M ist eindeutig!
- Solche modularen Darstellungen verwendet man vorteilhaft beim exakten Rechnen mit rationalen Zahlen, die grosse Nenner haben.

Beispiel:

- $a = 20853, M = 5544 = 7 \cdot 8 \cdot 9 \cdot 11$
- Mit $m_1 = 7, m_2 = 8, m_3 = 9, m_4 = 11$ ergibt sich

$$\begin{array}{lll} v_1 = (8*9*11)^{-1} \bmod 7 = 1 & a \bmod 7 = 1 & a \cdot v_1 \bmod 7 = 1 \\ v_2 = (7*9*11)^{-1} \bmod 8 = 5 & a \bmod 8 = 5 & a \cdot v_2 \bmod 8 = 1 \\ v_3 = (7*8*11)^{-1} \bmod 9 = 7 & a \bmod 9 = 8 & a \cdot v_3 \bmod 9 = 2 \\ v_4 = (7*8*9)^{-1} \bmod 11 = 5 & a \bmod 11 = 9 & a \cdot v_4 \bmod 11 = 1 \end{array}$$

- Darstellung

$$\frac{a}{M} = \frac{20853}{5544} = 3 + \frac{3221}{5544} = 3 + \frac{1}{7} + \frac{1}{8} + \frac{2}{9} + \frac{1}{11}$$

Partialbruchzerlegung für Polynome

- Betrachten Polynome in der Variablen X über einem Körper k
- $M(X) \in k[X]$ ein normiertes Polynom
- $M(X) = m_1(X) \cdot m_2(X) \cdots m_k(X)$
  Zerlegung von $M(X)$ in normierte teilerfremde Faktoren
- Für jedes Polynom $a(X) \in k[X]$ hat die rationale Funktion $a(X)/M(X)$ genau eine Darstellung

$$\frac{a(X)}{M(X)} = \alpha_0(X) + \frac{\alpha_1(X)}{m_1(X)} + \frac{\alpha_2(X)}{m_2(X)} + \cdots + \frac{\alpha_k(X)}{m_k(X)}$$

mit $\alpha_0(X), \ldots, \alpha_k(X) \in k[X]$ und
$\deg \alpha_i(X) < \deg m_i(X)$ $(1 \le i \le k)$

- Modulare Arithmetik lässt sich sehr effizient z.B. für die exakte Lösung von ganzzahligen Gleichungssystemen mit sehr grossen Koeffizienten einsetzen
- Modulare Arithmetik lässt sich sehr effizient für das exakte Rechnen mit "grossen" rationalen Zahlen einsetzen
- Die Prinzipien der modularen Arithmetik für $\mathbb{Z}$ übertragen sich wörtlich auf Polynomringe $k[X]$.
- Literaturhinweise
  - J. D. Lipson,
    Elements of Algebra and Algebraic Computing,
    Addison-Wesley, Kapitel 8.
  - D. E. Knuth,
    The Art of Computer Programming, vol. 2, (Seminumerical Algorithms), Addison-Wesley, Abschnitt 4.3.2.
  - J. v.z. Gathen, J. Gerhard,
    Modern Computer Algebra,
    Cambridge University Press, Kap. 5.

Primzahltest-Beispiel

- $\mathbb{Z}_N$ ist ein Körper $\Leftrightarrow$ N ist Primzahl
- Allgemein gilt: ein Polynom k-ten Grades hat in einem Körper höchstens k Nullstellen.
- Die Gleichung $X^2 = 1$ hat in einem Körper genau zwei Nullstellen, nämlich ± 1
  - Sonderfall: im Körper $\mathbb{Z}_2$ hat $X^2 = 1$ nur die Nullstelle 1, aber die zählt doppelt (wegen $+1 = -1$)
- Beobachtet man in einem $\mathbb{Z}_N$ eine Zahl $1 < a < N - 1$ mit $a^2 = 1$, so ist N keine Primzahl!
- Beispiele:
  - $N = 8$: in $\mathbb{Z}_8$ gilt $3^2 = 5^2 = 1$, die Gleichung $X^2 = 1$ hat 4 Lösungen.
  - $N = 21$: in $\mathbb{Z}_{21}$ gilt $8^2 = 13^2 = 1$, die Gleichung $X^2 = 1$ hat 4 Lösungen.

$N = 1105 = 5 \cdot 13 \cdot 17$
EEA:

$$1 \cdot 13 \cdot 17 - 44 \cdot 5 = 1 \quad \Rightarrow \quad e_5 = 221 \equiv \begin{cases} 1 & \mod 5 \\ 0 & \mod 13 \\ 0 & \mod 17 \end{cases}$$

$$2 \cdot 5 \cdot 17 - 13 \cdot 13 = 1 \quad \Rightarrow \quad e_{13} = 170 \equiv \begin{cases} 0 & \mod 5 \\ 1 & \mod 13 \\ 0 & \mod 17 \end{cases}$$

$$(-6) \cdot 5 \cdot 13 + 23 \cdot 17 = 1 \quad \Rightarrow \quad e_{17} = 715 \equiv \begin{cases} 0 & \mod 5 \\ 0 & \mod 13 \\ 1 & \mod 17 \end{cases}$$

Die Gleichung $X^2 = 1$ hat in $\mathbb{Z}_{1105} \equiv \mathbb{Z}_5 \times \mathbb{Z}_{13} \times \mathbb{Z}_{17}$ insgesamt $2^3 = 8$ Lösungen ($\mathbb{Z}_5, \mathbb{Z}_{13}, \mathbb{Z}_{17}$ sind Körper).

$\mathbb{Z}_5 \times \mathbb{Z}_{13} \times \mathbb{Z}_{17}$		$\mathbb{Z}_{1105}$
$(+1,+1,+1) = (1,1,1)$	$\leftrightarrow$	$e_5 + e_{13} + e_{17} = 1$
$(+1,+1,-1) = (1,1,16)$	$\leftrightarrow$	$e_5 + e_{13} - e_{17} = 781$
$(+1,-1,+1) = (4,12,1)$	$\leftrightarrow$	$e_5 - e_{13} + e_{17} = 766$
$(-1,+1,+1) = (4,1,1)$	$\leftrightarrow$	$-e_5 + e_{13} + e_{17} = 664$
$(+1,-1,-1) = (1,12,16)$	$\leftrightarrow$	$e_5 - e_{13} - e_{17} = 441$
$(-1,+1,-1) = (4,1,16)$	$\leftrightarrow$	$-e_5 + e_{13} - e_{17} = 339$
$(-1,-1,+1) = (4,12,1)$	$\leftrightarrow$	$-e_5 - e_{13} + e_{17} = 324$
$(-1,-1,+1) = (4,12,16)$	$\leftrightarrow$	$-e_5 - e_{13} - e_{17} = 1104$

MILLER-RABIN-Test

- $N - 1 = 2^t \cdot u$ mit $t \geq 1$ und u ungerade
- Wähle $a \in \mathbb{Z}_N$ mit $\text{ggT}(a, N) = 1$
- Berechne durch fortgesetztes Quadrieren in $\mathbb{Z}_N$:

$$a^u \mapsto a^{2 \cdot u} \mapsto a^{4 \cdot u} \mapsto \cdots \mapsto a^{2^t \cdot u} = a^{N-1}$$

- Falls in dieser Folge die 1 nicht auftaucht, ist N keine Primzahl (FERMAT!)
- Falls die 1 auftaucht mit Vorgänger $\neq -1$, ist N keine Primzahl ($\mathbb{Z}_N$ ist kein Körper)
- Im Erfolgsfalle: a ist "Zeuge" für die Nicht-Primheit von N
- MILLER-RABIN: falls N nicht prim, sind solche Zeugen häufig!

Beispiel: $N = 1105, a = 2$

- $N - 1 = 1104 = 2^4 \cdot 69$, also $t = 4, u = 69$
- Ordnungen

$$\begin{array}{l} ord_5(2) = 4 = \phi(5) \\ ord_{13}(2) = 12 = \phi(13) \\ ord_{17}(5) = 8 \,|\, 16 = \phi(17) \end{array} \quad \Rightarrow ord_{1105}(2) = \text{kgV}(4, 12, 8) = 24$$

- Berechnung von 2^{69} in $\mathbb{Z}_{1105}$ bei Kenntnis der Faktorisierung (!)

$$\begin{array}{ll} \mathbb{Z}_5 & : 2^{69} = 2^1 = 2 \\ \mathbb{Z}_{13} & : 2^{69} = 2^9 = 5 \\ \mathbb{Z}_{17} & : 2^{69} = 2^5 = 15 \end{array} \quad \Rightarrow \quad \begin{array}{l} 2^{69} = 2 \cdot e_5 + 5 \cdot e_{13} - 2 \cdot e_{17} \\ \quad = -138 = 967 \in \mathbb{Z}_{1005} \end{array}$$

- Test durch Quadrieren

$$\begin{array}{rcl} \mathbb{Z}_5 \times \mathbb{Z}_{13} \times \mathbb{Z}_{17} & & \mathbb{Z}_{1105} \\ (2^{69}, 2^{69}, 2^{69}) = (2, 5, 15) & \leftrightarrow & 2 \cdot e_5 + 5 \cdot e_{13} - 2 \cdot e_{17} = 967 \\ (2^2, 5^2, 5^2) = (4, 12, 4) & \leftrightarrow & 4 \cdot e_5 - e_{13} + 4 \cdot e_{17} = 259 \\ (4^2, 12^2, 4^2) = (1, 1, 16) & \leftrightarrow & e_5 + e_{13} - e_{17} = 781 \\ (1^2, 1^2, 16^2) = (1, 1, 1) & \leftrightarrow & e_5 + e_{13} + e_{17} = 1 \end{array}$$

- Fazit: $a = 2$ bezeugt, dass $N = 1105$ keine Primzahl ist, denn

$$2^{4 \cdot 69} = 781 \neq 1 \quad \text{und} \quad 2^{8 \cdot 69} = \left(2^{69}\right)^2 = 1 \quad \text{in } \mathbb{Z}_{1105}$$

- Aus einem alten indischen Lehrbuch der Astronomie und Mathematik:
  - Man bestimme die kleinste positive Zahl, die bei Division durch 3,4,5 und 6 die Reste 2,3,4, bzw. 5 lässt.
  - Lösung: $x = 59$
- Aus FIBONACCIS *Liber abbaci*:
  - Man bestimme die kleinste positive Zahl, die bei Division durch 2,3,4,5,6 jeweils den Rest 1 lässt und durch 7 teilbar ist
  - Lösung: $x = 301$
- Die Moduln sind nicht mehr teilerfremd!

Was tun, wenn die Moduln nicht teilerfremd sind?

- Wegen

$$n \mid (x - a) \wedge d \mid n \quad \Rightarrow \quad d \mid (x - a)$$

gilt

$$x \equiv a \bmod n \wedge d \mid n \quad \Rightarrow \quad x \equiv a \bmod d$$

und somit:

$$\begin{cases} x \equiv a \bmod m \\ x \equiv b \bmod n \end{cases} \text{ lösbar} \quad \Leftrightarrow \quad \text{ggT}(m, n) \mid a - b$$

- Allgemein für beliebige Moduln m_i $(1 \leq i \leq k)$

$$\left.\begin{array}{c} x \equiv a_i \bmod m_i \\ (1 \leq i \leq k) \end{array}\right\} \text{ lösbar} \Leftrightarrow \text{ggT}(m_i, m_j) \mid a_i - a_j \ (1 \leq i < j \leq k)$$

- Die Lösung ist eindeutig modulo $\text{kgV}(m_1, m_2, \ldots, m_k)$.

Beispiele

- Lösung von

$$\begin{cases} x \equiv 3 \bmod 6 \\ x \equiv 7 \bmod 8 \end{cases}$$

existiert wegen $\text{ggT}(6, 8) = 2 \mid (3 - 7)$.

$$\begin{cases} x \equiv 3 \bmod 6 \\ x \equiv 7 \bmod 8 \end{cases} \Leftrightarrow \begin{cases} x \equiv 1 \bmod 2 \\ x \equiv 0 \bmod 3 \\ x \equiv 7 \bmod 8 \end{cases} \Leftrightarrow \begin{cases} x \equiv 0 \bmod 3 \\ x \equiv 7 \bmod 8 \end{cases} \Leftrightarrow x \equiv 15 \bmod 24$$

- Lösung von

$$\begin{cases} x \equiv 7 \bmod 9 \\ x \equiv 2 \bmod 12 \end{cases}$$

existiert nicht wegen $\text{ggT}(9, 12) = 3 \not\mid (7 - 2)$.
Genauer:

$$\begin{array}{rcl} x \equiv 7 \bmod 9 & \Rightarrow & x \equiv 1 \bmod 3 \\ x \equiv 2 \bmod 12 & \Rightarrow & x \equiv 2 \bmod 3 \end{array}$$

Ergänzende Bemerkung zur Algebra:

- Sind $p, q \in \mathbb{Z}_{>0}$ beliebig, so gibt es einen Isomorphismus von Ringen
$$\mathbb{Z}_p \times \mathbb{Z}_q \;\simeq\; \mathbb{Z}_{\mathtt{ggT}(p,q)} \times \mathbb{Z}_{\mathtt{kgV}(p,q)}$$
- Jede endliche kommutative Gruppe $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \ldots \times \mathbb{Z}_{m_k}$ ist isomorph zu genau einer Gruppe
$$\mathbb{Z}_{d_1} \times \mathbb{Z}_{d_2} \times \ldots \times \mathbb{Z}_{d_\ell} \quad \text{mit} \quad 2 \leq d_1 \,|\, d_2 \,|\, \ldots \,|\, d_\ell$$
(Elementarteilersatz)
- Isomorphie von endlichen kommutativen Gruppen
$$\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \ldots \times \mathbb{Z}_{m_k} \overset{?}{\simeq} \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \ldots \times \mathbb{Z}_{n_\ell}$$
kann man mittels lokaler Transformationen $(p, q) \mapsto (\mathtt{ggT}(p, q), \mathtt{kgV}(p, q))$ effizient entscheiden, indem man die Indexfolgen $(m_1, \ldots, m_k)$ und $(n_1, \ldots, n_\ell)$ in Elementarteilerform transformiert.

Die Primheit von Primzahlen kann man effizient *verifizieren*

oder

$\textsc{Primes} \in \mathbb{NP}$

- Die Zahl

  $n = 114381625757888867669235779976146612010218296721242$
  $362562561842935706935245733897830597123563958705$
  $058989075147599290026879543541$

  ist *keine* Primzahl.
- Klar! Denn $n = p \cdot q$ mit

  $p = 349052951084765094914784961990389813341776463849$
  87843990820577

  $q = 327691329932667095499619881908344614131776429679$
  942539798288533

- p und q *sind* Primzahlen! Aber wie garantiert man das?
- n ist Kryptologen bekannt als RSA-129.

Problema, numeros primos a compositis dignoscendi . . . ad gravissima ac utilissima tabus arithmeticae pertinere. . .
. . . scientiae dignitas requirere videtur, ut omnia subsidia ad solutionem problematis tam elegantis ac celebris sedulo excolantur.

J. C. F. Gauss (1777–1855),
Disquisitiones Arithmeticae, Artikel 329

aus der Übersetzung von Maser (1889):

Dass die Aufgabe, die Primzahlen von den zusammengesetzten zu unterscheiden und letztere in ihre Primfactoren zu zerlegen, zu den wichtigsten und nützlichsten der gesamten Arithmetik gehört und die Bemühungen und den Scharfsinn sowohl der alten als auch der neueren Geometer in Anspruch genommen hat, ist so bekannt, dass es überflüssig wäre, hierüber viele Worte zu verlieren. [. . .] ausserdem dürfte es die Würde der Wissenschaft erheischen, alle Hülfsmittel zur Lösung jenes so eleganten und berühmten Problems fleissig zu vervollkommnen.

Trotzdem muss man gestehen, dass alle bisher angegebenen Methoden entweder auf sehr specielle Fälle beschänkt oder so mühsam und weitläufig sind, dass sie [. . .] auf grössere Zahlen aber meistenteils kaum angewendet werden können.

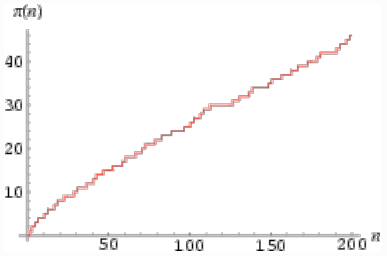
K. Gödel (Brief an J. von Neumann, 1956)

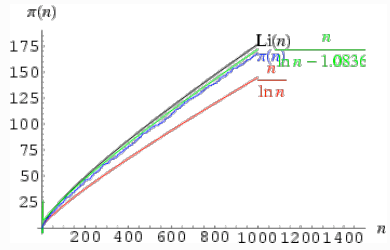
Wenn es eine Maschine mit ... gäbe, hätte das Folgerungen von der grössten Tragweite. Es würde offenbar bedeuten, dass man trotz der Unlösbarkeit des Entscheidungsproblems die Denkarbeit der Mathematiker bei ja-oder-nein-Fragen vollständig (abgesehen von der Aufstellung der Axiome) durch Maschinen ersetzen könnte.
... bedeutet, dass die Anzahl der Schritte gegenüber dem blossen Probieren von N auf $\log N$ *verringert werden kann. So starke Verringerungen kommen aber bei anderen finiten Problemen durchaus vor, z.B. bei der Berechnung eines quadratischen Restsymbols durch wiederholte Anwendung des Reziprozitätsgesetzes. Es wäre interessant zu wissen, wie es damit z.B. bei der Feststellung, ob eine Zahl Primzahl ist, steht und wie stark im allgemeinen bei finiten kombinatorischen Problemen die Anzahl der Schritte gegenüber dem blossen Probieren verringert werden kann.*

Häufigkeit von Primzahlen

- Primzahlsatz (von Gauss vermutet, erst viel später von Hadamard und de la Vallée-Poussin bewiesen)

$$\pi(n) = \sharp\{p \leq n;\ p \text{ Primzahl}\} \sim \frac{n}{\ln n}$$





Primzahltest mittels Probedivision

- Kriterium:

$$n \text{ ist Primzahl} \quad \Leftrightarrow \quad \forall_{a:2\leq a\leq\sqrt{n}}\ a \not| n$$

- Logarithmisches Kostenmodell:
  - Input-Grösse ist $\log n$
  - Anzahl der Probedivisionen: $O(\sqrt{n}) = O(2^{\frac{1}{2}\log n})$
  - Jede Probedivision erfordert $O(\log^2 n)$ Bit-Operationen
  - Laufzeit für m-stelliges n ist im worst-case $O(m^2 \cdot 2^{m/2})$
  - Exponentieller Aufwand in Problemgrösse $m = \log n$ — prohibitiv!
  - NB: man kann sich bei den Probeteilern a auf Primzahlen beschränken.

Konkreter:

- Das Testen einer Zahl $n \sim 10^k$ erfordert etwa $\dfrac{10^{k/2}}{(k/2)\log 10}$ Probedivisionen
- Annahme: man kann pro Sekunde 10^6 Divisionen ausführen.
- Dann benötigt ein Primzahltest per Probedivision für eine Zahl mit k Dezimalstellen etwa

$$\frac{10^{k/2}}{(k/2)\log 10 \times 10^6 \times 60 \times 60 \times 24 \times 356} = \frac{10^{k/2}}{k} \times 2.75 \times 10^{-14} \text{ Jahre}$$

- Einige Zahlenbeispiele:

k=30	$\Rightarrow$	11 Monate
k=50	$\Rightarrow$	5.5×10^9 Jahre
k=100	$\Rightarrow$	2.75×10^{34} Jahre

Entscheidung versus Verifikation

- Beispiele:

  $2^{67}-1 \in \text{PRIM}$? $\quad 2^{128}+1 \in \text{PRIM}$? $\quad 2^{858433}-1 \in \text{PRIM}$?

- Zusammengesetztheit hat effiziente Zeugen (z.B. Teiler)

$$\begin{aligned} 2^{67}-1 &= 147.573.952.589.676.412.927 \\ &= 761.838.257.287 \cdot 193.707.721 \end{aligned}$$

  [Frank COLE (1903) benötigte "die Sonntage dreier Jahre"...]

  Es ginge in diesem Fall auch (FERMAT)

$$3^{2^{67}-2} \pmod{2^{67}-1} = 95.591.506.202.441.271.281$$

  aber das geht nicht immer!

- Wie kann man jemanden effizient davon überzeugen, dass eine vorgelegte (sehr grosse) Zahl N *keine Primzahl* ist?
  - Man nimmt zwei geeignete Zahlen P, Q, multipliziert sie und überprüft:

$$N \stackrel{?}{=} P \times Q$$

  - Teiler einer Zahl sind "Zeugen" dafür, dass die Zahl nicht Primzahl ist.
  - Analog kann man ggT-Berechnungen verwenden.
  - Es wird nicht verlangt, dass Zeugen leicht zu finden sind!
  - Fazit: Nicht-Primzahlen lassen sich effizient verifizieren!

- Wie kann man jemanden effizient davon überzeugen, dass eine vorgelegte (sehr grosse) Zahl N *eine Primzahl* ist?
  Gibt es "Zeugen" dafür, dass eine Zahl keine echten Teiler besitzt?
  Lassen sich Primzahlen effizient verifizieren?
  - Ja! Aber das ist keineswegs offensichtlich (PRATT, 1975).
- Grundsätzliche Bemerkung:
  Per def. ist Primheit eine "universelle" und Nicht-Primheit eine "existentielle" Eigenschaft von Zahlen.

Körper und Ordnungen

- $\mathbb{F}$ Körper, $\mathbb{F}^* = \mathbb{F} \setminus \{0\}$: multiplikative Gruppe
- Fakt: In einem Körper $\mathbb{F}$ hat ein Polynom vom Grad k höchstens k Nullstellen (Vielfachheiten mitgezählt)
- Für $a \in \mathbb{F}^*$: $ord_{\mathbb{F}^*}(a)$ Ordnung von a in $\mathbb{F}^*$

$$ord_{\mathbb{F}^*}(a) = \sharp\langle a \rangle = \min\{t \geq 1\,;\, a^t = 1\}$$

- Für $a \in \mathbb{F}^*, n \geq 1$:

$$ord_{\mathbb{F}^*}(a) \mid n \quad \Leftrightarrow \quad a \text{ ist Nullstelle von } X^n - 1$$

- Folgerung: In einem Körper $\mathbb{F}$ gibt es höchstens n Elemente $a \in \mathbb{F}$ mit $ord_{\mathbb{F}^*}(a) \mid n$ $(n \geq 1)$.

- Lemma: Für $n \geq 1$ ist die Anzahl der Elemente $a \in \mathbb{F}$ mit $ord_{\mathbb{F}^*}(a) = n$ entweder $= 0$ oder $= \varphi(n)$.
  - Beweis: sei $a \in \mathbb{F}^*$ mit $ord_{\mathbb{F}^*}(a) = n$.
    Für $a^k \in \langle a \rangle$ $k \in \mathbb{Z}_n$ gilt $(a^k)^n = (a^n)^k = 1^k = 1$,
    dies sind genau n verschiedene Nullstellen von $X^n - 1$.
    Weitere kann es nicht geben. Es gilt

$$ord_{\mathbb{F}^*}(a^k) = \frac{n}{\mathrm{ggT}(k,n)},$$
$$ord_{\mathbb{F}^*}(a^k) = n \text{ für } k \in \mathbb{Z}_n^*.$$

- Satz: Jede *endliche* Untergruppe G von $\mathbb{F}^*$ ist zyklisch.
  - Beweis: Sei $\sharp G = n$, also (LAGRANGE!) $ord_{\mathbb{F}^*}(a) \mid n$ für alle $a \in G$. Für $d \mid n$ sei

$$\chi(d) = \begin{cases} 1 & \text{es gibt ein } a \in G \text{ mit } ord_{\mathbb{F}^*}(a) = d \\ 0 & \text{sonst} \end{cases}$$

  Dann ist

$$n = \sum_{d \mid n} \chi(d) \cdot \varphi(d) \leq \sum_{d \mid n} \varphi(d) = n$$

  Also gilt Gleichheit und somit $\chi(d) = 1$ für alle $d \mid n$.
  Insbesondere ist $\chi(n) = 1$, d.h. es gibt Elemente $a \in G$ mit $ord_{\mathbb{F}^*}(a) = n$.

- Folgerung: $\mathbb{F}$ endlicher Körper $\Rightarrow \mathbb{F}^*$ zyklische Gruppe.
- Die $a \in \mathbb{F}^*$ mit $\langle a \rangle = \mathbb{F}^*$ nennt man *primitive Elemente* von $\mathbb{F}$.
- Es ist nicht klar, wie man primitive Elemente tatsächlich findet — ausser durch Probieren....

Beispiel:

- $N = 13$, $\varphi(\varphi(13)) = \varphi(12) = 4$
  primitive Elemente $2, 6, 7, 11$
- Ordnungen in $\mathbb{Z}_{13}^*$

a	1	2	3	4	5	6	7	8	9	10	11	12
$ord_{13}(a)$	1	12	3	6	4	12	12	4	3	6	12	2

- 2 als primitives Element von $\mathbb{Z}_{13}^*$

k	0	1	2	3	4	5	6	7	8	9	10	11
2^k	1	2	4	8	3	6	12	11	9	5	10	7
$ord_{13}(2^k)$	1	12	6	4	3	12	2	12	3	4	6	12
$\mathrm{ggT}(k, 12)$	12	1	2	3	4	1	6	1	4	3	2	1

Beispiel:

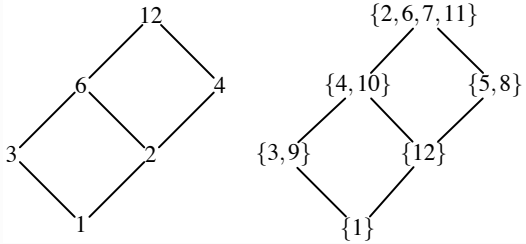
- $N = 17$, $\varphi(\varphi(17)) = \varphi(16) = 8$
  primitive Elemente $3, 5, 6, 7, 10, 11, 12, 14$
- Ordnungen in $\mathbb{Z}_{17}^*$

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$ord_{17}(a)$	1	8	16	4	16	16	16	8	8	16	16	16	4	16	8	2

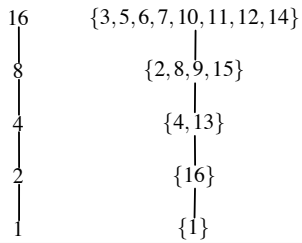
- 7 als primitives Element von $\mathbb{Z}_{17}^*$

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7^k	1	7	15	3	4	11	9	12	16	10	2	14	13	6	8	5
$ord_{17}(7^k)$	1	16	8	16	4	16	8	16	2	16	8	16	4	16	8	16
$\text{ggt}(k, 16)$	1	1	2	1	4	1	2	1	8	1	2	1	4	1	2	1

Struktur von $\mathbb{Z}_{13}^*$



Struktur von $\mathbb{Z}_{17}^*$



Endliche Körper

- Ist $\mathbb{F}$ ein endlicher Körper, so gibt es eine eindeutig bestimmte Primzahl p mit $p \cdot 1 = \underbrace{1 + 1 + \cdots + 1}_{p} = 0$.
  Dieses p nennt man die "Charakteristik" von $\mathbb{F}$.
- Ist $\mathbb{F}$ ein endlicher Körper der Charakterisitk p, so hat $\mathbb{F}$ p^n Elemente für ein $n >= 1$.
- Zu jeder Primzahl p und jedem $n \geq 1$ existiert ein Körper mit p^n Elementen.
- Alle Körper mit p^n Elementen sind "isomorph".
- Speziell für $n = 1$: diese Körper sind die $\mathbb{Z}_p$.
- Körper mit p^n Elementen ($n > 1$) entstehen aus $\mathbb{Z}_p$ durch "algebraische Erweiterung".
  (Sie haben nichts mit $\mathbb{Z}_{p^n}$ zu tun!)

Primzahlkriterium

- Für $n \in \mathbb{N}$ gilt:

$$n \text{ ist Primzahl} \quad \Leftrightarrow \quad \exists a \in \mathbb{Z}_n^* : ord_n(a) = n - 1$$

- Beweis:
  - $\Rightarrow$: n Primzahl $\rightarrow$ $\mathbb{Z}_n$ Körper $\rightarrow$ $\mathbb{Z}_n^*$ zyklisch mit $\sharp\mathbb{Z}_n^* = n - 1$.
  - $\Leftarrow$: Aus $\sharp\mathbb{Z}_n^* = n - 1 = \varphi(n)$ folgt bereits, dass n Primzahl.
- Wichtige Bemerkung: das ist ein *existentielles* Kriterium für Primheit! Ein solches a ist Zeuge dafür, dass n Primzahl ist.

- Das Ordnungskriterium:

$$ord_n(a) = n-1 \quad \Leftrightarrow \quad \begin{cases} a^{n-1} \equiv 1 \bmod n & \wedge \\ \forall_{1 \le t < n-1}\ a^t \not\equiv 1 \bmod n \end{cases}$$

- Das Ordnungskriterium präzisiert:

$$ord_n(a) = n-1 \quad \Leftrightarrow \quad \begin{cases} a^{n-1} \equiv 1 \bmod n & \wedge \\ \forall_{\substack{1 \le t < n-1 \\ t \mid (n-1)}}\ a^t \not\equiv 1 \bmod n \end{cases}$$

- Das Ordnungskriterium ganz ökonomisch:

$$ord_n(a) = n-1 \quad \Leftrightarrow \quad \begin{cases} a^{n-1} \equiv 1 \bmod n & \wedge \\ \forall_{p \text{ prim}, p \mid n-1}\ a^{(n-1)/p} \not\equiv 1 \bmod n \end{cases}$$

- Das ganz ökonomische Primzahlkriterium (E. Lucas):

$$n \text{ ist Primzahl} \quad \Leftrightarrow \quad \exists a \in \mathbb{Z}_n^* : \begin{cases} a^{n-1} \equiv 1 \bmod n & \wedge \\ \forall_{p \text{ prim}, p \mid n-1}\ a^{(n-1)/p} \not\equiv 1 \bmod n \end{cases}$$

- Also: Primheit von n lässt sich mittels $k+1$ Exponentiationen modulo n verifizieren, wobei k = Anzahl der Primteiler von n.
- Die Ökonomie hat ihren Preis: das Kriterium ist rekursiv!
- Ist das noch effizient verifizierbar?
- JA!
  V. Pratt: *Every prime has a succinct certificate*,
  SIAM Journal on Computing, 4 (1975), 214-220.

Zertifikate für Primzahlen

Zertifikat $C(n)$ für die Primheit von einer Primzahl n:

- Angabe einer Faktorisierung

$$n-1 = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$$

  wobei die p_i ganze Zahlen ≥ 2 und die α_i ganze Zahlen ≥ 1 sind;
- Nachweis, dass die Zahlen $p_1, \ldots, p_k$ *Primzahlen* sind, durch Angabe von Zertifikaten $C(p_i)$ für die $p_i > 2$;
- Angabe einer positiven Zahl $a < n$ mit

$$a^{n-1} \equiv 1 \mod n \ \text{ und } \ a^{(n-1)/p_i} \not\equiv 1 \mod n \ (1 \le i \le k).$$

- 79 ist Primzahl, denn es gilt
  - $79 - 1 = 78 = 2 \cdot 3 \cdot 13$
  - $2, 3$ und 13 sind Primzahlen
  - $3^{78} \equiv 1$, $3^{78/2} \equiv 78$, $3^{78/3} \equiv 23$, $3^{78/13} \equiv 18 \mod 79$

  d.h. $C(79) = [\ 2 \cdot 3 \cdot 13\ ;\ C(3), C(13)\ ;\ 3\ ]$ ist ein Prim-Zertifikat für 79.
- 13 ist eine Primzahl, denn es gilt
  - $13 - 1 = 12 = 2^2 \cdot 3$
  - 2 und 3 sind Primzahlen
  - $2^{12} \equiv 1$, $2^6 \equiv 12$, $2^4 \equiv 3 \mod 13$

  d.h. $C(13) = [\ 2^2 \cdot 3\ ;\ C(3)\ ;\ 2\ ]$ ist ein Prim-Zertifikat für 13.
- 3 ist Primzahl, denn es gilt
  - $3 - 1 = 2 = 2$
  - 2 ist Primzahl
  - $2^2 \equiv 1$, $2^1 \equiv 2 \mod 3$

  d.h. $C(3) = [\ 2\ ;\ -\ ;\ 2\ ]$ ist ein Prim-Zertifikat für 3.

Ein grösseres Beispiel:

- $N = 1653701519$ ist Primzahl, denn:

$$N - 1 = 2 \cdot 7 \cdot 19 \cdot 23 \cdot 137 \cdot 1973 \ , \ 7^{N-1} \equiv 1 \bmod N \text{ und}$$

- ◊ 2 ist Primzahl und $7^{(N-1)/2} \equiv 1653701518 \bmod N$
- ◊ 7 ist Primzahl und $7^{(N-1)/7} \equiv 356579618 \bmod N$
- ◊ 19 ist Primzahl und $7^{(N-1)/19} \equiv 120777631 \bmod N$
- ◊ 23 ist Primzahl und $7^{(N-1)/23} \equiv 1080868740 \bmod N$
- ◊ 137 ist Primzahl und $7^{(N-1)/137} \equiv 101758286 \bmod N$
- ◊ 1973 ist Primzahl und $7^{(N-1)/1973} \equiv 1287679432 \bmod N$

- 1973 ist Primzahl, denn:

$$1972 = 2 \cdot 2 \cdot 17 \cdot 29, 3^{1972} \equiv 1 \bmod 1973 \text{ und}$$

- ◊ 2 ist Primzahl und $3^{1972/2} \equiv 1972 \bmod 1973$
- ◊ 17 ist Primzahl und $3^{1972/17} \equiv 273 \bmod 1973$
- ◊ 29 ist Primzahl und $3^{1972/29} \equiv 934 \bmod 1973$

- 137 ist Primzahl, denn:

$$136 = 2 \cdot 2 \cdot 2 \cdot 17 \ , \ 3^{136} \equiv 1 \bmod 137 \quad \text{und}$$

- ◊ 2 ist Primzahl und $3^{136/2} \equiv 136 \bmod 137$
- ◊ 17 ist Primzahl und $3^{136/17} \equiv 122 \bmod 13$

- 29 ist Primzahl, denn:

$$28 = 2 \cdot 2 \cdot 7 \ , \ 2^{28} \equiv 1 \bmod 29 \text{ und}$$

- ◊ 2 ist Primzahl und $2^{28/2} \equiv 28 \bmod 29$
- ◊ 7 ist Primzahl und $2^{28/7} \equiv 16 \bmod 29$

- 23 ist Primzahl, denn:

$$22 = 2 \cdot 11 \ , \ 5^{22} \equiv 1 \bmod 22 \text{ und}$$

- ◊ 2 ist Primzahl und $5^{22/2} \equiv 22 \bmod 23$
- ◊ 11 ist Primzahl und $5^{22/11} \equiv 2 \bmod 23$

- 19 ist Primzahl, denn:

$$18 = 2 \cdot 3 \cdot 3 \ , \ 2^{18} \equiv 1 \bmod 19 \text{ und}$$

- ◊ 2 ist Primzahl und $2^{18/2} \equiv 18 \bmod 19$
- ◊ 3 ist Primzahl und $2^{18/3} \equiv 7 \bmod 19$

- 17 ist Primzahl, denn:

$$16 = 2 \cdot 2 \cdot 2 \cdot 2 \ , \ 3^{16} \equiv 1 \bmod 17 \text{ und}$$

- ◊ 2 ist Primzahl und $3^{16/2} \equiv 16 \bmod 17$

- 11 ist Primzahl, denn:

$$10 = 2 \cdot 5 \ , \ 2^{10} \equiv 1 \bmod 11 \text{ und}$$

- ◊ 2 ist Primzahl und $2^{10/2} \equiv 10 \bmod 11$
- ◊ 5 ist Primzahl und $2^{10/5} \equiv 4 \bmod 11$

- 7 ist Primzahl, denn:

$$6 = 2 \cdot 3\ ,\ 3^6 \equiv 1 \bmod 7 \text{ und}$$

$\Diamond$ 2 ist Primzahl und $3^{6/2} \equiv 6 \bmod 7$
$\Diamond$ 3 ist Primzahl und $3^{6/3} \equiv 3 \bmod 7$

- 5 ist Primzahl, denn:

$$4 = 2 \cdot 2\ ,\ 2^4 \equiv 1 \bmod 5 \text{ und}$$

$\Diamond$ 2 ist Primzahl und $2^{4/2} \equiv 4 \bmod 5$

- 3 ist Primzahl, denn:

$$2 = 2\ ,\ 2^2 \equiv 1 \bmod 3 \text{ und}$$

$\Diamond$ 2 ist Primzahl und $2^{2/2} \equiv 2 \bmod 3$

Als "Zertifikate" geschrieben:

$$\begin{aligned}
C(1653701519) &= [\,2 \cdot 7 \cdot 19 \cdot 23 \cdot 137 \cdot 1973\,; \\
&\qquad C(7), C(19), C(23), C(137), C(1973)\,;\, 7\,] \\
C(1973) &= [\,2^2 \cdot 17 \cdot 29\ ;\ C(17), C(29)\ ;\ 3\,] \\
C(137) &= [\,2^3 \cdot 17\ ;\ C(17)\ ;\ 3\,] \\
C(29) &= [\,2^2 \cdot 7\ ;\ C(7)\ ;\ 2\,] \\
C(23) &= [\,2 \cdot 11\ ;\ C(11)\ ;\ 5\,] \\
C(19) &= [\,2 \cdot 3^2\ ;\ C(3)\ ;\ 2\,] \\
C(17) &= [\,2^4\ ;\ -\ ;\ 3\,] \\
C(11) &= [\,2 \cdot 5\ ;\ C(5)\ ;\ 2\,] \\
C(7) &= [\,2 \cdot 3\ ;\ C(3)\ ;\ 3\,] \\
C(5) &= [\,2^2\ ;\ -\ ;\ 2\,] \\
C(3) &= [\,2\ ;\ -\ ;\ 2\,]
\end{aligned}$$

Aufwand der Prim-Verifikation

Aufwandsabschätzung für das Überprüfen eines Zertifikats $C(n)$:
$T(n)$ = Anzahl der Verifikationen von Produkten

$$m \stackrel{?}{=} q_1^{\beta_1} q_2^{\beta_2} \cdot \ldots$$

und von mod-q-Berechnungen

$$x \stackrel{?}{\equiv} 1 \bmod q$$

die für das Überprüfen von $C(n)$ benötigt werden.

- Rekursion: Für $n - 1 = p_1^{\alpha_1} \cdot \ldots \cdot p_k^{\alpha_k}$

$$T(n) = 1 + \sum_{i=2}^{k} T(p_i) + k + 1$$

Dabei ist $p_1 = 2$, und $C(2)$ ist "kostenlos", d.h. $T(2) = 0$.

- Setze $S(n) = T(n) + 1$, dann ist

$$S(n) = \sum_{i=2}^{r} S(p_i) + 4$$

- Verifiziere per Induktion, dass für alle Primzahlen n:

$$S(n) \leq 4 \cdot \log_2 n$$

  - Für $n = 2$ ist nichts zu zeigen.
  - Für Primzahlen $n > 2$ mit

$$n - 1 \;=\; p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k} \quad (p_1 = 2)$$

  gilt dann:

$$S(n) \leq \sum_{i=2}^{k} (4 \cdot \log_2 p_i) + 4 = 4 \cdot \sum_{i=1}^{k} \log_2 p_i$$

$$= 4 \cdot \log \prod_{i=1}^{k} p_i \leq 4 \cdot \log n$$

- Beachte: alle Operationen (inklusive Exponentiationen (!!) mod q) werden mit Zahlen mit $\leq \log n$ Binärstellen durchgeführt — insgesamt ist der Rechenaufwand für das Verifizieren von $C(n)$ polynomial in $\log n$.
- Ganz wichtig: der Aufwand für das Finden der Zertifikate wird nicht berücksichtigt – es geht ausschliesslich um das Verifizieren!
- Fazit: Primheit lässt sich effizient *verifizieren*,

$$\textsc{Primes} \;\in\; \mathbb{NP}$$

- Tatsächlich gilt sogar: Primheit lässt sich effizient *entscheiden*

$$\textsc{Primes} \;\in\; \mathbb{P}$$

  Aber das weiss man erst seit 2002.

- Im August 2002 wurde von Agrawal, Kayal, Saxena das lange offene Problem endlich gelöst:

$$\textsc{Primes} \in \mathbb{P}$$

  Der AKS-Algorithmus
  - ist verblüffend einfach ($\Rightarrow$ folgende Seite)
  - die Begründung der Korrektheit ist nicht ganz so einfach, aber interessierten Studenten durchaus zugänglich
  - hat eine (bewiesene!) Laufzeit $\mathcal{O}(\log^{12} n)$ (aber vermutlich real noch deutlich besser)
- Hinweis: sehr instruktiver Artikel mit vielen weiteren Hinweisen F. Bornemann, Primes is in P: Ein Durchbruch für "Jedermann", *Mitteilungen der Deutschen Mathematiker-Vereinigung*, 4-2002, 14–21.
  engl. Übersetzung: Primes is in P: A Breakthrough for "Everyman", *Notices of the American Mathematical Society* 50/5 (2003), 545–552.

Der Algorithmus von Agrawal, Kayal, Saxena

Input: integer n > 1
if $n = a^b$ for $a \in \mathbb{N}$ and $b > 1$ **then**
  output COMPOSITE
end if
Find the smallest r such that $\text{ord}_r(n) > 4 \log^2 n$
if $1 < (a, n) < n$ for some $a \leq r$ **then**
  output COMPOSITE
end if
if $n \leq r$ **then**
  output PRIME
end if
for $a = 1$ to $\lfloor 2\sqrt{\phi(r)} \log n \rfloor$ **do**
  if $(X + a)^n \neq X^n + a \pmod{X^r - 1, n}$ **then**
    output COMPOSITE
  end if
end for
output PRIME

Problema, numeros primos a compositis dignoscendi . . . ad gravissima ac utilissima tabus arithmeticae pertinere. . .
. . . scientiae dignitas requirere videtur, ut omnia subsidia ad solutionem problematis tam elegantis ac celebris sedulo excolantur.
J. C. F. Gauss *(1777–1855),*
Disquisitiones Arithmeticae, Artikel 329

- Probedivision: n ist Primzahl $\Leftrightarrow \forall_{k:2\le k\le\sqrt{n}}\ k \not| n$

```
IS_PRIME (int n)
{
    for (int i = 2; i ≤ √n; i++)
        if (i | n) return FALSE;
    return TRUE;
}
```

- logarithmisches Kostenmodell:
  - input-Grösse ist $\log n$
  - Anzahl der Schleifendurchläufe ist $O(\sqrt{n}) = O(2^{\frac{1}{2}\log n})$
  - Jede Probedivision erfordert $O(\log^2 n)$ Bit-Operationen
  - Laufzeit für m-stelliges n ist im worst-case $O(m^2 \cdot 2^{m/2})$

Pseudo-Primzahltests

- $A(n,a)$: Eigenschaft ganzer Zahlen mit

$$n \text{ Primzahl} \quad \Rightarrow \quad \forall_{a:1<a<n}\ A(n,a)$$

  - Wird w $(1 < w < n)$ gefunden mit $\neg A(n,w)$, so ist n keine Primzahl:

$$\exists_{w:1<w<n}\ \neg A(n,w) \quad \Rightarrow \quad n \text{ ist keine Primzahl}$$

  - Solch ein w heisst *Zeuge* (witness) für die Zusammengesetztheit von n

- Beispiele
  - Teilbarkeitstest

$$D(n,a) : a \not| n$$

  - Euklid-Test

$$E(n,a) : \mathrm{ggT}(n,a) = 1$$

  - Fermat-Test

$$F(n,a) : \mathrm{ggT}(n,a) = 1 \Rightarrow a^{n-1} \equiv 1 \pmod{n}$$

  - SPP-Test ("strong probable prime", Miller-Rabin)
    sei $n - 1 = 2^t \cdot u$ mit ungeradem u

$$MR(n,a) : \begin{cases} a^u \equiv 1 \mod n & \text{oder} \\ a^{u\cdot 2^i} \equiv -1 \mod n & \text{für ein } i \text{ mit } 0 \le i < t \end{cases}$$

- Begründung für den SPP-Test

$$MR(n,a): n-1 = 2^t \cdot u \text{ mit ungeradem } u$$

$$\begin{cases} a^u \equiv 1 \mod n & \text{oder} \\ a^{u\cdot 2^i} \equiv -1 \mod n & \text{für ein } i \text{ mit } 0 \le i < t \end{cases}$$

  - n Primzahl $\Leftrightarrow$ $\mathbb{Z}_n$ Körper
  - In einem Körper hat die Gleichung $x^2 = 1$ genau zwei Lösungen $x = \pm 1$
    (allgemeiner: $x^2 = 1$ hat für $n = p^e$ ($p \ge 3$ Primzahl) in $\mathbb{Z}_n$ genau die beiden Lösungen $x = \pm 1$)
  - wird in $\mathbb{Z}_n^*$ ein Element $z \neq \pm 1$ mit $z^2 = 1$ gefunden, so ist n keine Primzahl
- beachte sukzessive Quadrierungen in $\mathbb{Z}_n^*$

$a^u \bmod n$, $a^{2u} \bmod n$, $a^{2^2 u} \bmod n$, $a^{2^3 u} \bmod n$, ... , $a^{2^t u} \bmod n$

Beispiele

- $n = 25, n-1 = 3 \cdot 8, a = 7$:

$$\begin{aligned} 7^3 &\equiv 18 \mod 25 \\ 7^6 &\equiv 24 \mod 25 \\ 7^{12} &\equiv 1 \mod 25 \\ 7^{24} &\equiv 1 \mod 25 \end{aligned}$$

Test bringt keine Information! $a = 7$ ist kein MR-Zeuge.

- $n = 25, n-1 = 3 \cdot 8, a = 2$:

$$\begin{aligned} 2^3 &\equiv 8 \mod 25 \\ 2^6 &\equiv 14 \mod 25 \\ 2^{12} &\equiv 21 \mod 25 \\ 2^{24} &\equiv 16 \mod 25 \end{aligned}$$

Test zeigt, daß 25 nicht prim ist! $a = 2$ ist MR-Zeuge.

- $n = 2047 = 23 \cdot 89, n-1 = 2 \cdot 1023, a = 2$:

$$\begin{aligned} 2^{1023} &\equiv 1 \mod 2047 \\ 2^{2046} &\equiv 1 \mod 2047 \end{aligned}$$

Test bringt keine Information! $a = 2$ ist kein MR-Zeuge.

- $n = 2047 = 23 \cdot 89, n - 1 = 2 \cdot 1023, a = 3$:

$$\begin{aligned} 3^{1023} &\equiv 1565 \bmod 2047 \\ 3^{2046} &\equiv 1013 \bmod 2047 \end{aligned}$$

Test zeigt, daß 2047 nicht prim ist! $a = 3$ ist MR-Zeuge.

- $n = 341 = 11 \cdot 31, n - 1 = 4 \cdot 85, a = 2$

$$\begin{aligned} 2^{85} &\equiv 32 \bmod 341 \\ 2^{170} &\equiv 1 \bmod 341 \\ 2^{340} &\equiv 1 \bmod 341 \end{aligned}$$

Test zeigt, daß 341 nicht prim ist! $a = 2$ ist MR-Zeuge.

- $n = 561 = 3 \cdot 11 \cdot 17, n - 1 = 16 \cdot 35, b = 2$

$$\begin{aligned} 2^{35} &\equiv 263 \bmod 561 \\ 2^{70} &\equiv 166 \bmod 561 \\ 2^{140} &\equiv 67 \bmod 561 \\ 2^{280} &\equiv 1 \bmod 561 \\ 2^{560} &\equiv 1 \bmod 561 \end{aligned}$$

Test zeigt, daß 561 nicht prim ist! $b = 2$ ist MR-Zeuge.
In der Tat ist 561 die kleinste Carmichael-Zahl, also eine Zahl, bei der es ausser den GGT-Zeugen keine weiteren Fermat-Zeugen gibt! Man weiss erst seit 1994, dass es unendlich-viele solche Zahlen gibt
(Alford, W.R.; Granville, A.; and Pomerance, C).
Siehe:
`http://mathworld.wolfram.com/CarmichaelNumber.html`

- $n = 2243, n - 1 = 2 \cdot 1121, b = 2$

$$\begin{aligned} 2^{1121} &\equiv 2242 \bmod 2243 \\ 2^{2242} &\equiv 1 \bmod 2243 \end{aligned}$$

$b = 2$ ist kein MR-Zeuge für die Zusammengesetztheit von 2243. Solche Zeugen darf und kann es nicht geben, denn 2243 ist Primzahl!

Beispiele für Zeugenmengen:

- $n = 13$

$$\begin{aligned} Dzeugen(13) &= [\,] \\ Ezeugen(13) &= [\,] \\ Fzeugen(13) &= [\,] \\ MRzeugen(13) &= [\,] \end{aligned}$$

- $n = 14$

$$\begin{aligned} Dzeugen(14) &= [2, 7] \\ Ezeugen(14) &= [2, 4, 6, 7, 8, 10, 12] \\ Fzeugen(14) &= [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] \\ MRzeugen(14) &= [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] \end{aligned}$$

- $n = 15$

$$\begin{aligned} Dzeugen(15) &= [3, 5] \\ Ezeugen(15) &= [3, 5, 6, 9, 10, 12] \\ Fzeugen(15) &= [2, 3, 5, 6, 7, 8, 9, 10, 12, 13] \\ MRzeugen(15) &= [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] \end{aligned}$$

- $n = 25$

$$\begin{aligned} Dzeugen(25) &= [5] \\ Ezeugen(25) &= [5, 10, 15, 20] \\ Fzeugen(25) &= [2, \ldots, 6, 8, \ldots, 17, 20, \ldots, 23] \\ MRzeugen(25) &= [2, \ldots, 6, 8, \ldots, 17, 20, \ldots, 23] \end{aligned}$$

Vergleich der Grösse von Zeugenmengen

n	D	E	F	MR
2	0	0	0	0
3	0	0	0	0
4	1	1	2	2
5	0	0	0	0
6	2	3	4	4
7	0	0	0	0
8	2	3	6	6
9	1	2	6	6
10	2	5	8	8
12	4	7	10	10
14	2	7	12	12
15	2	6	10	12

Vergleich der Grösse von Zeugenmengen

n	D	E	F	MR
16	3	7	14	14
18	4	11	16	16
20	4	11	18	18
21	2	8	16	18
22	2	11	20	20
24	6	15	22	22
25	1	4	20	20
26	2	13	24	24
27	2	8	24	24
28	4	15	24	24

Vergleich der Grösse von Zeugenmengen

n	D	E	F	MR
30	6	21	28	28
⋮	⋮	⋮	⋮	⋮
105	6	56	88	102
⋮	⋮	⋮	⋮	⋮
169	1	12	156	156
⋮	⋮	⋮	⋮	⋮
561	6	240	240	550
⋮	⋮	⋮	⋮	⋮
1105	6	336	336	1074
⋮	⋮	⋮	⋮	⋮
1729	6	432	432	1566
⋮	⋮	⋮	⋮	⋮

- Idee (probabilistischer) Primzahltests:

  Gelingt es trotz intensiver (zufälliger) Bemühungen nicht, einen Zeugen für die Zusammengesetztheit von n aufzutreiben, wird man n für eine Primzahl halten
- Annahme:
  - für n, a mit $1 < a < n$ ist die Un/Gültigkeit von $A(n, a)$ leicht zu überprüfen
  - ist n keine Primzahl, so sind Zeugen für die Zusammengesetztheit von n häufig

- Probabilistisches Verfahren:
  - wähle zufällig Kandidaten $k_1, k_2, \ldots, k_r$ mit $1 < k_i < n$ und überprüfe $A(n, k_i)(1 \leq i \leq r)$
  - wird dabei mindestens ein Zeuge für die Zusammengesetztheit von n gefunden, d.h. $\neg A(n, k_i)$, so ist n in der Tat zusammengesetzt
    — diese Aussage ist korrekt!
  - wird kein Zeuge für die Zusammengesetztheit von n gefunden, so wird n als Primzahl deklariert
    — dies ist mit nur sehr geringer Wahrscheinlichkeit eine falsche Entscheidung

Diskussion der Tests:

- Teilbarkeitstest
  Unbrauchbar, da es Nicht-Primzahlen n mit nur zwei Teilern $(\neq 1, n)$ gibt (Teilbarkeits-Zeugen)
- Euklid-Test
  Unbrauchbar, da es Nicht-Primzahlen n mit nur wenigen Euklid-Zeugen gibt
- Fermat-Test
  Unbrauchbar, da es zusammengesetzte Zahlen n gibt mit

$$\forall_{1<a<n} : \mathrm{ggT}(n, a) = 1 \quad \Rightarrow \quad a^{n-1} \equiv 1 (\mathrm{mod}\ n)$$

  d.h. alle Fermat-Zeugen sind schon Euklid-Zeugen

- Der Primzahltest von Miller-Rabin:
  - wähle (iteriert und zufällig) Zahlen

$$a \in \mathbb{Z}_n, a \neq \{0, 1\} \quad \text{mit } \mathrm{ggT}(n, a) = 1$$

    (falls a mit $\mathrm{ggT}(n, a) \neq 1$: sowieso fertig)
  - berechne $a^{n-1} \mathrm{mod}\ n$ durch "schnelle Exponentiation", d.h. durch iteriertes Quadrieren und Multiplizieren:

$$a^{(n-1)\,\mathrm{div}\,2^j} \quad \text{für } j = k, k-1, k-2, \ldots, 1, 0,$$

    wobei $k = \ell(n-1) + 1$
  - falls auf diesem Weg eine Situation

$$z \mapsto z^2 = 1 \quad \text{mit } z \neq \pm 1$$

    angetroffen wird: Zeuge für Zusammengesetztheit von n gefunden!


```
Miller_Rabin (int n)
{
    choose a ∈ [2 : n − 1] at random;
    /* primes are odd except for 2, which is an odd prime :-) */
    if (n == 2) return TRUE;
    if (n == 1 || even(n)) return FALSE;
    /* primes are relatively prime to a */
    if (ggT(a, n) > 1) return FALSE;
    /* compute z = a^{n−1} ≠ 1 mod n using iterated squaring */
    let (b_k, ..., b_0) be the binary representation of n − 1;
    int z = 1;
    for (int i = k; i ≥ 0; i−−)
    {
        int x = z;
        z = z^2 mod n;
        /* primes allow only trivial solutions of x^2 ≡ 1 mod p */
        if ((z == 1) && (x ≠ 1) && (x ≠ n − 1)) return FALSE;
        if (b_i == 1) z = z · a mod n;
    }
    if (z ≠ 1 mod n) return FALSE;      /* z = a^{n−1} */
    return TRUE;      /* no witness found: in dubio pro reo */
}
```

Beachte:

- ist $n-1 = 2^t \cdot u$ mit ungeradem u, so
  - sind die letzten $t+1$ Werte von z beim Quadrieren und Multiplizieren

    $a^u \bmod n$, $a^{2u} \bmod n$, $a^{2^2 u} \bmod n$, $a^{2^3 u} \bmod n$, ... , $a^{2^t u} \bmod n$
  - sind die letzten t Operationen sind nur Quadrierungen

Lemma:

- *Miller-Rabin-Zeugen sind häufig,*
  genauer:
  ist n eine zusammengesetzte Zahl, so ist die Anzahl der Miller-Rabin-Zeugen für diese Tatsache mindestens $(n-1)/2$

zu diesem Zweck wird gezeigt:

- Nicht-Zeugen sind Elemente von $\mathbb{Z}_n^*$ (klar!)
- Die Nicht-Zeugen bilden eine echte Untergruppe von $\mathbb{Z}_n^*$
- Wegen des Satzes von Lagrange hat diese Untergruppe $\leq \sharp\mathbb{Z}_n^*/2 \leq (n-1)/2$ Elemente
- es gilt sogar (Beweis etwas aufwendiger):
  die Untergruppe der Nicht-Zeugen hat $\leq \varphi(n)/4$ Elemente

- *der einfache Fall*:

  es gibt in $\mathbb{Z}_n^*$ einen Fermat-Zeugen,
  d.h. ein $a \in \mathbb{Z}_n^*$ mit $a^{n-1} \not\equiv 1 \bmod n$

  - $B = \{b \in \mathbb{Z}_n^* ;\ b^{n-1} \equiv 1 \bmod n\}$ ist eine Untergruppe von $\mathbb{Z}_n^*$
  - alle Nicht-Zeugen gehören zu B
  - wegen $x \notin B$ ist B eine echte Untergruppe von $\mathbb{Z}_n^*$

- *der etwas weniger einfache Fall*:

  es gibt in $\mathbb{Z}_n^*$ keine Fermat-Zeugen,
  d.h. $a^{n-1} \equiv 1 \bmod n$ für alle $a \in \mathbb{Z}_n^*$

- *der einfache Unter-Fall des etwas weniger einfachen Falles*:

  $n = p^e$ mit Primzahl $p > 2$ und $e > 2$

  - Fakt: $\mathbb{Z}_{p^e}^*$ ist eine zyklische Gruppe
  - $\sharp\mathbb{Z}_{p^e}^* = \varphi(p^e) = p^{e-1}(p-1)$
  - Ist $a \in \mathbb{Z}_{p^e}^*$ ein Element der Ordnung $\varphi(p^e)$, so gilt $a^{\varphi(p^e)} \equiv 1 \pmod n$ und $a^{n-1} \equiv 1 \pmod n$,
    also $\varphi(p^e) = p^{e-1}(p-1) \mid p^e - 1$: unmöglich!

- *der etwas weniger einfache Unter-Fall des etwas weniger einfachen Falles*:

  $n = n_1 \cdot n_2$ mit $n_1, n_2 > 1$ und $\mathrm{ggT}(n_1, n_2) = 1$

  - $n - 1 = 2^t \cdot u$ mit ungeradem u, für $a \in \mathbb{Z}_n^*$ betrachte

    $[a] = \langle a^u \bmod n, a^{2u} \bmod n, a^{2^2 u} \bmod n, a^{2^3 u} \bmod n, \ldots, a^{2^t u} \bmod n \rangle$

  - beachte: die letzte Komponente von $[a]$ ist immer $= 1$;
  - sei j mit $0 \leq j < t$ maximal mit der Eigenschaft, dass es ein $v \in \mathbb{Z}_n^*$ gibt $v^{2^j u} \equiv -1 \bmod n$

- $B = \{x \in \mathbb{Z}_n^* ; x^{2^j u} \equiv \pm 1 \bmod n\} \neq \emptyset$
- B ist eine Untergruppe von $\mathbb{Z}_n^*$, die alle Nicht-Zeugen enthält
- B ist eine echte Untergruppe von $\mathbb{Z}_n^*$:
  - sei $v \in \mathbb{Z}_n^*$ mit $v^{2^j u} \equiv -1 \bmod n$
  - $\Rightarrow v^{2^j u} \equiv -1 \bmod n_1$ und $v^{2^j u} \equiv -1 \bmod n_2$
  - Konstruiere mittels Chinesischem Restesatz $w \in \mathbb{Z}_n^*$ mit $w \equiv v \bmod n_1$ und $w \equiv 1 \bmod n_2$
  - $\Rightarrow w^{2^j u} \equiv -1 \bmod n_1$ und $w^{2^j u} \equiv 1 \bmod n_2 \Rightarrow w \notin B$

Theorem (Miller, Rabin, 1976)

- *Der Miller-Rabin-Primzahltest beurteilt bei m Iterationen eine zusammengesetzte Zahl n fälschlicherweise als Primzahl mit einer Wahrscheinlichkeit* $< (1/2)^m$ *bei einer Laufzeit von* $\mathcal{O}(m \cdot \ell(n)^3)$

In der Terminologie der Komplexitätstheorie

$$\textsc{Primes} \in \text{co-}\mathbb{RP}$$

wobei: $\mathbb{RP}$ = *random polynomial time*
= Klasse der Probleme mit effizienten probabilistischen Entscheidungsverfahren mit einseitigem Fehler ("biased Monte Carlo")

- Zahlenbeispiel zur Effizienz des Miller-Rabin-Tests:
  - ε : Fehlerwahrscheinlichkeit des MR-Tests
  - Aufwand zum Testen einer k-stelligen Zahl ($\log \frac{1}{\epsilon} \cdot k^3$)
  - Annahme:
    10^6 arithmetische Operationen pro Sekunde, $\varepsilon = 10^{-100}$

$k = 30$	$\Rightarrow$	1 sec
$k = 50$	$\Rightarrow$	12.5 sec
$k = 100$	$\Rightarrow$	100 sec

Ein wesentlich schwieriger darzustellendes und zu begründendes Verfahren von Adleman und Huang (1992) zeigt

$$\textsc{Primes} \in \mathbb{RP}$$

Aus beiden Aussagen zusammen erhält man

$$\textsc{Primes} \in \mathbb{ZPP} = \mathbb{RP} \cap \text{co-}\mathbb{RP}$$

wobei: $\mathbb{ZPP}$ = *zero error random polynomial time*
= Klasse der Probleme mit probabilistischen Entscheidungsverfahren, die im Mittel effizient sind
("Las Vegas")

Weitere Informationen zur Komplexitätssituation für Primes

- Primzahlen sind effizient verifizierbar (Pratt, 1975)

$$\textsc{Primes} \in \mathbb{NP}$$

- Zusammen mit dem offensichtlichen (!!) $\textsc{Primes} \in \text{co-}\mathbb{NP}$ ergibt sich

$$\textsc{Primes} \in \mathbb{NP} \cap \text{co-}\mathbb{NP}$$

- Miller hat 1976 gezeigt, dass Primzahlen *deterministisch* mit Aufwand $\mathcal{O}(\log^5 n)$ erkannt werden können, denn
  - ist n keine Primzahl, dann ist kleinste MR-Zeuge für die Zusammengesetztheit von n kleiner als $2 \ln^2 n$ (Bach, 1985)

  Hierbei wird allerdings eine bislang unbewiesene Hypothese des Zahlentheorie (ERH) verwendet!
- Es gibt einen (in der Praxis!!) sehr effizienten *deterministischen* Primzahltest von Adelman, Pomerance, Rumeley (1983) mit Laufzeit

$$\mathcal{O}((\log n)^{c \cdot \log\log\log n})$$

- Im August 2002 wurde von Agrawal, Kayal, Saxena das lange offene Problem endlich gelöst:

$$\textsc{Primes} \in \mathbb{P}$$

- Literaturhinweise:
  - R. Crandall, C. Pomerance, *Prime Numbers, A Computational Perspective*, Springer-Verlag, 2001.
  - M. Dietzfelbinger, *Primality Testing in Polynomial Time*, Springer Verlag 2004.
  - siehe auch Webseite zur Vorlesung

```
> pptest := proc (N::integer,t::integer)
  #
  # Probabilistischer Primzahltest, der mit
  # zufällig gewählten Startwerten 1 < a < N
  # maximal t Iterationen von
  #   Teilbarkeitstest,
  #   ggT-Test,
  #   Fermat-Kongruenztest
  #   Miller-Rabin-Test
  # durchführt.
  #
  local a,d,e,j,k,s,u,f,found,randomelement;
  randomelement := rand(2..N-1);
  for s from 1 to t do
  a := randomelement();
  #
  # Teilbarkeitstest
  #
  if N mod a = 0 then
  RETURN(N,`ist keine Primzahl:`,a,`teilt`,N) fi;
  #
  # Euklid-Test
  #
  d := igcd(a,N);
  if d>1 then
  RETURN(N,`ist keine Primzahl: der GGT von`,a,`und`,N,`ist`,d) fi;
  #
  # Fermat-Test
  #
  e := a &^(N-1) mod N;
  if not (e=1) then
  RETURN(N,`ist keine Primzahl:
  Fermat-Test mit`,a,`liefert a^(N-1) mod N =`,e) fi;
  #
  # Miller-Rabin-Test
  #
  print(`Miller-Rabin-Test wird gestartet`);
  u := N-1;
  k := 0;
  while (u mod 2 = 0) do
     u := u/2;
     k := k+1;
  od;
  f := a &^u mod N;
  print(f);
  if (f=-1 mod N) or (f=1 mod N) then break fi;
  found := false;
  for j from 1 to k do
     f := f &^2 mod N;
     print(f);
     if f=-1 mod N then found := true; break fi;
     od;
  if found then break else
  RETURN(N,`ist keine Primzahl: Miller-Rabin-Test!`)
  fi;
  od;
  print(`MR-Test`,t,`-mal bestanden:`,N,` ist vermutlich Primzahl`);
  1;
  end:
```

```
> pptest(41,3);
                    Miller-Rabin-Test wird gestartet
                                  1
MR-Test, 3, -mal bestanden:, 41, ist vermutlich Primzahl
                                  1
```

```
> pptest(561,5);
                    Miller-Rabin-Test wird gestartet
                                 287
                                 463
                                  67
                                  1
                                  1
                    561, ist keine Primzahl: Miller-Rabin-Test!
```

```
> pptest(10007,5);
                    Miller-Rabin-Test wird gestartet
                                10006
MR-Test, 5, -mal bestanden:, 10007, ist vermutlich Primzahl
                                  1
```

```
> isprime(10007);
                                 true
```

```
> findprime := proc (N::integer,t::integer)
  #
  # die nächstgrössere Primzahl nach N
  # wird mittels probabilistische Primzahltest
  # bestimmt
  # t = Anzahl der Iterationen
  #
  local n;
  n := N;
  if (n mod 2 = 0) then n := n+1 fi;
  while true do
  if pptest(n,t)=1 then break fi;
  n := n+2;
  od;
  end;
findprime := proc(N::integer, t::integer)
local n;
    n := N;
    if `mod`(n, 2) = 0 then n := n + 1 end if
```

```
    do if pptest(n, t) = 1 then break end if n := n + 2 end do
end proc
```

```
> findprime(8234857628934756343,5);
                                    Miller-Rabin-Test wird gestartet
                                      7156348946777158704
                                      3903333351366841698
                                      8234857628934756360
MR-Test, 5, -mal bestanden:, 8234857628934756361, ist vermutlich Primzahl
                                      8234857628934756361
```

```
> isprime(%);
                                             true
```

```
>
```

Verschlüsselung durch Exponentiation (Pohlig, Hellman, 1976)

- p : eine (grosse) Primzahl
- e : Zahl $0 < e < p$ mit $\mathrm{ggT}(e, p-1) = 1$
- d Inverses von e in $\mathbb{Z}^*_{p-1}$, d.h. $d \cdot e \equiv 1 \bmod p-1$ $(= \phi(p))$
- M : numerisch codierter Text
- M in Blöcke M_i mit $0 < M_i < p$ zerlegen (i=1,2,3...)
- Codierung (*encryption*)

$$M_i \mapsto C_i := M_i^e \bmod p$$

(i=1,2,3,...)

- Decodierung (*decryption*)

$$C_i \mapsto C_i^d \equiv (M_i^e)^d \equiv (M_i)^{e \cdot d} \equiv M_i^{k \cdot \phi(p)+1} \equiv M_i \bmod p$$

(i=1,2,3,...)

numerische Codierung

␣	A	B	C	D	...	Y	Z
↕	↕	↕	↕	↕	...	↕	↕
00	01	02	03	04	...	25	26

Parameter: $p = 7951$, $e = 91, d = 961$

Text: ENCRYPTION REGULATION

Numerisch codierter Text;

$$M = 05\ 14\ 03\ 18\ 25\ 16\ 20\ 09\ 15\ 14\ 00\ 18\ 05\ 07\ 21\ 12\ 01\ 20\ 09\ 15\ 14\ 00$$

Zerlegung in Blöcke mit je 4 Ziffern

0514 0318 2516 2009 1514 0018 0507 2112 0120 0915 1400

Verschlüsselung

$$\begin{array}{rcrcr} C_1 & = & 0514^{91} \bmod 7951 & = & 2174 \\ C_2 & = & 0318^{91} \bmod 7951 & = & 4468 \\ C_3 & = & 2516^{91} \bmod 7951 & = & 7889 \\ \vdots & & \vdots & & \vdots \\ C_{11} & = & 1400^{91} \bmod 7951 & = & 7114 \end{array}$$

Ciphertext

2174 4468 7889 6582 0924 5460 7868 7319 0726 2890 7114

Entschlüsselung

$$\begin{array}{rcrcr} M_1 & = & 2174^{961} \bmod 7951 & = & 514 \\ M_2 & = & 4468^{961} \bmod 7951 & = & 318 \\ M_3 & = & 7889^{961} \bmod 7951 & = & 2516 \\ \vdots & & \vdots & & \vdots \\ M_{11} & = & 7114^{961} \bmod 7951 & = & 1400 \end{array}$$

Public-Key-Kryptographie

- Ziel: sicherer Informationsaustausch zwischen Teilnehmern $\{Alice, Bob, Cesar, \ldots\}$ an einem öffentlichen Netz, bei dem die transportierten (verschlüsselten) Daten abgehört werden können
- Klassisch ("symmetrische Kryptosysteme", z.B. DES): jedes Paar $\langle A, B \rangle$ von Teilnehmern besitzt identische Schlüssel zum Verschlüsseln und Entschlüsseln, die nur diesen bekannt sind
  Problem: wie können A und B identische Schlüssel erhalten/erzeugen?

als nicht-technische Lektüre dringendst empfohlen:
S. Singh, *Geheime Botschaften*, dtv, 2001. (Engl.: *The Code Book*)

- Public-Key-Idee ("asymmetrisch", W. Diffie, M. Hellman, 1976):
  Schlüssel über das Netz selbst zugänglich machen
  - Jeder Teilnehmer B erzeugt ein Paar $\langle k_E, k_D\rangle$ von Schlüsseln,
  - k_E wird von B öffentlich bekanntgegeben ("Telefonbuch"),
    k_D bleibt Geheimnis von B
  - Will eine Teilnehmerin A am Netz eine Nachricht N in verschlüsselter Form an B schicken, so
    * besorgt sich A den Schlüssel k_E (unverschlüsselt) aus dem Telefonbuch
    * verschlüsselt N mittels k_E zu $\mathcal{E}(N, k_E)$ mittels eines geeigneten Verfahrens $\mathcal{E}$ und sendet $\mathcal{E}(N, k_E)$ zu B
    * B erhält $\mathcal{E}(N, k_E)$ und entschlüsselt dies zu $\mathcal{D}(\mathcal{E}(N, k_E), k_D) = N$ mit Hilfe eines geeigneten Verfahrens $\mathcal{D}$
  - Sicherheit: mit vernünftigem Aufwand
    * aus verschlüsselter Nachricht $\mathcal{E}(N, k_E)$ keine Rückschlüsse auf N
    * aus der Kenntnis von k_E keine Kenntnis von k_D

    möglich, auch wenn Verfahren $\mathcal{E}$ und $\mathcal{D}$ öffentlich bekannt sind

alternative Verwendung asymmetrischer Systeme:

Authentifizieren von (nicht geheimen) Nachrichten, falls

$$\mathcal{D}(\mathcal{E}(N, k_E), k_D) = N = \mathcal{E}(\mathcal{D}(N, k_D), k_E)$$

- B benutzt seinen privaten Schlüssel k_D um Nachricht N zu "verschlüsseln": $\mathcal{D}(N, k_D)$ ("Signatur")
- B sendet $\langle N, \mathcal{D}(N, k_D)\rangle$ zu A
- A verschlüsselt die Signatur $\mathcal{D}(N, k_D)$ mit k_E und vergleicht das Ergebnis $\mathcal{E}(\mathcal{D}(N, k_D), k_E)$ mit der Nachricht N

Das RSA-Verfahren

R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, *Communications of the ACM*, 21 (1978), 120–126

O.E.: Nachrichten sind als (Blöcke von) positiven ganzen Zahlen codiert

Jeder Teilnehmer am System

- wählt zwei grosse Primzahlen p und q (typisch: ≥ 100 Dezimalstellen)
- berechnet $n = p \cdot q$ und $\phi(n) = (p-1)(q-1)$
- wählt eine grosse "zufällige" ungerade Zahl d mit $1 < d < \phi(n)$ und $\mathtt{ggT}(d, \phi(n)) = 1$
- berechnet $e = d^{-1} \bmod \phi(n)$
- veröffentlicht das Paar $k_E = (e, n)$ als öffentlichen Schlüssel
- hält die Daten (d, p, q) geheim, privater Schlüssel $k_D = (d, n)$
- Verschlüsselung: $\mathcal{E}_{(e,n)} : \mathbb{Z}_n^* \to \mathbb{Z}_n^* : N \mapsto N^e \bmod n$
- Entschlüsselung: $\mathcal{D}_{(d,n)} : \mathbb{Z}_n^* \to \mathbb{Z}_n^* : C \mapsto C^d \bmod n$

Demonstrationsbeispiel

Systemparameter:

$p=$	47	Primzahl
$q=$	59	Primzahl
$n = p \cdot q=$	2773	
$\phi(n)=$	$(p-1)(q-1) = 2668$	Eulers Funktion
$d=$	157	$\mathrm{ggT}(d, \phi(n)) = 1$
$e = d^{-1} \bmod \phi(n)=$	17	mittels erweitertem EA
$k_E=$	$(17, 2773)$	öffentlicher Schlüssel
$k_D=$	$(157, 2773)$	privater Schlüssel

Text

ITS ALL GREEK TO ME

W. Shakespeare, *Julius Cesar*, 1. Akt, 2. Szene

numerisch codierter Text

0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

Verschlüsselung

$$\begin{array}{rcccc} C_1 & = & 920^{17} \bmod 2773 & = & 948 \\ C_2 & = & 1900^{17} \bmod 2773 & = & 2342 \\ C_3 & = & 0112^{17} \bmod 2773 & = & 1084 \\ \vdots & & \vdots & & \\ C_{10} & = & 0500^{17} \bmod 2773 & = & 1665 \end{array}$$

Ciphertext

0948 2342 1084 1444 2663 2390 0778 0774 0219 1655

Entschlüsselung

Ciphertext

0948 2342 1084 1444 2663 2390 0778 0774 0219 1655

$$\begin{array}{rcccc} M_1 & = & 948^{157} \bmod 2773 & = & 920 \\ M_2 & = & 2342^{157} \bmod 2773 & = & 1900 \\ M_3 & = & 1084^{157} \bmod 2773 & = & 0112 \\ \vdots & & \vdots & & \\ M_{10} & = & 1665^{157} \bmod 2773 & = & 0500 \end{array}$$

numerisch codierter Text

$M =$ 0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

Text

ITS ALL GREEK TO ME

Klassisches Beispiel 1

Parameter:

$n =$ RSA-129 $=$ 114381625757888867669235779976146612010218296721242362562
561842935706935245733897830597123563958705058989075147599290026879543541

$e = 9007$

Text:

ITS ALL GREEK TO ME

W. Shakespeare, *Julius Cesar*, 1. Akt, 2. Szene

numerisch codierter Text

092019000112120007180505110020150013 05

verschlüsselter Text:

199935131497805100452317122740260647423204017058391463103703717 4
062597160894892750430992096267258267501289355446135382376974802 6

Klassisches Beispiel 2: das 100 $ - RSA -Problem

> M. Gardner, “Mathematical Games - A New Kind of Cipher that Would Take Millions of Years to Break”, *Scientific American*, 237, 2 (1977), 120-124.

Parameter:

$n =$ 114381625757888867669235779976146612010218296721242362562561842
935706935245733897830597123563958705058989075147599290026879543541

$e = 9007$

numerisch codierter Text:

968696137546220614771409222543558829057599911245 7
431987469512093081629822514570835693147662288398 9
628013391990551829945157815154

Faktorisierung von RSA-129

D. Atkins, M. Graff, A. K. Lenstra, P. Leyland et al.,
2. April 1994, mit Aufwand von ca. 5000 mips-Jahren, 8 Monate Rechenzeit auf ≥ 600 workstations
Methode: "Multiple Polynomial Quadratic Sieve"

RSA-129 ist das Produkt der beiden Primzahlen

$p = 3490529510847650949147849619903898133417764638493387843990820577$

$q = 32769132993266709549961988190834461413177642967992942539798288533$

Zur Entschlüsselung benötigt man das Inverse d von $e = 9007$ modulo $\phi(n) = \phi(p \cdot q) = (p-1)(q-1)$

$$d = 106698614368578024442868771328920154780709906633937862801226224496631063125911774470873340168597462306553968544513277109053606095$$

Die Entschlüsselung

$$M \mapsto M^d \bmod n$$

wird mittels "Schneller Exponentiation" in $\mathbb{Z}_n$ ausgeführt, wobei man die 426 Bit lange Binärdarstellung von d verwendet

100111011001111110010100110010001000001000001110100111100100110
010011110100111000000000000011111110100001101010110001011101111
010100001111101100000010000011101101010101111010101001111110110
110100001111110100000011110100110001011001011001101001010001100
100111010110000101110100101011010000011100000001110001110101010
011011101000111101001110001101011010101010010011101010001001111
000000100111010011000110111110101100100011001111

Nach Entschlüsselung

$$200805001301070903002315180419000118050019172105011309190800151919090618010705$$

Übersetzung in Text

THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

Die Verschlüsselungsabbildung

$$N \mapsto N^e \bmod n$$

wird natürlich auch mittels "Schneller Exponentiation" in $\mathbb{Z}_n^*$ ausgeführt, wobei $e = 9007$ die Binärdarstellung

10001100101111

hat

Die 100$-Nachricht von Rivest, Shamir, Adleman war signiert mit Hilfe der Entschlüsselungsabbildung:

$$167178611503808442460152713891683982454369010323583112178350384469290626554487922371144905095786086556624965779748400040570203738446929062655448792237114490509578608655662496577974840004057020373$$

Mittels Verschlüsselungsabbildung erhält man

$$0609181920001915122205180023091419001514050008211404180504000415121201181 9$$

im Klartext:

FIRST SOLVER WINS ONE HUNDRED DOLLARS

Korrektheit

- für die Hintereinanderausführung von Verschlüsselung und Entschlüsselung

$$N \overset{\mathcal{E}_{(e,n)}}{\longrightarrow} N^e \bmod n \overset{\mathcal{D}_{(d,n)}}{\longrightarrow} (N^e)^d \bmod n$$

für $N \in \mathbb{Z}_n^*$ und $e \cdot d = 1 + k \cdot \phi(n)$

gilt wegen des Satzes von Euler:

$$N^{ed} \equiv N^{1+k\cdot\phi(n)} \equiv N \cdot (N^{\phi(n)})^k \equiv N \cdot 1 \equiv N \bmod n$$

Effizienz

- grosse Primzahlen p, q mittels randomisiertem Primzahltest gewinnen: mittlere Anzahl der Versuche um eine ℓ-stellige Primzahl zu finden ist wegen Primzahlsatz $\in \Theta(\ell)$
- Multiplikationen $n = p \cdot q$ und $\phi(n) = (p-1)(q-1)$
- Exponent d zufällig wählen und mittels EA auf Teilerfremdheit mit $\phi(n)$ testen
- Inverses $e = d^{-1} \bmod \phi(n)$ mittels erweitertem EA berechnen
- Ver- und Entschlüsselung mittels schneller Exponentiation (Quadrieren und Multiplizieren)

Sicherheit

- Wenn es einem Angreifer gelingt, die Faktorisierung $n = p \cdot q$ zu ermitteln, kann er auch $d = e^{-1} \bmod \phi(n)$ mittels EA berechnen
- Bislang keine wirklich effizienten Faktorisierungsalgorithmen bekannt (aber: Quantencomputer? Algorithmus von Shor (1994))
- Ist es wirklich nötig, n zu faktorisieren, um RSA zu brechen? (wenn man n und $\phi(n)$ kennt, kann man p und q berechnen! Wie?)
- RSA hat "Schwachstellen", die man respektieren sollte (z.B. sehr kleine e bzw. d, oder wenn $p-1$ und $q-1$ viele kleine Primteiler haben)

Empfohlene Lektüre:
D. Boneh, *Twenty Years of Attacks on the RSA Cryptosystem*,
Notices of the American Mathematical Society, vol. 46, 203–213, 1999.
`http://www.ams.org/notices/199902/199902-toc.html`

aus: Eric Weisstein's Mathworld

⇒ http://mathworld.wolfram.com/news/2003-12-05/rsa/

MathWorld Headline News

RSA-576 Factored

By Eric W. Weisstein

December 5, 2003--On December 3, the day after the announcement of the discovery of the largest known prime by the Great Internet Mersenne Prime Search on December 2 (*MathWorld* headline news, December 2, 2003), a team at the German Federal Agency for Information Technology Security (BIS) announced the factorization of the 174-digit number

> 1881 9881292060 7963838697 2394616504 3980716356 3379417382
> 7007633564 2298885971 5234665485 3190606065 0474304531
> 7388011303 3967161996 9232120573 4031879550 6569962213
> 0516875930 7650257059

known as RSA-576.

RSA numbers are composite numbers having exactly two prime factors (i.e., so-called semiprimes) that have been listed in the Factoring Challenge of RSA Security®.

number	digits	prize	factored
RSA-100	100		Apr. 1991
RSA-110	110		Apr. 1992
RSA-120	120		Jun. 1993
RSA-129	129	$100	Apr. 1994
RSA-130	130		Apr. 10, 1996
RSA-140	140		Feb. 2, 1999
RSA-150	150	withdrawn?	open [see postscript]
RSA-155	155		Aug. 22, 1999
RSA-160	160		Apr. 1, 2003
RSA-576	174	$10,000	Dec. 3, 2003
RSA-640	193	$20,000	open
RSA-704	212	$30,000	open
RSA-768	232	$50,000	open
RSA-896	270	$75,000	open
RSA-1024	309	$100,000	open
RSA-1536	463	$150,000	open
RSA-2048	617	$200,000	open

Postscript added August 24, 2004:
RSA-150 was factored into two 75-digit primes by Aoki *et al.* in a preprint dated April 16, 2004.

RSA-Aufgaben

- *In einem RSA-System wird der Ciphertext 10 übertragen. Der öffentliche Schlüssel ist* $(5, 35)$. *Welches war die Nachricht?*

  Lösung:

  - $n = 35 = 5 \cdot 7 \;\Rightarrow\; \phi(n) = (5-1)(7-1) = 24$
  - $e = 5 \;\Rightarrow\; d = 5^{-1} \bmod 24 = 5$
  - $10^5 \bmod 35 = 5 \bmod 35 \;\Rightarrow\; M = 5$

- *Ein RSA-System hat* $(31, 3599)$ *als öffentlichen Schlüssel. Welches ist der private Schlüssel?*

  Lösung:

  - $n = 3599 = 59 \cdot 61 \;\Rightarrow\; \phi(n) = 58 \cdot 60 = 3480$
  - $d = 31^{-1} \mod 3480 = -499 \mod 3480 = 3031$ folgt aus eeA mit Bézout-Beziehung
    $$4 \cdot 3480 - 449 \cdot 31 = 1$$
  - privater Schlüssel $(3031, 3599)$

MathWorld Headline News

RSA-576 Factored

By Eric W. Weisstein

December 5, 2003--On December 3, the day after the announcement of the discovery of the largest known prime by the Great Internet Mersenne Prime Search on December 2 (*MathWorld* headline news, December 2, 2003), a team at the German Federal Agency for Information Technology Security (BIS) announced the factorization of the 174-digit number

> 1881 9881292060 7963838697 2394616504 3980716356 3379417382
> 7007633564 2298885971 5234665485 3190606065 0474304531
> 7388011303 3967161996 9232120573 4031879550 6569962213
> 0516875930 7650257059

known as RSA-576.

RSA numbers are composite numbers having exactly two prime factors (i.e., so-called semiprimes) that have been listed in the Factoring Challenge of RSA Security®.

While composite numbers are defined as numbers that can be written as a product of smaller numbers known as factors (for example, 6 = 2 x 3 is composite with factors 2 and 3), prime numbers have no such decomposition (for example, 7 does not have any factors other than 1 and itself). Prime factors therefore represent a fundamental (and unique) decomposition of a given positive integer. RSA numbers are special types of composite numbers particularly chosen to be difficult to factor, and they are identified by the number of digits they contain.

While RSA-576 is a *much* smaller number than the 6,320,430-digit monster Mersenne prime announced earlier this week, its factorization is significant because of the curious property of numbers that proving or disproving a number to be prime ("primality testing") seems to be *much* easier than actually identifying the factors of a number ("prime factorization"). Thus, while it is trivial to multiply two large numbers p and q together, it can be extremely difficult to determine the factors if only their product pq is given. With some ingenuity, this property can be used to create practical and efficient encryption systems for electronic data.

RSA Laboratories sponsors the RSA Factoring Challenge to encourage research into computational number theory and the practical difficulty of factoring large integers and also because it can be helpful for users of the RSA encryption public-key cryptography algorithm for choosing suitable key lengths for an appropriate level of security. A cash prize is awarded to the first person to factor each challenge number.

RSA numbers were originally spaced at intervals of 10 decimal digits between one and five hundred digits, and prizes were awarded according to a complicated formula. These original numbers were named according to the number of decimal digits, so RSA-100 was a hundred-digit number. As computers and algorithms became faster, the unfactored challenge numbers were removed from the prize list and replaced with a set of numbers with fixed cash prizes. At this point, the naming convention was also changed so that the trailing number indicates the number of digits in the *binary* representation of the number. Hence, RSA-576 has 576 binary digits, which translates to 174 digits in decimal.

RSA numbers received widespread attention when a 129-digit number known as RSA-129 was used by R. Rivest, A. Shamir, and L. Adleman to publish one of the first public-key messages together with a $100 reward for the message's decryption (Gardner 1977). Despite widespread belief at the time that the message encoded by RSA-129 would take millions of years to break, it was factored in 1994 using a distributed computation that harnessed networked computers spread around the globe performing a multiple polynomial quadratic sieve (Leutwyler 1994). The result of all the concentrated number crunching was decryption of the encoded message to yield the profound plain-text message "The magic words are squeamish ossifrage." (An ossifrage is a rare predatory vulture found in the mountains of Europe.)

Factorization of RSA-129 followed earlier factorizations of RSA-100, RSA-110, and RSA-120. The challenge numbers RSA-130, RSA-140, RSA-155, and RSA-160 were also subsequently factored between 1996 and April of this year. (Amusingly, RSA-150 apparently remains unfactored following its withdrawal from the RSA Challenge list. [However, see postscript.])

On December 2, Jens Franke circulated an email announcing factorization of the smallest prize number RSA-576. The factorization was accomplished using a prime factorization algorithm known as the general number field sieve. The two 87-digit factors found using this sieve are

> 3980750 8642406493 7397125500 5503864911 9906436234 2526708406
> 3851895759 4638895726 1768583317
> x
> 4727721 4610743530 2536223071 9730482246 3291469530 2097116459
> 8521711305 2071125636 3590397527

and can easily be multiplied to verify that they do indeed give the original number.

Franke's note detailed the factorization process in which "lattice" sieving was done by J. Franke and T. Kleinjung using hardware at the Scientific Computing Institute and the Pure Mathematics Institute at Bonn University, Max Planck Institute of Mathematics in Bonn, and Experimental Mathematics Institute in Essen; and "line" sieving was done by P. Montgomery and H. te Riele at CWI, F. Bahr and his family, and NFSNET (which at that time consisted of D. Leclair, P. Leyland, and R. Wackerbarth). Post-processing of this data to construct the actual factors was then done with the support of the BSI.

For their efforts, the team will receive a cash prize of $10,000 from RSA Security. However, award seekers need not be deterred. As the following table shows, RSA-640 to RSA-2048 remain open, carrying awards from $20,000 to $200,000 to whoever is clever and persistent enough to track them down. A list of the open challenge numbers may be downloaded from RSA or in the form of a *Mathematica* package from the *MathWorld* package archive.

number	digits	prize	factored
RSA-100	100		Apr. 1991
RSA-110	110		Apr. 1992
RSA-120	120		Jun. 1993
RSA-129	129	$100	Apr. 1994
RSA-130	130		Apr. 10, 1996
RSA-140	140		Feb. 2, 1999
RSA-150	150	withdrawn?	open [see postscript]
RSA-155	155		Aug. 22, 1999
RSA-160	160		Apr. 1, 2003
RSA-576	174	$10,000	Dec. 3, 2003
RSA-640	193	$20,000	open
RSA-704	212	$30,000	open
RSA-768	232	$50,000	open
RSA-896	270	$75,000	open
RSA-1024	309	$100,000	open
RSA-1536	463	$150,000	open
RSA-2048	617	$200,000	open

Postscript added August 24, 2004:
RSA-150 was factored into two 75-digit primes by Aoki *et al.* in a preprint dated April 16, 2004.

References

Franke, J. "RSA576." Privately circulated email reposted to primenumbers Yahoo! Group.

Gardner, M. "Mathematical Games: A New Kind of Cipher That Would Take Millions of Years to Break." *Sci. Amer.* **237**, 120-124, Aug. 1977.

Leutwyler, K. "Superhack: Forty Quadrillion Years Early, a 129-Digit Code Is Broken." *Sci. Amer.* **271**, 17-20, 1994.

NFSNet: Large-Scale Distributed Factoring.
http://www.nfsnet.org

RSA Security®. "The New RSA Factoring Challenge."
http://www.rsasecurity.com/rsalabs/challenges/factoring

RSA Security®. "The RSA Challenge Numbers."
http://www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html

Weisstein, E. W. *Mathematica* package `RSANumbers.m`.

RSA Laboratories

CRYPTOBYTES
TECH NOTES
BULLETINS
SUBMISSIONS
STAFF & ASSOCIATES
CHALLENGES

The New RSA Factoring Challenge
The RSA Challenge Numbers
The RSA Factoring Challenge FAQ
Factorization Submission Form
RSA-640 is factored!
RSA-200 is factored!
RSA-576 is factored!
RSA-160 is factored!
RSA-155 is factored!
RSA-140 is factored!
The RSA Laboratories Secret-Key Challenge
DES Challenge III

RFID
NIGHTINGALE
PKCS
OTPS
RSA ALGORITHM
CRYPTO FAQ

Home: Cryptographic Challenges: The New RSA Factoring Challenge

RSA-640 is factored!

The factoring research team of F. Bahr, M. Boehm, J. Franke, T. Kleinjung continued its productivity with a successful factorization of the challenge number RSA-640, reported on November 2, 2005. The factors [verified by RSA Laboratories] are:

163473364580925384844313388386509085984178367003309231281111085238933310010450815121211816751157 9

and

190087128166482211312685157393541397547189678996851549366663853908802710380210449895719126146557 1

The effort took approximately 30 2.2GHz-Opteron-CPU years according to the submitters, over five months of calendar time. (This is about half the effort for RSA-200, the 663-bit number that the team factored in 2004.)

Top of Page

Email to a
Print

RSA Security - RSA-640 is factored! 26.01.2006 09:27 Uhr

MathWorld Headline News

RSA-640 Factored

By Eric W. Weisstein

November 8, 2005--A team at the German Federal Agency for Information Technology Security (BSI) recently announced the factorization of the 193-digit number

310 7418240490 0437213507 5003588856 7930037346 0228427275
4572016194 8823206440 5180815045 5634682967 1723286782
4379162728 3803341547 1073108501 9195485290 0733772482
2783525742 3864540146 9173660247 7652346609

known as RSA-640 (Franke 2005). The team responsible for this factorization is the same one that previously factored the 174-digit number known as RSA-576 (*MathWorld* headline news, December 5, 2003) and the 200-digit number known as RSA-200 (*MathWorld* headline news, May 10, 2005).

RSA numbers are composite numbers having exactly two prime factors (i.e., so-called semiprimes) that have been listed in the Factoring Challenge of RSA Security®.

While composite numbers are defined as numbers that can be written as a product of smaller numbers known as factors (for example, 6 = 2 x 3 is composite with factors 2 and 3), prime numbers have no such decomposition (for example, 7 does not have any factors other than 1 and itself). Prime factors therefore represent a fundamental (and unique) decomposition of a given positive integer. RSA numbers are special types of composite numbers particularly chosen to be difficult to factor, and they are identified by the number of digits they contain.

While RSA-640 is a *much* smaller number than the 7,816,230-digit monster Mersenne prime known as M_{42} (which is the largest prime number known), its factorization is significant because of the curious property that proving or disproving a number to be prime ("primality testing") seems to be much easier than actually identifying the factors of a number ("prime factorization"). Thus, while it is trivial to multiply two large numbers p and q together, it can be extremely difficult to determine the factors if only their product pq is given. With some ingenuity, this property can be used to create practical and efficient encryption systems for electronic data.

RSA Laboratories sponsors the RSA Factoring Challenge to encourage research into computational number theory and the practical difficulty of factoring large integers and also because it can be helpful for users of the RSA encryption public-key cryptography algorithm for choosing suitable key lengths for an appropriate level of security. A cash prize is awarded to the first person to factor each challenge number.

RSA numbers were originally spaced at intervals of 10 decimal digits between one and five hundred digits, and prizes were awarded according to a complicated formula. These original numbers were named according to the number of decimal digits, so RSA-100 was a hundred-digit number. As computers and algorithms became faster, the unfactored challenge numbers were removed from the prize list and replaced with a set of numbers with fixed cash prizes. At this point, the naming convention was also changed so that the trailing number indicates the number of digits in the *binary* representation of the number. Hence, RSA-640 has 640 binary digits, which translates to 193 digits in decimal.

While RSA-640 has slightly fewer digits than the previously factored RSA-200, its factorization carries the additional benefit of a cash reward of $20,000 from RSA Laboratories to the team responsible for this feat.

RSA numbers received widespread attention when a 129-digit number known as RSA-129 was used by R. Rivest, A. Shamir, and L. Adleman to publish one of the first public-key

messages together with a $100 reward for the message's decryption (Gardner 1977). Despite widespread belief at the time that the message encoded by RSA-129 would take millions of years to break, it was factored in 1994 using a distributed computation that harnessed networked computers spread around the globe performing a multiple polynomial quadratic sieve (Leutwyler 1994). The result of all the concentrated number crunching was decryption of the encoded message to yield the profound plain-text message "The magic words are squeamish ossifrage." (An ossifrage is a rare predatory vulture found in the mountains of Europe.)

Factorization of RSA-129 followed earlier factorizations of RSA-100, RSA-110, and RSA-120. The challenge numbers RSA-130, RSA-140, RSA-150, RSA-155, RSA-160, RSA-200, and RSA-576 were also subsequently factored between 1996 and May of 2005.

The factorization of the latest RSA number to fall involved "lattice" sieving done by J. Franke and T. Kleinjung using hardware at the Scientific Computing Institute and the Pure Mathematics Institute at Bonn University, Max Planck Institute of Mathematics in Bonn, and Experimental Mathematics Institute in Essen. The factorization of RSA-640 was accomplished using a prime factorization algorithm known as the general number field sieve. Sieving was done on 80 2.2-GHz Opteron CPUs and took 3 months. The matrix step was performed on a cluster of 80 2.2-GHz Opterons connected via a Gigabit network and took about 1.5 months. The two 97-digit factors found using this sieve are

1634733 6458092538 4844313388 3865090859 8417836700 3309231218
1110852389 3331001045 0815121211 8167511579
x
1900871 2816648221 1312685157 3935413975 4718967899 6851549366
6638539088 0271038021 0449895719 1261465571

These numbers can easily be multiplied to verify that their product is indeed equal to the original number.

As the following table shows, RSA-704 to RSA-2048 remain open, carrying awards from $30,000 to $200,000 to whoever is clever and persistent enough to track them down. A list of the open challenge numbers may be downloaded from RSA or in the form of a *Mathematica* package from the *MathWorld* package archive.

number	digits	prize	factored
RSA-100	100		Apr. 1991
RSA-110	110		Apr. 1992
RSA-120	120		Jun. 1993
RSA-129	129	$100	Apr. 1994
RSA-130	130		Apr. 10, 1996
RSA-140	140		Feb. 2, 1999
RSA-150	150		Apr. 16, 2004
RSA-155	155		Aug. 22, 1999
RSA-160	160		Apr. 1, 2003
RSA-200	200		May 9, 2005
RSA-576	174	$10,000	Dec. 3, 2003
RSA-640	193	$20,000	Nov. 4, 2005
RSA-704	212	$30,000	open
RSA-768	232	$50,000	open
RSA-896	270	$75,000	open
RSA-1024	309	$100,000	open
RSA-1536	463	$150,000	open
RSA-2048	617	$200,000	open

References

Franke, J. Email sent 4 Nov 2005.
http://www.crypto-world.com/announcements/rsa640.txt

Franke, J. Email to NMBRTHRY@LISTSERV.NODAK.EDU. 10 Nov 2005.
http://listserv.nodak.edu/cgi-bin/wa.exe?A1=ind0511&L=nmbrthry

Gardner, M. "Mathematical Games: A New Kind of Cipher That Would Take Millions of Years to Break." *Sci. Amer.* **237**, 120-124, Aug. 1977.

Leutwyler, K. "Superhack: Forty Quadrillion Years Early, a 129-Digit Code Is Broken." *Sci. Amer.* **271**, 17-20, 1994.

NFSNet: Large-Scale Distributed Factoring.
http://www.nfsnet.org

RSA Security®. "The New RSA Factoring Challenge."
http://www.rsasecurity.com/rsalabs/challenges/factoring

RSA Security®. "The RSA Challenge Numbers."
http://www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html

Weisstein, E. W. *Mathematica* package `RSANumbers.m`.

Weisstein, E. W. "RSA-576 Factored." *MathWorld* Headline News. Dec. 5, 2003.
http://mathworld.wolfram.com/news/2003-12-05/rsa

Weisstein, E. W. "RSA-200 Factored." *MathWorld* Headline News. May 10, 2005.
http://mathworld.wolfram.com/news/2005-05-10/rsa-200

- Für $c \in \mathbb{Z}, m \in \mathbb{N}_{\geq 2}$ beschreibt die Kongruenz $x \equiv c \bmod m$ die Menge

$$\langle c, m\rangle = c + m\cdot\mathbb{Z} = \{\ldots, c-2m, c-m, c, c+m, c+2m, \ldots\}$$

(arithmetische Progression)

- Betrachten nun Systeme von Kongruenzen

$$KS \;:\; \begin{cases} x \equiv c_1 \bmod m_1 & (\ \Leftrightarrow \langle c_1, m_1\rangle\ ) \\ x \equiv c_2 \bmod m_2 & (\ \Leftrightarrow \langle c_2, m_2\rangle\ ) \\ \qquad\vdots & \\ x \equiv c_k \bmod m_k & (\ \Leftrightarrow \langle c_k, m_k\rangle\ ) \end{cases}$$

mit nicht notwendig teilerfremden Moduln m_i,
$M := \mathtt{kgV}(m_1, m_2, \ldots, m_k)$.

- *Konsistenzproblem*: ist KS in $\mathbb{Z}$ lösbar,
d.h. *gibt es* eine ganze Zahl, die *alle* Kongruenzen erfüllt,
d.h. $\exists x \in \mathbb{Z}\, \forall i \in [1..k] : x \in \langle c_i, m_i\rangle$?

$$\bigcap_{1\leq i\leq k} \langle c_i, m_i\rangle \neq \emptyset \ ?$$

- *Überdeckungsproblem*: überdeckt KS ganz $\mathbb{Z}$,
d.h. erfüllt *jede* ganze Zahl *mindestens eine* Kongruenz,
d.h. $\forall x \in \mathbb{Z}\, \exists i \in [1..k] : x \in \langle c_i, m_i\rangle$?

$$\bigcup_{1\leq i\leq k} \langle c_i, m_i\rangle = \mathbb{Z} \ ?$$

Zum Konsistenzproblem (1):

- Aus dem Chinesischen Restesatz ergibt sich

$$\begin{aligned} KS \text{ ist lösbar} &\iff \bigcap_{1\leq i\leq k} \langle c_i, m_i\rangle \neq \emptyset \\ &\iff \bigcap_{1\leq i\leq k} \langle c_i, m_i\rangle \cap [0, M) \neq \emptyset \\ &\iff \sharp\left(\bigcap_{1\leq i\leq k} \langle c_i, m_i\rangle \cap [0, M)\right) = 1 \\ &\iff \forall_{1\leq i<j\leq k}\ \mathtt{ggT}(m_i, m_j) \mid c_i - c_j \end{aligned}$$

- Folglich ist das Konsistenzproblem *effizient* entscheidbar!
- Vorsicht! Es ist nicht sofort klar, dass man die Lösungen, wenn sie existieren, auch *effizient konstruieren* kann!

Zum Konsistenzproblem (2)

- Man kann ein konsistentes Kongruenzsystem

$$KS \;:\; \begin{cases} x \equiv c_1 \bmod m_1 & (\ \Leftrightarrow \langle c_1, m_1\rangle\ ) \\ \qquad\vdots & \\ x \equiv c_k \bmod m_k & (\ \Leftrightarrow \langle c_k, m_k\rangle\ ) \end{cases}$$

effizient (ohne Primfaktorisierung!) in ein äquivalentes System

$$KS' \;:\; \begin{cases} x \equiv c'_1 \bmod m'_1 & (\ \Leftrightarrow \langle c'_1, m'_1\rangle\ ) \\ \qquad\vdots & \\ x \equiv c'_\ell \bmod m'_\ell & (\ \Leftrightarrow \langle c'_\ell, m'_\ell\rangle\ ) \end{cases}$$

mit teilerfremden Moduln $m'_1, m'_2, \ldots, m'_\ell$ transformieren.

Zum Konsistenzproblem (3):

- Die gesuchte Transformation $KS \mapsto KS'$ ist einfach, wenn man die Primfaktorisierung der Moduln kennt.
- Man kann immer eine "Quasi-Faktorisierung" der Moduln $\{m_1, m_2, \ldots m_k\}$ *effizient konstruieren*:
  - Zu $\mathcal{M} = \{m_1, m_2, \ldots m_k\} \subset \mathbb{N}_{\geq 2}$ kann man $\mathcal{P} = \{\pi_1, \pi_2, \ldots, \pi_\ell\} \subset \mathbb{N}_{\geq 2}$ konstruieren mit:
    – die $\pi_j \in \mathcal{P}$ sind paarweise teilerfremd;
    – jedes $m_i \in \mathcal{M}$ hat eine eindeutige Darstellung

$$(*) \qquad m_i = \pi_1^{\alpha_1}\pi_2^{\alpha_2}\cdots\pi_\ell^{\alpha_\ell}$$

    mit $\alpha_j \in \mathbb{N}$.
  - Die Elemente von $\mathcal{P}$ müssen keine Primzahlen sein!
  - Sowohl die Konstruktion von $\mathcal{P}$ aus auch die Konstruktion der Darstellungen $(*)$ ist effizient!

Zum Konsistenzproblem (4):

- Zwei nützliche Funktionen:
  - Für $(m, n) \in \mathbb{N} \times \mathbb{N}$ sei

$$\texttt{split1}(m,n) = (u,v) \text{ mit } \begin{cases} m = u \cdot v \\ \texttt{ggT}(u,n) = 1 \\ p \mid v \Rightarrow p \mid n \quad (\forall p \text{ Primzahl}) \end{cases}$$

    `split1` ist durch diese Forderungen eindeutig definiert!
  - Für $(m, n) \in \mathbb{N} \times \mathbb{N}$ sei

$$\texttt{split2}(m,n) = (u,v) \text{ mit } \begin{cases} u \mid m \\ v \mid n \\ \texttt{ggT}(u,v) = 1 \\ u \cdot v = \texttt{kgV}(m,n) \end{cases}$$

    `split2` ist durch diese Forderungen noch nicht eindeutig definiert! Es wird eindeutig, wenn man noch verlangt, dass u maximal und v minimal mit diesen Eigenschaften sind.

Zum Konsistenzproblem (5):

- `split1` und `split2` lassen sich in Bezug auf die Primfaktorisierungen

$$m = \prod_{p \text{ prim}} p^{\alpha_p}, \quad n = \prod_{p \text{ prim}} p^{\beta_p}$$

  einfach beschreiben:
  - Ist $\texttt{split1}(m,n) = (u,v)$, so gilt

$$u = \prod_{\substack{p \text{ prim} \\ p \nmid n}} p^{\alpha_p}, \quad v = \prod_{\substack{p \text{ prim} \\ p \mid n}} p^{\alpha_p}.$$

  - Ist $\texttt{split2}(m,n) = (u,v)$, so gilt

$$u = \prod_{\substack{p \text{ prim} \\ \beta_p \leq \alpha_p}} p^{\alpha_p}, \quad v = \prod_{\substack{p \text{ prim} \\ \beta_p > \alpha_p}} p^{\beta_p}.$$

Zum Konsistenzproblem (6):

- Beispiele:

$\texttt{split1}(15,28) = (15,1)$	$\texttt{split2}(15,28) = (15,28)$
$\texttt{split1}(45,75) = (1,45)$	$\texttt{split2}(45,75) = (9,25)$
$\texttt{split1}(36,14) = (9,4)$	$\texttt{split2}(36,14) = (36,7)$

- Wichtig: $\texttt{split1}(m,n)$ und $\texttt{split2}(m,n)$ lassen sich effizient berechnen, also ohne Rückgriff auf die Primfaktorisierungen der Argumente, nur mit ggT und Division!
- Ein konsistentes Kongruenzsystem

$$x \equiv a \bmod m, x \equiv b \bmod n$$

  mit $\texttt{split2}(m,n) = (u,v)$ ist äquivalent zu

$$x \equiv a \bmod u, x \equiv b \bmod v.$$

- Die allgemeine Aussage von (2) lässt sich mit Hilfe von (3) analog zum Fall $k = 2$ behandeln.

Zum Überdeckungsproblem (1):

- Aus dem Chinesischen Restesatz ergibt sich

$$KS \text{ überdeckt } \mathbb{Z} \iff \bigcup_{1\le i\le k} \langle c_i, m_i\rangle = \mathbb{Z}$$
$$\iff \bigcap_{1\le i\le k} \langle c_i, m_i\rangle \supseteq [0, M)$$

- Dieses Problem ist co-NP-vollständig!
  Es ist unter dem Namen Simultaneuos Incongruences (SI) bekannt (Stockmeyer, Meyer 1973; Garey, Johnson 1979).

Zum Überdeckungsproblem (2):

- Ziel: Reduktion 3-Sat $\to$ SI
  - $Y_n = \{y_1, y_2, \ldots, y_n\}$: boolesche Variable
  - $Y \cup \overline{Y} = \{y_1, y_2, \ldots, y_n\} \cup \{\overline{y_1}, \overline{y_2}, \ldots, \overline{y_n}\}$: Literale
  - 3-Klauseln: Disjunktionen von 3 Literalen

$$C = \lambda_a \vee \lambda_b \vee \lambda_c,$$

    wobei $1 \le a < b < c \le n$ mit $\lambda_i \in \{y_i, \overline{y_i}\}$ $(1 \le i \le n)$.
  - AL-Formeln in 3-CNF: Konjunktionen von 3-Klauseln:

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

    wobei $C_j = \lambda_a^j \vee \lambda_b^j \vee \lambda_c^j$, $(1 \le j \le m)$.
- 3-Sat : Erfüllbarkeit von AL-Formeln F in CNF mit 3 Literalen pro Klausel
- Theorem von Cook: 3-Sat ist ein NP-vollständiges Problem.

Zum Überdeckungsproblem (2):

- $p_1, p_2, \ldots, p_n$: die ersten n Primzahlen
  $(p_1 = 2, p_2 = 3, p_3 = 5, \ldots)$,
  $P_n = p_1\, p_2 \cdots p_n$.
- Zu $\mathbf{e} = (e_1, e_2, \ldots, e_n) \in \mathbb{B}^n$ sei $x^{\mathbf{e}}$ die eindeutig bestimmte Lösung des Kongruenzsystems

$$\begin{cases} x \equiv e_1 \bmod p_1 \\ x \equiv e_2 \bmod p_2 \\ \quad\vdots \\ x \equiv e_n \bmod p_n \end{cases}$$

mit $0 \le x < P_n$.

Zum Überdeckungsproblem (3):

- Eine Zahl x mit $0 \le x < P_n$ ist numerische Codierung einer Bewertung, d.h. $x = x^{\mathbf{e}}$ für ein $\mathbf{e} \in \mathbb{B}^n$, genau dann, wenn

$$(S_n) \quad \begin{cases} x \not\equiv 2 \bmod 3 \\ x \not\equiv 2, 3, 4 \bmod 5 \\ \quad\vdots \\ x \not\equiv 2, 3, \ldots, p_n - 1 \bmod p_n \end{cases}$$

- Beispiel $n = 3$:

$\mathbf{e}$	000	100	010	001	110	101	011	111
$x^{\mathbf{e}}$	0	15	10	6	25	21	16	1

Zum Überdeckungsproblem (4):

- Für jedes $1 \leq i \leq n$ sei

$$[y_i] : x \equiv 0 \bmod p_i, \qquad [\overline{y_i}] : x \equiv 1 \bmod p_i$$

- Zu jeder Klausel $C = \lambda_a \vee \lambda_b \vee \lambda_c$ sei $x^C \in [0 \,..\, p_a p_b p_c)$ die eindeutig bestimmte simultane Lösung von

$$[\lambda_a], [\lambda_b], [\lambda_c]$$

- Für alle $\mathbf{e} \in \mathbb{B}^n$ und alle Klauseln C_j gilt:

$$\mathbf{e} \models C_j = \lambda_a^j \vee \lambda_b^j \vee \lambda_c^j \iff x^{\mathbf{e}} \not\equiv x^{C_j} \bmod p_a^j p_b^j p_c^j$$

- Beispiel:

$$C = y_1 \vee \overline{y_2} \vee \overline{y_3} \quad \leftrightarrow \quad x^C \equiv \begin{cases} 0 \bmod 2 \\ 1 \bmod 3 \\ 1 \bmod 5 \end{cases} \quad \leftrightarrow \quad x^C = 16$$

Zum Überdeckungsproblem (5):

- Für alle $\mathbf{e} \in \mathbb{B}^n$ und alle Formeln $F = C_1 \wedge \ldots \wedge C_m$ gilt:

$$\mathbf{e} \models F \iff (T_F) \begin{cases} x^{\mathbf{e}} \not\equiv x^{C_1} \bmod p_a^1 p_b^1 p_c^1 \\ \quad \vdots \\ x^{\mathbf{e}} \not\equiv x^{C_m} \bmod p_a^m p_b^m p_c^m \end{cases}$$

- Für alle Formeln $F = C_1 \wedge \ldots \wedge C_m$ gilt:

$$\begin{aligned} F \text{ ist erfüllbar} &\iff \exists\, \mathbf{e} \in \mathbb{B}^n : x^{\mathbf{e}} \models (T_F) \\ &\iff \exists_{0 \leq x < P_n} : x \models (S_n) \text{ und } (T_F) \end{aligned}$$

- Bekannte Aussagen über die Grösse von p_n und Effizienz der Lösbarkeit von Kongruenzsystemen (Konstruktion der x^{C_i}) zeigen, dass dies eine polynomielle Reduktion ist.

Zum Überdeckungsproblem (6):

Beispiel einer erfüllbaren Formel $F = C_1 \wedge C_2 \wedge \cdots \wedge C_8$:

$$\begin{array}{lrcl}
C_1: & \overline{y_1} \vee \overline{y_2} \vee \overline{y_3} & \leftrightarrow & x^{C_1} \equiv 1 \bmod 30 \\
C_2: & \overline{y_1} \vee \overline{y_2} \vee y_3 & \leftrightarrow & x^{C_2} \equiv 25 \bmod 30 \\
C_3: & y_1 \vee \overline{y_2} \vee y_4 & \leftrightarrow & x^{C_3} \equiv 28 \bmod 42 \\
C_4: & \overline{y_1} \vee y_2 \vee \overline{y_4} & \leftrightarrow & x^{C_4} \equiv 15 \bmod 42 \\
C_5: & \overline{y_1} \vee \overline{y_2} \vee \overline{y_4} & \leftrightarrow & x^{C_5} \equiv 1 \bmod 42 \\
C_6: & y_1 \vee y_3 \vee \overline{y_4} & \leftrightarrow & x^{C_6} \equiv 50 \bmod 70 \\
C_7: & y_1 \vee \overline{y_3} \vee \overline{y_4} & \leftrightarrow & x^{C_7} \equiv 36 \bmod 70 \\
C_8: & y_2 \vee y_3 \vee y_4 & \leftrightarrow & x^{C_8} \equiv 0 \bmod 105
\end{array}$$

Bewertungen als Zahlen mod 210:

$\mathbf{e}$	0000	1000	0100	1100	0010	1010	0110	1110
$x^{\mathbf{e}}$	0	105	70	175	126	21	196	91
$\mathbf{e}$	0001	1001	0101	1101	0011	1011	0111	1111
$x^{\mathbf{e}}$	120	15	190	85	36	141	106	1

$(y_1, y_2, y_3, y_4) = 1010$ erfüllt F: 21 ist
$\not\equiv 1, 25 \bmod 30, \not\equiv 1, 15, 28 \bmod 42, \not\equiv 50, 36 \bmod 70, \not\equiv 0 \bmod 105$

Zum Überdeckungsproblem (7):

Beispiel einer unerfüllbaren Formel $F = C_1 \wedge C_2 \wedge \cdots \wedge C_8$:

$$\begin{array}{lrcl}
C_1: & \overline{y_1} \vee \overline{y_2} \vee \overline{y_3} & \leftrightarrow & x^{C_1} \equiv 1 \bmod 30 \\
C_2: & \overline{y_1} \vee \overline{y_2} \vee y_3 & \leftrightarrow & x^{C_2} \equiv 25 \bmod 30 \\
C_3: & y_1 \vee \overline{y_2} \vee y_4 & \leftrightarrow & x^{C_3} \equiv 28 \bmod 42 \\
C_4: & \overline{y_1} \vee y_2 \vee \overline{y_4} & \leftrightarrow & x^{C_4} \equiv 15 \bmod 42 \\
C_5: & y_1 \vee y_3 \vee \overline{y_4} & \leftrightarrow & x^{C_5} \equiv 50 \bmod 70 \\
C_6: & y_1 \vee \overline{y_3} \vee \overline{y_4} & \leftrightarrow & x^{C_6} \equiv 36 \bmod 70 \\
C_7: & y_2 \vee y_3 \vee y_4 & \leftrightarrow & x^{C_7} \equiv 0 \bmod 105 \\
C_8: & y_2 \vee \overline{y_3} \vee y_4 & \leftrightarrow & x^{C_8} \equiv 21 \bmod 105
\end{array}$$

$$\begin{array}{llll}
0000 \not\models C_7 \leftrightarrow 0 \equiv 0 \bmod 105 & 0001 \not\models C_5 \leftrightarrow 120 \equiv 50 \bmod 70 \\
1000 \not\models C_7 \leftrightarrow 105 \equiv 0 \bmod 105 & 1001 \not\models C_4 \leftrightarrow 15 \equiv 15 \bmod 42 \\
0100 \not\models C_3 \leftrightarrow 70 \equiv 28 \bmod 42 & 0101 \not\models C_5 \leftrightarrow 190 \equiv 50 \bmod 70 \\
1100 \not\models C_2 \leftrightarrow 175 \equiv 25 \bmod 30 & 1101 \not\models C_2 \leftrightarrow 85 \equiv 25 \bmod 30 \\
0010 \not\models C_8 \leftrightarrow 126 \equiv 21 \bmod 105 & 0011 \not\models C_6 \leftrightarrow 36 \equiv 36 \bmod 70 \\
1010 \not\models C_8 \leftrightarrow 21 \equiv 21 \bmod 105 & 1011 \not\models C_4 \leftrightarrow 141 \equiv 15 \bmod 42 \\
0110 \not\models C_3 \leftrightarrow 196 \equiv 28 \bmod 42 & 0111 \not\models C_6 \leftrightarrow 106 \equiv 36 \bmod 70 \\
1110 \not\models C_1 \leftrightarrow 91 \equiv 1 \bmod 30 & 1111 \not\models C_1 \leftrightarrow 1 \equiv 1 \bmod 30
\end{array}$$

Drei Anwendungen der schnellen Exponentiation

- Schnelle Exponentiation berechnet für ganze Zahlen a, e, n mit $n \geq 2, e \geq 0, 0 \leq a < n$ die Abbildung

$$(a, e, n) \mapsto a^e \pmod{n}$$

mit einem Aufwand von $\mathcal{O}\left((\lg e)(\lg n)^2\right)$ bit-Operationen. Die Umkehrabbildung

$$(n, a, a^e \pmod{n}) \mapsto e$$

bezeichnet man als *diskreten Logarithmus*.

- Das *key-exchange*-Schema von Diffie-Hellman
  - A und B verständigen sich öffentlich über
    - eine sehr grosse (Prim-)Zahl p
    - eine Zahl b mit $2 \leq b < p$
  - A wählt eine sehr grosse Zahl m
    B wählt eine sehr grosse Zahl n
  - A berechnet $b^m \pmod{p}$ und schickt dies an B
    B berechnet $b^n \pmod{p}$ und schickt dies an A
  - A berechnet $(b^n \pmod{p})^m \pmod{p} = b^{mn} \pmod{p}$
    B berechnet $(b^m \pmod{p})^n \pmod{p} = b^{mn} \pmod{p}$
  - A und B verfügen jetzt gemeinsam über die private Information (den "Schlüssel") $b^{mn} \pmod{p}$
  - Ein Gegner müsste, um diesen Schlüssel zu berechnen, m oder n kennen, also z.B.

$$(p, b, b^m \pmod{p}) \mapsto m$$

    (schnell) berechnen können — und dies ist ein Fall des "diskreten Logarithmus"

- Das Kryptosystem von Shamir
  - A und B verständigen sich öffentlich über eine sehr grosse Primzahl p
  - A will eine Nachricht M mit $1 < M < p$ an B schicken, die geheim bleiben soll
  - A wählt (privat) eine Zahl $1 < a < p-1$ mit $\mathrm{ggT}(a, p-1) = 1$ und berechnet a' mit $a \cdot a' \equiv 1 \pmod{p-1}$
    B wählt (privat) eine Zahl $1 < b < p-1$ mit $\mathrm{ggT}(b, p-1) = 1$ und berechnet b' mit $b \cdot b' \equiv 1 \pmod{p-1}$
  - Der Nachrichtenaustausch geht so vor sich:
    1. A schickt an B : $C := M^a \pmod{p}$
    2. B schickt an A : $D := C^b \pmod{p}$
    3. A schickt an B : $E := D^{a'} \pmod{p}$
  - B berechnet nun : $F := E^{b'} \pmod{p}$
  - Es gilt nun $F = M$, denn es ist

$$F = E^{b'} = D^{a'b'} = C^{ba'b'} = M^{aba'b'} = \left(M^{aa'}\right)^{bb'} \pmod{p}$$

    und die Behauptung folgt aus dem Satz von Euler-Fermat, denn

$$a \cdot a' \equiv 1 \pmod{\phi(p)} \quad \text{und} \quad b \cdot b' \equiv 1 \pmod{\phi(p)}$$

  - Um M zu ermitteln, müsste ein Gegner diskrete Logarithmen (schnell) berechnen können.

Bemerkung: "sehr gross" bedeutet für praktische Zwecke in diesem Kontext bspw. 10^{200}

- Das *public-key* Kryptosystem von ElGamal
  - Alle Teilnehmer an dem System verständigen sich öffentlich auf eine gemeinsam zu benutzende grosse Primzahl p und eine Primitivwurzel modulo p, also eine Zahl g mit $1 < g < p$ derart, dass jede Zahl a mit $1 \leq a < p$ auf genau eine Weise als $a = g^e$ mit $0 \leq e < p-1$ geschrieben werden kann. Anders gesagt: g ist ein erzeugendes Element der zyklischen Gruppe U_p.
  - Jeder Teilnehmer A an dem System wählt eine Zahl x_A mit $1 \leq x_A \leq p-1$ als *privaten* Schlüssel
  - Jeder Teilnehmer berechnet $y_A := g^{x_A} \pmod{p}$ und gibt diese Zahl als seinen *öffentlichen* Schlüssel bekannt
  - Wenn Teilnehmer A an Teilnehmer B eine geheimzuhaltende Nachricht schicken will, geht er so vor
    1. A wählt eine (zufällige) Zahl k mit $1 \leq k < p$
    2. A berechnet den "Schlüssel" $K := y_B^k \pmod{p}$
    3. A schickt an B das Paar (C_1, C_2) von Zahlen mit
    $$C_1 = g^k \pmod{p} \quad \text{und} \quad C_2 = K \cdot M \pmod{p}$$
  - Empfänger B geht folgendermassen vor
    1. B berechnet K mittels
    $$K = y_B^k = g^{x_B k} = (g^k)^{x_B} = C_1^{x_B} \pmod{p}$$
    2. B ermittelt M durch Division (modulo P) von C_2 durch K
  - Auch die Sicherheit dieses Systems beruht auf der (vermuteten) Schwierigkeit, diskrete Logarithmen zu berechnen.

In der Arbeit

- E. Berlekamp, R. McEliece, H. van Tilborg, On the inherent intractability of certain coding problems, *IEEE Transactions on Information Theory* 24:384-386,1978.

zeigen die Autoren, dass gewisse Anforderungen, die bei der Decodierung linearer Codes ganz natürlich auftreten, leider Inkarnationen von NP-vollständigen Problemen sind,

- $A \in \mathbb{B}^{t\times s}$ sei $(t \times s)$-Matrix ber dem booleschen Ring $\mathbb{B}$.
- $\widetilde{A} \in \mathbb{B}^{t(s+1)\times s(t+1)+t}$ sei gegeben durch

$$\widetilde{A} = \left[\begin{array}{c|c} A & \\ E_s & \\ E_s & E_{st+t} \\ \vdots & \\ E_s & \end{array}\right]$$

wobei E_s die (s x s)-Einheitsmatrix ist.

- Für $0 \leq k \leq s(t+1)+t$ seien q, r durch die Divisionseigenschaft von $\mathbb{Z}$:

$$k = q \cdot (t+1) + r \quad \text{mit} \quad 0 \leq q \leq s, 0 \leq r \leq t$$

- Die beiden folgenden Aussagen sind äquivalent:
  1. $\exists \mathbf{u} \in \mathbb{B}^s$ mit $\|\mathbf{u}\| = q, \|A \cdot \mathbf{u}\| = r$.
  2. $\exists \mathbf{v} \in \mathbb{B}^{s(t+1)+t}$ mit $\|v\| = k, \|\widetilde{A} \cdot \mathbf{v}\| = 0$.

- Beweis: Ist $\mathbf{v} \in \mathbb{B}^{s(t+1)+t}$ Spaltenvektor der Länge $s(t+1)+t$ und schreibt man

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}' \\ \mathbf{v}'' \end{bmatrix} \quad \text{mit} \quad \mathbf{v}' \in \mathbb{B}^s, \mathbf{v}'' \in \mathbb{B}^{st+t},$$

so gilt

$$\widetilde{A} \cdot \mathbf{v} = \begin{bmatrix} A \cdot \mathbf{v}' \\ \mathbf{v}' \\ \vdots \\ \mathbf{v} \end{bmatrix} \oplus \mathbf{v}''$$

1. $\Rightarrow$ 2. Existiert ein $\mathbf{u}$ wie angegeben, so setzt man

$$\mathbf{v} = \begin{bmatrix} \mathbf{u} \\ A\cdot\mathbf{u} \\ \mathbf{u} \\ \vdots \\ \mathbf{u} \end{bmatrix} \in \mathbb{B}^{s(t+1)+t}, \text{ d.h. } \mathbf{v}' = \mathbf{u}, \mathbf{v}'' = \begin{bmatrix} A\cdot\mathbf{u} \\ \mathbf{u} \\ \vdots \\ \mathbf{u} \end{bmatrix}$$

und erhält nach obiger Bemerkung

$$\widetilde{A}\cdot\mathbf{v} = \begin{bmatrix} A\cdot\mathbf{u} \\ \mathbf{u} \\ \vdots \\ \mathbf{u} \end{bmatrix} \oplus \begin{bmatrix} A\cdot\mathbf{u} \\ \mathbf{u} \\ \vdots \\ \mathbf{u} \end{bmatrix} = \mathbf{0}$$

Ausserdem gilt

$$\|\mathbf{v}\| = \|A\cdot\mathbf{u}\| + (t+1)\|\mathbf{u}\| = k.$$

2. $\Rightarrow$ 1. Existiert ein $\mathbf{v}$ wie angegeben, so folgt aus

$$\mathbf{0} = \widetilde{A}\cdot\mathbf{v} = \begin{bmatrix} A\cdot\mathbf{v}' \\ \mathbf{v}' \\ \vdots \\ \mathbf{v}' \end{bmatrix} \oplus \mathbf{v}'', \text{ dass } \mathbf{v}'' = \begin{bmatrix} A\cdot\mathbf{v}' \\ \mathbf{v}' \\ \vdots \\ \mathbf{v}' \end{bmatrix}.$$

Somit ist $\|\mathbf{v}''\| = \|A\cdot\mathbf{v}'\| + t\cdot\|\mathbf{v}'\|$ und daher

$$k = \|\mathbf{v}\| = \|\mathbf{v}'\| + \|\mathbf{v}''\| = (t+1)\cdot\|\mathbf{v}'\| + \|A\cdot\mathbf{v}'\|.$$

Wegen $0 \le A\cdot\mathbf{v}' \le t$ und der Divisionsbeziehung zwischen k, t, q und r folgt $\|\mathbf{v}'\| = q$ und $\|A\cdot\mathbf{v}'\| = r$.

- 3-dim Matching (3DM)
  - *Instanzen*: Eine Menge T und eine Menge von Tripeln $S \subseteq T \times T \times T$ mit $s = \sharp S \ge \sharp T = t$.
  - *Frage*: Gibt es $M \subseteq S$ mit $\sharp M = t$, wobei sich je zwei verschiedene Tripel aus M in allen drei Komponenten unterscheiden sollen.
    Anders gesagt: projiziert man die Tripel aus M auf ihre drei Komponenten, so tritt jedes Element von T in jeder der drei Komponentnen genau einmal auf.
  - *Kommentar*: (3DM) ist ein wohlbekanntes $\mathcal{NP}$-vollständiges Problem. Garey,Johnson *Computers and Intractability*, Freeman 1979, oder Papadimitriou, *Computational Complexity*, Addison-Wesley 1994.

- Beispiel mit $T = \{1,2,3,4\}$
  - $S = \{(1,2,2),(2,4,3),(3,1,4),(3,3,4),(4,1,2),(4,3,1)\}$ hat das Matching
    $$M = \{(1,2,2),(2,4,3),(3,1,4),(4,3,1)\}.$$
  - $S = \{(1,3,1),(1,4,2),(2,1,4),(2,3,3),(3,4,4),(4,2,3)\}$ hat kein Matching!

- Linear Decoding (LD)
  - *Instanzen*: Matrix $H \in \mathbb{B}^{s,t}$, Vektor $\mathbf{s} \in \mathbb{B}^s$, Zahl $w \in \mathbb{N}$.
  - *Frage*: Gibt es einen Vektor $\mathbf{x} \in \mathbb{B}^t$ mit $H \cdot \mathbf{x} = \mathbf{s}$ und mit $\|\mathbf{x}\| \leq w$?
  - *Kommentar*: Der Vektor $\mathbf{s}$ spielt die Rolle des Syndroms beim Decodieren linearer Codes. Dabei sucht man unter allen Vektoren, die dasselbe Syndrom $\mathbf{s}$ liefern einen Vektor mit minimalem Gewicht: dies ist im Sinne der üblichen *maximum-likelihood-Decodierung* der wahrscheinlichste Fehlervektor. Kann man LD effizient lösen, so kann man durch systematisches Probieren mit $w = 1, 2, \ldots$ auch den Fehlervektor von minimalem Gewicht finden, also das Decodierungsproblem für lineare Codes lösen.

- Exact Minimum Distance (EMD)
  - *Instanzen*: Matrix $H \in \mathbb{B}^{s,t}$, Zahl $w \in \mathbb{N}$.
  - *Frage*: Gibt es einen Vektor $\mathbf{x} \in \mathbb{B}^t$ mit $H \cdot \mathbf{x} = \mathbf{0} \in \mathbb{B}^s$ und $\|\mathbf{x}\| = w$?
  - Für einen linearen Code, gegeben durch die Kontrollmatrix H ist die Minimaldistanz das minimale Gewicht der vom Nullvektor verschiedenen Codevektoren. Gefragt wird hier, ob es einen Codevektor vom Gewicht w in dem durch H definierten Code gibt.
  - *Kommentar*:Es ist (scheinbar) nicht bekannt, ob das analoge Problem, aber mit der Anforderung $\|\mathbf{x}\| \leq w$, $\mathcal{NP}$-vollständig ist.

- Offensichtlich: 3DM, LD, EMD $\in \mathcal{NP}$
- Zeigen: 3DM $\leq_p$ LD und 3DM $\leq_p$ LD
- Damit sind auch LD und EMD $\mathcal{NP}$-vollständig.

- (T, S) sei Instanz von 3DM mit $T = \{1, 2, \ldots, t\}$.
- Zuordnung
$$T \ni a \mapsto \vec{\mathbf{e}}_a \in \mathbb{B}^t$$
(als Spaltenvektor geschrieben)
- Zuordnung
$$T \times T \times T \ni \mathbf{a} = (a_1, a_2, a_3) \mapsto \vec{\mathbf{a}} = \begin{bmatrix} \vec{\mathbf{e}}_{a_1} \\ \vec{\mathbf{e}}_{a_2} \\ \vec{\mathbf{e}}_{a_3} \end{bmatrix} \in \mathbb{B}^{3t}$$
- Zuordnung
$$S = \{\mathbf{a}, \mathbf{b}, \ldots, \mathbf{z}\} \subseteq T \times T \times T \mapsto A_S = \begin{bmatrix} \vec{\mathbf{a}} & \vec{\mathbf{b}} & \ldots & \vec{\mathbf{z}} \end{bmatrix} \in \mathbb{B}^{3t \times s}$$

- Übersetzung der Beispiele in Matrizenschreibweise
  - $S = \{(1,2,2),(2,4,3),(3,1,4),(3,3,4),(4,1,2),(4,3,1)\}$
    $M = \{(1,2,2),(2,4,3),(3,1,4),(4,3,1)\}$

$$A_S = \begin{bmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&1&0&0\\ 0&0&0&0&1&1\\ 0&0&1&0&1&0\\ 1&0&0&0&0&0\\ 0&0&0&1&0&1\\ 0&1&0&0&0&0\\ 0&0&0&0&0&1\\ 1&0&0&0&1&0\\ 0&1&0&0&0&0\\ 0&0&1&1&0&0 \end{bmatrix} \qquad A_S \cdot \begin{bmatrix} x_1\\x_2\\x_3\\x_4\\x_5\\x_6 \end{bmatrix} = \begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1 \end{bmatrix}$$

Lösung $\mathbf{x}$ mit $\|\mathbf{x}\| \leq 4$: $\begin{bmatrix}1 & 1 & 1 & 0 & 1 & 0\end{bmatrix}^t$.

- $S = \{(1,3,1),(1,4,2),(2,1,4),(2,3,3),(3,4,4),(4,2,3)\}$

$$A_S = \begin{bmatrix} 1&1&0&0&0&0\\ 0&0&1&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\\ 0&0&1&0&0&0\\ 0&0&0&0&0&1\\ 1&0&0&1&0&0\\ 0&1&0&0&1&0\\ 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&0&1&0&1\\ 0&0&1&0&1&0 \end{bmatrix} \qquad A_S \cdot \begin{bmatrix} x_1\\x_2\\x_3\\x_4\\x_5\\x_6 \end{bmatrix} = \begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1 \end{bmatrix}$$

hat keine Lösung $\mathbf{x} \in \mathbb{B}^6$ mit $\|\mathbf{x}\| \leq 4$.

- Reduktion 3DM $\leq_p$ LD

$$(T,S) \mapsto (A_S, \vec{\mathbf{1}}, t)$$

Das Tripelsystem (T,S) enthält genau ein Matching M, wenn Instanz $(A_S, \vec{\mathbf{1}}, t)$ von LD eine Lösung hat, wobei $\vec{\mathbf{1}} \in \mathbb{B}^{3t}$ der Spaltenvektor mit 1en in allen Komponenten ist,

- Reduktion 3DM $\leq_p$ EMD
  Dies geht ganz analog, wobei statt A_S die Matrix

$$\widetilde{A_S} \in \mathbb{B}^{3t(s+1)\times 3t(s+1)+s}$$

verwendet wird, Obige Überlegung zu booleschen Matrizen mit $(t,q,r) = (3t,t,3t)$, also mit Zielvektor $\mathbf{x}$ vom Gewicht $k = t(3t+1)+3t = 3t2+4t$ zeigt, dass

$$(T,S) \mapsto (\widetilde{A_S}, 3t^2+4t)$$

die gesuchte Reduktion leistet,

Ausgangspunkt:

- NP-vollständige Probleme zur Konstruktion von *trapdoor*-Funktionen verwenden!
- Das (Syndrom-)Decodierungsproblem für lineare Codes ist NP-vollständig – also mutmasslich hoffnungslos schwierig!
- Man kennt grosse Klassen von linearen Codes (BCH-Codes, Goppa-Codes), für die es effiziente Decodieralgorithmen gibt!

Vorsicht! Andere Kryptosysteme auf der Basis von NP-vollständigen Problemen haben sich als nicht sicher erwiesen, so das auf dem Knapsack-Problem basierende System von Merkle und Hellman.

Szenario:

- Teilnehmer *Bob* wählt – privat – einen t-Fehler korrigierenden linearen Code $\mathcal{C}$, für den es einen effizienten Decodieralgorithmus gibt.
- *Bob* "verdreht" (*scramble*) diesen Code $\mathcal{C}$ mit einer nur ihm bekannten reversiblen Transformationen T zu einem linearen Code $\widetilde{\mathcal{C}} = T(\mathcal{C})$ mit mutmasslich NP-harter Decodierung, dessen Daten (Generatormatrix $\widetilde{G}$) er öffentlich bekannt gibt.
- *Alice* codiert Nachrichten für *Bob* mittels $\widetilde{G}$ und zufällig gewähltem Fehlervektor mit Gewicht $\leq t$.
- *Bob* rekonstruiert Nachricht mittels effizienter Decodierung für $\mathcal{C}$.
- *Eve* müsste das NP-harte Decodierungsproblem für $\widetilde{\mathcal{C}}$ lösen, um Nachricht zu erfahren!

Realisierung (setup):

- *Bob*
  - wählt einen t-Fehler korrigierenden linearen $[n, k]$-Code $\mathcal{C}$ mit effizienter Decodierung — z.B. Goppa-Code mit Parametern

    $$n = 2^m, k = n - m \cdot t, d \geq 2t + 1,$$

    realistischerweiseweise etwa (McEliece)

    $$m = 10, n = 2^{10} = 1024, t = 50, k = 524.$$

    $G \in \mathbb{B}^{k \times n}$ sei Generatormatrix für diesen Code.
    NB: es gibt *viele* Goppa-Codes mit diesen Parametern!
  - wählt ferner eine invertierbare Matrix ("scrambler") $S \in \mathbb{B}^{k \times k}$ und eine Permutationsmatrix $P \in \mathbb{B}^{n \times n}$.
  - berechnet $\widetilde{G} = S \cdot G \cdot P$ und gibt diese Matrix als öffentlichen Schlüssel bekannt.

Beachte:

- Die Matrix $\widetilde{G} = S \cdot G \cdot P$ ist Generatormatrix eines linearen Codes $\widetilde{\mathcal{C}}$
  - mit den gleichen Parametern $[n, k, d]$ wie $\mathcal{C}$
  - ohne (für Angreifer) erkennbare Struktur:
    für $\widetilde{\mathcal{C}}$ kommt nur Syndrom-Decodierung in Frage
    – und dies ist ein NP-vollständiges Problem!

Realisierung (Übertragung)

- *Alice* will $\mathbf{a} \in \mathbb{B}^k$ sicher zu *Bob* übertragen. Sie
  - wählt zufällig einen Vektor $\mathbf{f} \in \mathbb{B}^n$ mit $\|\mathbf{f}\| \leq t$.
  - berechnet
$$\mathbf{b} = \mathbf{a} \cdot \widetilde{G} \oplus \mathbf{f}$$
    und überträgt $\mathbf{b}$ zu *Bob*.
- *Bob*
  - berechnet
$$\mathbf{y} = \mathbf{b} \cdot P^{-1} = \underbrace{\mathbf{a} \cdot S}_{\mathbf{x}} \cdot G \oplus \underbrace{\mathbf{f} \cdot P^{-1}}_{\mathbf{z}}$$
  - beachtet $\mathbf{x} \cdot G \in \mathcal{C}$ und $\|\mathbf{z}\| = \|\mathbf{f}\| \leq t$ und kann aus $\mathbf{y}$ effizient $\mathbf{x} \cdot G \in \mathcal{C}$ berechnen (Decodierungsalgorithmus für $\mathcal{C}$!!)
    — und somit auch $\mathbf{x}$.
  - berechnet $\mathbf{a} = \mathbf{x} \cdot S^{-1}$.

Realisierung (Angriff)

- *Eve*
  - steht vor dem Problem, aus Kenntnis von $\mathbf{b}$ und $\widetilde{G}$ die ursprüngliche Nachricht $\mathbf{a}$ zu berechnen, wobei nur $d(\mathbf{a} \cdot \widetilde{G}, \mathbf{b}) \leq t$ bekannt ist:
    – das ist das Decodierungsproblem für den Code $\widetilde{\mathcal{C}}$!
  - mit den Beispieldaten von oben:
$$\sum_{s=0}^{50} \binom{1024}{s} = 3.362\ldots \cdot 10^{85}$$

Beispiel: $\mathcal{C}$ = der [7,4]-Hamming-Code (systematische Form)

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\widetilde{G} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} \mathbf{a} &= \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \\ \mathbf{f} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{b} = \mathbf{a} \cdot \widetilde{G} \oplus \mathbf{f} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \\ \mathbf{y} = \mathbf{b} \cdot P^{-1} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{x} \cdot G &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \\ \mathbf{x} &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \\ \mathbf{a} = \mathbf{x} \cdot S^{-1} &= \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \end{aligned}$$

Kommentare:

- Dieses von R.J. McEliece vorgeschlagene Kryptosystem gilt als sicher (unter der Annahme $\mathbf{P} \neq \mathbf{NP}$ natürlich).
- Nachteile:
  - Die öffentlichen Schlüssel $\widetilde{G}$ sind sehr gross!
  - Verschlüsselte Nachrichten sind deutlich länger als der zu verschlüsselnde Text.
  - Verschlüsselung und Entschlüsselung sind nicht vertauschbar! Das System ist (in dieser Form) für Authentifizierung und Signaturen nicht geeignet.
- Literatur:
  - R.J. Mc Eliece: *A public-key cryptosystem based on algebraic coding theory*, TR Jet Propulsion Labs (1978).
  - F. Chabaud: *On the security of some crypotsystems based on algebraic codes* EURORYPT '94.
  - W. Tappe, L. Washington: *Introduction to Cryptography with Coding Theory*, 2nd. ed., Pearson, 2006.

Jean Baptiste Joseph Fourier (1768–1830)

Fourier-Transformation klassisch (z.B. Signalverarbeitung):

- FOURIERs Idee: Darstellung von (periodischen) Funktionen als Überlagerung von einfachen periodischen Schwingungen verschiedener Frequenz und Intensität
- Dualität: Schwingungsphänomene (Wellenausbreitung) lassen sich darstellen sowohl im "Ortsbereich" als auch im "Frequenzbereich"
- Fourier-Transformation: Übergang von einer Funktionsdarstellung im Ortsbereich zu einer Darstellung im Frequenzbereich (und umgekehrt)
- Wichtige Anwendung: Filterung

- Fouriertransformation

$$\mathcal{F} : f(x) \mapsto F(s) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i \cdot s \cdot x} dx = \mathcal{F}[f](s)$$

- Rücktransformation

$$\mathcal{F}_{\mathtt{inv}} : F(s) \mapsto f(x) = \int_{-\infty}^{\infty} F(s)\, e^{2\pi i \cdot s \cdot x} ds = \mathcal{F}_{\mathtt{inv}}[F](x)$$

- inverse Beziehung — spektrale Zerlegung

$$f(x) = \mathcal{F}_{\mathtt{inv}}[\mathcal{F}[f]](x) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(y)\, e^{-2\pi i \cdot s \cdot y} dy \right] e^{2\pi i \cdot s \cdot x} ds$$

Die Funktion $x \mapsto f(x)$ ist dargestellt als Überlagerung (Linearkombination) der periodischen Funktionen ("harmonischen Schwingungen") $x \mapsto e^{2\pi i \cdot s \cdot x}$ mit Frequenz $2\pi s$. Das innere Integral ist deren "Intensität".

- Faltung

$$(f(x), g(x)) \mapsto (f \otimes g)(x) = \int_{-\infty}^{\infty} f(y) \cdot g(x-y)\, dy$$

Faltungsoperationen verwendet man, um Merkmale der einer gegebenen Funktion f mittels geeigneter "Test"-Funktionen g zu erkennen und zu bewerten.

- Faltungstheorem

$$\mathcal{F}[f \otimes g](s) = \mathcal{F}[f](s) \cdot \mathcal{F}[g](s)$$

Konsequenz: die Faltung im "Ortsbereich" kann auch im "Frequenzbereich" durchgeführt werden.

Beweis des Faltungstheorems

$$\begin{aligned}
\mathcal{F}[f \otimes g](s) &= \mathcal{F}\left[\int_{-\infty}^{\infty} f(y)\cdot g(x-y)\,dy\right](s) \\
&= \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(y)\cdot g(x-y)\,dy\right] e^{-2\pi i\cdot s\cdot x}\,dx \\
&= \int_{-\infty}^{\infty} f(y)\left[\int_{-\infty}^{\infty} g(x-y)\,e^{-2\pi i\cdot s\cdot x}\,dx\right] dy \\
&= \int_{-\infty}^{\infty} f(y)\left[\int_{-\infty}^{\infty} g(z)\,e^{-2\pi i\cdot s\cdot z}e^{-2\pi i\cdot s\cdot y}\,dz\right] dy \\
&= \int_{-\infty}^{\infty} f(y)\left[\int_{-\infty}^{\infty} g(z)\,e^{-2\pi i\cdot s\cdot z}\,dz\right] e^{-2\pi i\cdot s\cdot y}\,dy \\
&= \mathcal{F}[f](s)\cdot\mathcal{F}[g](s)
\end{aligned}$$

- Schema des Faltungstheorems

	Faltung im Ortsbereich		punktweise Multiplikation im Frequenzbereich
$\mathcal{F}:$	$(f(x), g(x))$	$\longrightarrow_{\mathcal{F}}$	$(\mathcal{F}[f](s), \mathcal{F}[g](s))$
	$\downarrow \otimes$		$\downarrow \cdot$
$\mathcal{F}:$	$(f \otimes g)(x)$	$\longrightarrow_{\mathcal{F}_{\text{inv}}}$	$\mathcal{F}[f](s)\cdot\mathcal{F}[g](s)$

- typische Anwendung des Faltungstheorems

	Faltung im Ortsbereich		punktweise Multiplikation im Frequenzbereich
$\mathcal{F}:$	$(f(x), g(x))$	$\longrightarrow_{\mathcal{F}}$	$(\mathcal{F}[f](s), \mathcal{F}[g](s))$
	$\downarrow \otimes$		$\downarrow \cdot$
$\mathcal{F}_{\text{inv}}:$	$(f \otimes g)(x)$	$\longleftarrow_{\mathcal{F}_{\text{inv}}}$	$\mathcal{F}[f](s)\cdot\mathcal{F}[g](s)$

- Analogie: Darstellung von Polynomen
  - $\mathbb{K}[X]$: Ring der Polynome in einer Variablen X und mit Koeffizienten im Körper $\mathbb{K}$
  - übliche Koeffizienten-Darstellung

$$a(X) = \sum_{i=0}^{n} a_i X^i \quad \leftrightarrow \quad \langle a_0, a_1, a_2, \ldots, a_{n-1}, a_n\rangle \in \mathbb{K}^{n+1}$$

  - $\mathbb{K}[X]_{\leq n}$: Vektorraum der Polynome vom Grad $\leq n$ hat als Standardbasis die $n+1$ Polynome X^i $(0 \leq i \leq n)$
  - algebraische Sicht: $\langle a_0, a_1, a_2, \ldots, a_{n-1}, a_n\rangle$ ist die Darstellung von $a(X)$ bezüglich der Standardbasis
  - der Vektorraum $\mathbb{K}[X]_{\leq n}$ hat noch viele andere (und nützliche!) Basen

- Auswertung von Polynomen an einer Stelle $y \in \mathbb{K}$:

$$L_y : \mathbb{K}[X]_{\leq n} \to \mathbb{K} :$$
$$a(X) \mapsto a(y) = a_n\, y^n + a_{n-1}\, y^{n-1} + \cdots + a_1\, y + a_0$$
$$= (\ldots(a_n\, y + a_{n-1})\, y + \cdots + a_1)\, y + a_0$$

- praktisches Verfahren: Horner-Schema (optimal!)
  benötigt n Multiplikationen und n Additionen in $\mathbb{K}$
- algebraisch: die Auswertungsabbildung L_y an der Stelle $y \in \mathbb{K}$ ist ein Ring-Homomorphismus (Verträglichkeit mit Addition und Multiplikation)

- simultane Auswertung an mehreren (verschiedenen) Stellen $\mathbf{y} = (y_0, y_1, y_2, \ldots, y_n) \in \mathbb{K}^{n+1}$

$$L_{\mathbf{y}} : \mathbb{K}[X]_{\leq n} \to \mathbb{K}^{n+1} :$$
$$a(X) \mapsto \langle L_{y_0}(a(X)), L_{y_1}(a(X)), L_{y_2}(a(X)), \ldots, L_{y_n}(a(X))\rangle$$
$$= \langle a(y_0), a(y_1), a(y_2), \ldots, a(y_n)\rangle$$

- praktisches Verfahren: $(n+1)$-mal Horner-Schema
  benötigt $n(n+1)$ Multiplikationen und $n(n+1)$ Additionen in $\mathbb{K}$ (das ist nicht notwendig optimal!)
- algebraisch: die Auswertungsabbildung $L_{\mathbf{y}}$ an den Stellen $\mathbf{y} = (y_0, y_1, y_2, \ldots, y_n) \in \mathbb{K}^{n+1}$ ist ein Ring-Homomorphismus (Verträglichkeit mit Addition und Multiplikation)

- Rekonstruktion durch Interpolation
  ein Polynom n-ten Grades $a(X) = \sum_{i=0}^{n} a_i\, X^i$ ist durch seine Werte $a(y_i)\,(0 \leq i \leq n)$ an $n+1$ verschiedenen Stellen $\mathbf{y} = (y_0, y_1, y_2, \ldots, y_n)$ eindeutig bestimmt
  [Bem.: dies gilt über einem Körper mit mindestens $n+1$ Elementen]
  denn das lineare Gleichungssystem mit Vandermonde-Matrix

$$\underbrace{\begin{pmatrix} 1 & y_0 & y_0^2 & \ldots & y_0^n \\ 1 & y_1 & y_1^2 & \ldots & y_1^n \\ \vdots & \vdots & & \vdots & \\ 1 & y_n & y_n^2 & \ldots & y_n^n \end{pmatrix}}_{V_n(y_0, y_1, \ldots, y_n)} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a(y_0) \\ a(y_1) \\ \vdots \\ a(y_n) \end{pmatrix}$$

  hat eine eindeutig bestimmte Lösung wegen

$$\det V_n(y_0, y_1, \ldots, y_n) = \prod_{0 \leq i < j \leq n} (y_j - y_i) \neq 0$$

- Lösung des Gleichungssystems mittels Gauss-Elimination erfordert $O(n^3)$ Operationen in $\mathbb{K}$
- Effizienter: Interpolationsformel von Lagrange mit $O(n^2)$ Operationen

$$a(X) = \sum_{i=0}^{n} a(y_i) \cdot e_i(X)$$

  wobei

$$e_i(X) = \frac{\prod_{0 \leq j \leq n}^{j \neq i} (X - y_j)}{\prod_{0 \leq j \leq n}^{j \neq i} (y_i - y_j)} \quad \text{d.h.} \quad e_i(y_j) = \begin{cases} 1 & \text{falls } i = j \\ 0 & \text{falls } i \neq j \end{cases}$$

- Die Polynome $e_i(X)\ (0 \leq i \leq n)$ sind linear unabhängig und bilden eine Basis des Vektorraums $\mathbb{K}[X]_{\leq n}$.

- schematisch

$$\langle a_0, a_1, \ldots, a_{n-1}, a_n \rangle \underset{L_y^{-1}=\text{Interpol.}}{\overset{L_y=\text{Auswert.}}{\rightleftarrows}} \langle a(y_0), a(y_1), \ldots, a(y_{n-1}), a(y_n) \rangle$$

- Folgerung: L_y ist ein Isomorphismus von Vektorräumen

$$\mathbb{K}[X]_{\leq n} \quad \simeq \quad \underbrace{\mathbb{K} \times \mathbb{K} \times \cdots \times \mathbb{K}}_{n+1 \text{ Faktoren}}$$

- Für ein Objekt "Polynom vom Grad $\leq n$" sind
  - Koeffizientendarstellung

$$\langle a_0, a_1, \ldots, a_{n-1}, a_n \rangle$$

  - Wertedarstellung

$$\langle a(y_0), a(y_1), \ldots, a(y_{n-1}), a(y_n) \rangle$$

  Darstellungen in verschiedenen Basen des VR $\mathbb{K}[X]_{\leq n}$.
- Die Transformationen zwischen diesen Basen heissen "Auswertung" und "Interpolation".

Reminiszenz: Chinesischer Restesatz, modulare Arithmetik

- Die Beziehung

  Auswertung $\leftrightarrow$ Interpolation

  für Polynome erinnert nicht zufällig an den Chinesischen Restesatz für ganze Zahlen — es ist eine völlig analoge Situation im Bereich der Polynome
  - Für Polynome über einem Körper $\mathbb{K}$ gilt die Divisioneigenschaft

$$a(X) = b(X) \cdot q(X) + r(X)$$

    wobei $r(X) = 0$ oder $\deg r(X) < \deg b(X)$.
  - Man schreibt:

$$r(X) = a(X) \bmod b(X) \quad \text{und} \quad b(X) \,|\, a(X), \text{ falls } r(X) = 0$$

  - Grösste gemeinsame Teiler, Eindeutigkeit der "Prim-Zerlegung ("irreduzible" Polynome) funktionieren analog zu $\mathbb{Z}$.

- Algebraisch formuliert: $\mathbb{K}[X]$ ist ein "euklidischer Ring".
- Insbesondere: es gibt im Ring $\mathbb{K}[X]$ einen euklidischen Algorithmus zur Berechnung grösster gemeinsamer Teiler
- Alle Folgerungen (z.B. Bézout-Beziehung) ergeben sich daraus exakt wie im Falle des Ringes $\mathbb{Z}$.
- Der euklidische Algorithmus für Polynome ist der Algorithmus der Polynomarithmethik schlechthin — auch für viele Anwendungen (z.B. effiziente Decodierverfahren für fehlerkorrigierende "zyklische" Codes)
- Wenn Sie mehr wissen wollen: besuchen Sie meine Vorlesungen über "Computeralgebra"!

- Spezialfall der Polynomdivision: Auswertung
  - Auswertung eines Polynoms $a(X)$ an einer Stelle y ist gleichbedeutend mit Division von $a(X)$ durch $b(X) = X - y$:

$$a(X) = (X - y) \cdot q(X) + a(y)$$

d.h.

$$a(X) \bmod (X - y) = a(y)$$

  - Es gilt also

$$L_y \ : \ \mathbb{K}[X]_{\leq n} \to \mathbb{K} \ : \ a(X) \mapsto a(y) = a(X) \bmod (X - y)$$

- Konstruktion von "Faktorringen" des Polynomringes
  - Ansatz identisch zum Fall $\mathbb{Z} \to \mathbb{Z}_n : a \mapsto a \bmod n$
  - Jedes Polynom $b(X) \in \mathbb{K}[X]$ definiert eine Äquivalenzrelation (sogar 'Kongruenz")

$$f(X) \sim_{b(X)} g(X) \quad \leftrightarrow \quad b(X) \,|\, f(X) - g(X)$$

  - Ist der Grad deg $b(X) = n$, so kann man $\mathbb{K}[X]_{<n}$ mit einer Multiplikation versehen

$$(f(X), g(X)) \mapsto (f *_{b(X)} g)(X) = f(X) \cdot g(X) \bmod b(X)$$

  - Mit $*_{b(X)}$ und der üblichen Addition wird $\mathbb{K}[X]_{<n}$ zu einem Ring: $\mathbb{K}[X]/(b(X))$

- Ein "Chinesischer Restesatz" für simultane Kongruenzen von Polynomen gilt völlig analog zum Fall für ganze Zahlen:
  - Sind $a(X), b(X)$ teilerfremde Polynome, so gilt

$$\mathbb{K}[X]/(a(X) \cdot b(X)) \simeq \mathbb{K}[X]/(a(X)) \times \mathbb{K}/(b(X))$$

(Isomorphismus von Ringen)
  - Die algorithmische Realisierung des Isomorphismus geschieht wieder
    - Auswertung: mittels Polynomdivision
    - Interpolation: mittels euklidischem Algorithmus für Polynome
  - Entsprechend kann man die Aussage im Fall mehrer zueinander teilerfremder Polynome formulieren.

- Chinesischer Restesatz für Produkte von linearen Polynomen
  - Sind $y_0, y_1, \ldots, y_n \in \mathbb{K}$ verschiedene Elemente, so sind die Polynome $X - y_0, X - y_1, \ldots, X - y_n$ paarweise teilerfremd. Daher:

$$\underbrace{\mathbb{K}[X]/\prod_{i=0}^{n}(X - y_i)}_{\mathbb{K}[X]_{\leq n}} \simeq \underbrace{\underbrace{\mathbb{K}[X]/(X - y_0)}_{\simeq \mathbb{K}} \times \cdots \times \underbrace{\mathbb{K}[X]/(X - y_n)}_{\simeq \mathbb{K}}}_{\mathbb{K}^{n+1}}$$

als Isomorphismus von Ringen, der realisiert wird durch die simultane Auswertungsabbildung $L_{\mathbf{y}}$ mit $\mathbf{y} = (y_0, y_1, \ldots, y_n)$.
  - Diskrete Fouriertransformation ist ein Spezialfall hiervon!

- Polynommultiplikation mittels Auswertung und Interpolation
  - Gegeben Polynome $a(X), b(X) \in \mathbb{K}[X]_{<n}$ in Koeffizientendarstellung

$$a(X) = \sum_{i=0}^{n-1} a_i X^i \qquad b(X) = \sum_{i=0}^{n-1} b_i X^j$$

  - Berechne die Koeffizientendarstellung des Produkts

$$a(X) \cdot b(X) = c(X) = \sum_{i=0}^{2n-1} c_i X^i$$

$$c_i = \sum_{j=0}^{i} a_j \cdot b_{i-j} \ (0 \le i \le 2n-1)$$

  - als Operation auf Vektoren (Faltung, convolution)

$$\big(\underbrace{\langle a_0, \ldots, a_{n-1}\rangle}_{\mathbf{a}}, \underbrace{\langle b_0, \ldots, b_{n-1}\rangle}_{\mathbf{b}}\big) \quad \mapsto \quad \underbrace{\langle c_0, \ldots, c_{2n-1}\rangle}_{\mathbf{c}} = \mathbf{a} * \mathbf{b}$$

- traditionelle Lösung:
  Faltungsformel direkt berechnen$\longrightarrow O(n^2)$ Operationen in $\mathbb{K}$
- alternative Lösung mittels "modularer Arithmetik"
  - Polynome $a(X)$ und $b(X)$ an $2n$ Stützstellen $\mathbf{y} = (y_0, \ldots, y_{2n-1})$ auswerten

$$\mathbf{a} = \langle a_0, \ldots, a_{n-1}\rangle \mapsto L_{\mathbf{y}}(\mathbf{a}) = \langle a(y_0), \ldots a(y_{2n-1})\rangle$$
$$\mathbf{b} = \langle b_0, \ldots, b_{n-1}\rangle \mapsto L_{\mathbf{y}}(\mathbf{b}) = \langle b(y_0), \ldots b(y_{2n-1})\rangle$$

  - Multiplikation der Funktionswerte an den $2n$ Stützstellen y_i

$$c(y_i) = a(y_i) \cdot b(y_i) \ \ (0 \le i < 2n)$$

  - Rekonstruktion von $c(X)$ durch die Funktionswerte an den Stützstellen

$$\mathbf{c} = \langle c_0, \ldots, c_{2n-1}\rangle = L_{\mathbf{y}}^{-1}\langle c(y_0), \ldots, c(y_{2n-1})\rangle$$

  - Aufwand dieses Verfahrens: $O(n^2)$ Operationen in $\mathbb{K}$

- Schema der Polynommultiplikation mittels modularer Arithmetik

	Faltung im Koeffizientenbereich		komponentenweise Multiplikation im Wertebereich
Auswertung	$(\mathbf{a}, \mathbf{b})$	$\longrightarrow L_{\mathbf{y}}$	$(\langle a(y_i)\rangle, \langle b(y_i)\rangle)$
	$\downarrow *$		$\downarrow \cdot$
Interpolation	$\mathbf{c}$	$\longleftarrow L_{\mathbf{y}}^{-1}$	$\langle c(y_i)\rangle = \langle a(y_i) \cdot b(y_i)\rangle$

$\mathbb{K}[X]_{<n} \times \mathbb{K}[X]_{<n}$	$\longrightarrow L_{\mathbf{y}}$	$\mathbb{K}^{2n} \times \mathbb{K}^{2n}$
$\downarrow *$		$\downarrow \cdot$
$\mathbb{K}[X]_{<2n}$	$\longleftarrow L_{\mathbf{y}}^{-1}$	$\mathbb{K}^{2n}$

- Diskrete Fourier-Transformation
  - Ursprünglich Approximation der kontinuierlichen Fouriertransformation für numerische Berechnungen
  - Durchbruch bei der praktischen Verwendung durch Entdeckung der "Schnellen Fourier-Transformation" (FFT) durch Cooley und Tukey[1] — gehört seitdem zu den meistverwendeten Algorithmen überhaupt
  - Das FFT-Prinzip war schon Gauss bekannt (Notiz in seinem Tagebuch) und wurde auch später mehrmals entdeckt und publiziert[2]
  - Durchbruch aber erst als Computer-Verfahren
  - DFT bedeutet: Auswertung an "Einheitswurzeln" , d.h. Lösungen der Gleichungen $X^n = 1$ in $\mathbb{C}$

[1] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation 19 (1965)

[2] C. Runge, R. König *Vorlesungen über Numerisches Rechnen*, Springer (1924)

- Die Gleichung $X^n = 1$ hat in $\mathbb{C}$ genau n Lösungen

$$\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$$

wobei $\omega_n = e^{2\pi i/n} = \cos\left(\frac{2\pi}{n}\right) + i \cdot \sin\left(\frac{2\pi}{n}\right)$
("primitive" n-te Einheitswurzel)

- Diese "n-ten Einheitswurzeln" bilden eine zyklische Gruppe $\mathcal{R}_n$ der Ordnung n unter der Multiplikation.

$$(\mathbb{Z}_n, +) \to (\mathcal{R}_n, \cdot) : k \mapsto \omega_n^k$$

ist ein Isomorphismus.

- Aussagen über $\mathbb{Z}$ und $\mathbb{Z}_n$ übertragen sich auf die Gruppen $\mathcal{R}_n$
- $n|m \implies \mathcal{R}_n \subseteq \mathcal{R}_m$

- Beispiele
  - $n = 2 : \omega_2 = -1$

$$\mathcal{R}_2 = \{1, -1\}$$

  - $n = 3 : \omega_3 = (-1 + i \cdot \sqrt{3})/2$

$$\mathcal{R}_3 = \left\{1, \frac{-1 + i \cdot \sqrt{3}}{2}, \frac{-1 - i \cdot \sqrt{3}}{2}\right\}$$

  - $n = 4 : \omega_4 = i$

$$\mathcal{R}_4 = \{1, i, -1, -i\} = \mathcal{R}_2 \cup \{\pm i\}$$

  - $n = 5$:

$$\omega_5 = \frac{\sqrt{5} - 1 + i\sqrt{2}\sqrt{5 + \sqrt{5}}}{4}$$

$$\mathcal{R}_5 = \{1\} \cup \left\{\frac{\sqrt{5} - 1 \pm i\sqrt{2}\sqrt{5 + \sqrt{5}}}{4}, \frac{-\sqrt{5} - 1 \pm i\sqrt{2}\sqrt{5 - \sqrt{5}}}{4}\right\}$$

- $n = 6$:

$$\omega_6 = \frac{1 + i \cdot \sqrt{3}}{2}$$

$$\mathcal{R}_6 = \left\{1, \frac{1 + i \cdot \sqrt{3}}{2}, \frac{-1 + i \cdot \sqrt{3}}{2}, -1, \frac{-1 - i \cdot \sqrt{3}}{2}, \frac{1 - i \cdot \sqrt{3}}{2}\right\}$$

$$= \mathcal{R}_2 \cup \mathcal{R}_3 \cup \{\frac{1 \pm i \cdot \sqrt{3}}{2}\}$$

- $n = 7$

$$\mathcal{R}_7 = \{1\} \cup \left\{\cos\left(\frac{k\pi}{7}\right) \pm i \sin\left(\frac{k\pi}{7}\right) \; ; \; 1 \le k \le 3\right\}$$

- $n = 8$

$$\omega_8 = \frac{1 + i}{\sqrt{2}}$$

$$\mathcal{R}_8 = \mathcal{R}_4 \cup \{\omega_8, \omega_8^3, \omega_8^5, \omega_8^7\} = \mathcal{R}_4 \cup \left\{\pm\frac{1 \pm i}{\sqrt{2}}\right\}$$

- $n = 9$

$$\mathcal{R}_9 = \mathcal{R}_3 \cup \left\{\cos\left(\frac{k\pi}{9}\right) \pm i \sin\left(\frac{k\pi}{9}\right) \; ; \; k \in \{1, 2, 4\}\right\}$$

- $n = 10$

$$\mathcal{R}_{10} = \mathcal{R}_2 \cup \mathcal{R}_5 \cup \left\{ \cos\left(\frac{k\pi}{10}\right) \pm i \sin\left(\frac{k\pi}{10}\right) \; ; \; k \in \{1, 3\} \right\}$$

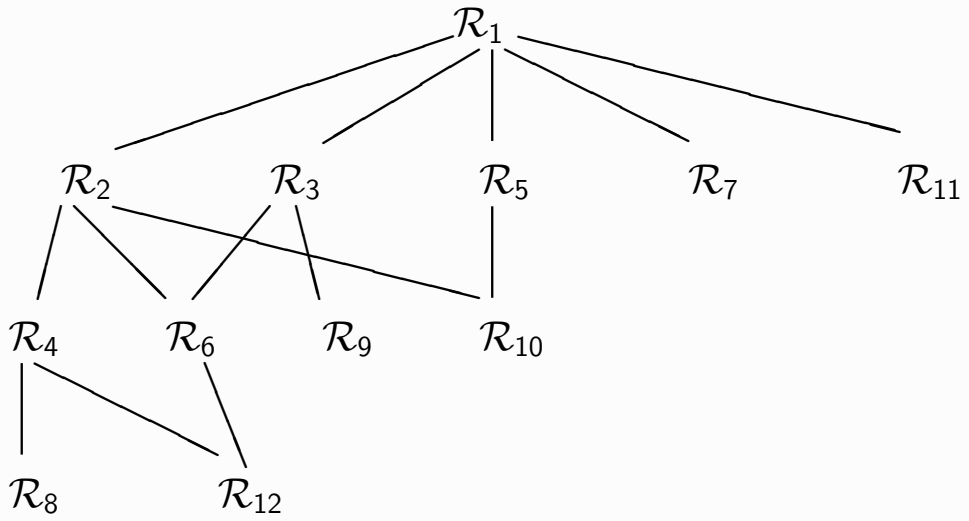
- $n = 11$

$$\mathcal{R}_{11} = \{1\} \cup \left\{ \cos\left(\frac{k\pi}{11}\right) \pm i \sin\left(\frac{k\pi}{11}\right) \; ; \; 1 \leq k \leq 5 \right\}$$

- $n = 12$

$$\omega_{12} = \frac{\sqrt{3} + i}{2}$$

$$\mathcal{R}_{12} = \mathcal{R}_4 \cup \mathcal{R}_6 \cup \left\{ \pm \frac{\sqrt{3} \pm i}{2} \right\}$$



- (Komplexe) Diskrete Fouriertransformation der Ordnung n:

$$DFT_{n,\omega} : \begin{cases} \langle a_0, a_1, \ldots, a_{n-1} \rangle & \rightarrow & \langle a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1}) \rangle \\ \mathbb{C}^n & \rightarrow & \mathbb{C}^n \end{cases}$$

  wobei $a(X) = \sum_{i=0}^{n-1} a_i X^i$

- $DFT_{n,\omega}$ ist simultane Auswertung von Polynomen $\in \mathbb{C}[X]_{<n}$ an den n-ten Einheitswurzeln

$$DFT_{n,\omega} = L_{\langle \omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1} \rangle}$$

- DFT als lineare Transformation

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_j \\ \vdots \\ a_{n-1} \end{pmatrix} \mapsto \underbrace{\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^j & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2j} & \ldots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & \omega_n^i & \omega_n^{2i} & \ldots & \omega_n^{ij} & \ldots & \omega_n^{i(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{j(n-1)} & \ldots & \omega_n^{(n-1)^2} \end{pmatrix}}_{V_n(\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1})} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_j \\ \vdots \\ a_{n-1} \end{pmatrix}$$

- Beachten: ω_n hat die Ordnung n in $\mathcal{R}_n$, also $\omega_n^{i \cdot j} = \omega_n^{i \cdot j \bmod n}$

- $n = 2$

$$DFT_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- $n = 3$

$$DFT_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega_3 & \omega_3^2 \\ 1 & \omega_3^2 & \omega_3^4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega_3 & \omega_3^2 \\ 1 & \omega_3^2 & \omega_3 \end{bmatrix}$$

mit $\omega_3 = \frac{-1+i\cdot\sqrt{3}}{2}, \omega_3^2 = \frac{-1-i\cdot\sqrt{3}}{2}$

- $n = 4$

$$DFT_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^0 & i^2 \\ 1 & i^3 & i^2 & i^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

- $n = 5$

$$DFT_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_5 & \omega_5^2 & \omega_5^3 & \omega_5^4 \\ 1 & \omega_5^2 & \omega_5^4 & \omega_5^6 & \omega_5^8 \\ 1 & \omega_5^3 & \omega_5^6 & \omega_5^9 & \omega_5^{12} \\ 1 & \omega_5^4 & \omega_5^8 & \omega_5^{12} & \omega_5^{16} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_5 & \omega_5^2 & \omega_5^3 & \omega_5^4 \\ 1 & \omega_5^2 & \omega_5^4 & \omega_5 & \omega_5^3 \\ 1 & \omega_5^3 & \omega_5 & \omega_5^4 & \omega_5^2 \\ 1 & \omega_5^4 & \omega_5^3 & \omega_5^3 & \omega_5 \end{bmatrix}$$

mit

$$\omega_5 = \frac{\sqrt{5} - 1 + i\sqrt{2}\sqrt{5+\sqrt{5}}}{4}$$

- $n = 6$

$$DFT_6 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_6 & \omega_6^2 & \omega_6^3 & \omega_6^4 & \omega_6^5 \\ 1 & \omega_6^2 & \omega_6^4 & \omega_6^6 & \omega_6^8 & \omega_6^{10} \\ 1 & \omega_6^3 & \omega_6^6 & \omega_6^9 & \omega_6^{12} & \omega_6^{15} \\ 1 & \omega_6^4 & \omega_6^8 & \omega_6^{12} & \omega_6^{16} & \omega_6^{20} \\ 1 & \omega_6^5 & \omega_6^{10} & \omega_6^{15} & \omega_6^{20} & \omega_6^{25} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_6 & \omega_6^2 & -1 & \omega_6^4 & \omega_6^5 \\ 1 & \omega_6^2 & \omega_6^4 & 1 & \omega_6^2 & \omega_6^4 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \omega_6^4 & \omega_6^2 & 1 & \omega_6^4 & \omega_6^2 \\ 1 & \omega_6^5 & \omega_6^4 & 1 & \omega_6^2 & \omega_6 \end{bmatrix}$$

mit

$$\omega_6 = \frac{1 + i\cdot\sqrt{3}}{2} \qquad \omega_6^2 = \frac{-1 + i\cdot\sqrt{3}}{2}$$

$$\omega_6^4 = \frac{-1 - i\cdot\sqrt{3}}{2} = -\omega_6 \qquad \omega_6^5 = \frac{1 - i\cdot\sqrt{3}}{2} = -\omega_6^2$$

- $n = 8$

$$DFT_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_8 & i & \omega_8^3 & -1 & \omega_8^5 & -i & \omega_8^7 \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & \omega_8^3 & -i & \omega_8 & -1 & \omega_8^7 & i & \omega_8^5 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \omega_8^5 & i & \omega_8^7 & -1 & \omega_8 & -i & \omega_8^3 \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & \omega_8^7 & -i & \omega_8^5 & -1 & \omega_8^3 & i & \omega_8 \end{bmatrix}$$

mit

$$\omega_8 = \frac{1+i}{\sqrt{2}} \qquad \omega_8^3 = \frac{-1+i}{\sqrt{2}} = \omega_8 - \sqrt{2}$$

$$\omega_8^5 = \frac{-1-i}{\sqrt{2}} = -\omega_8 \qquad \omega_8^7 = \frac{1-i}{\sqrt{2}} = \omega_8 - i\cdot\sqrt{2}$$

- Faktorisierung von DFT_4
  - Spalten vertauschen

$$DFT_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \mapsto \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & i & -i \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -i & i \end{bmatrix} = \widetilde{DFT}_4$$

  - Blockstruktur erkennen

$$\left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & i & -i \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -i & i \end{array}\right] = \left[\begin{array}{c|c} DFT_2 & \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} DFT_2 \\ \hline DFT_2 & \begin{pmatrix} -1 & 0 \\ 0 & -i \end{pmatrix} DFT_2 \end{array}\right]$$

- Faktorisierung von DFT_4
  - Faktorisierung erkennen

$$\widetilde{DFT}_4 = \left[\begin{array}{c|c} I_2 & \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\ \hline I_2 & -\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \end{array}\right] \cdot \left[\begin{array}{c|c} DFT_2 & 0_2 \\ \hline 0_2 & DFT_2 \end{array}\right]$$

wobei I_2 = Einheitsmatrix, 0_2 = Nullmatrix

- Faktorisierung von DFT_8
  - Spalten in DFT_8 vertauschen

$$\widetilde{DFT}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i & \omega_8 & \omega_8^3 & \omega_8^5 & \omega_8^7 \\ 1 & -1 & 1 & -1 & i & -i & i & -i \\ 1 & -i & -1 & i & \omega_8^3 & \omega_8 & \omega_8^7 & \omega_8^5 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & i & -1 & -i & \omega_8^5 & \omega_8^7 & \omega_8 & \omega_8^3 \\ 1 & -1 & 1 & -1 & -i & i & -i & i \\ 1 & -i & -1 & i & \omega_8^7 & \omega_8^5 & \omega_8^3 & \omega_8 \end{bmatrix}$$

- Faktorisierung von DFT_8
  - Blockstruktur erkennen

$$\widetilde{DFT}_8 = \left[\begin{array}{c|c} DFT_4 & \begin{bmatrix} 1 & & & \\ & \omega_8 & & \\ & & \omega_8^2 & \\ & & & \omega_8^3 \end{bmatrix} \cdot DFT_4 \\ \hline DFT_4 & \begin{bmatrix} \omega_8^4 & & & \\ & \omega_8^5 & & \\ & & \omega_8^6 & \\ & & & \omega_8^7 \end{bmatrix} \cdot DFT_4 \end{array}\right]$$

  - Faktorisierung erkennen

$$\widetilde{DFT}_8 = \begin{bmatrix} I_4 & D_4 \\ I_4 & -D_4 \end{bmatrix} \cdot \begin{bmatrix} DFT_4 & 0_4 \\ 0_4 & DFT_4 \end{bmatrix}$$

- Faktorisierung von DFT_{2n}

$$\widetilde{DFT}_{2n} = \left[\begin{array}{c|c} DFT_n & \begin{bmatrix} 1 & & & \\ & \omega_{2n} & & \\ & & \ddots & \\ & & & \omega_{2n}^{n-1} \end{bmatrix} \cdot DFT_n \\ \hline DFT_n & \begin{bmatrix} \omega_{2n}^{n} & & & \\ & \omega_{2n}^{n+1} & & \\ & & \ddots & \\ & & & \omega_{2n}^{2n-1} \end{bmatrix} \cdot DFT_n \end{array}\right]$$

$$= \begin{bmatrix} I_n & D_n \\ I_n & -D_n \end{bmatrix} \cdot \begin{bmatrix} DFT_n & 0_n \\ 0_n & DFT_n \end{bmatrix}$$

- Dabei ist

$$D_n = \begin{bmatrix} 1 & & & \\ & \omega_{2n} & & \\ & & \ddots & \\ & & & \omega_{2n}^{n-1} \end{bmatrix}$$

$$-D_n = \omega_{2n}^{n} \cdot D_n = \begin{bmatrix} \omega_{2n}^{n} & & & \\ & \omega_{2n}^{n+1} & & \\ & & \ddots & \\ & & & \omega_{2n}^{2n-1} \end{bmatrix}$$

- Beachte: in der Gleichung

$$\widetilde{DFT}_{2n} = \begin{bmatrix} I_n & D_n \\ I_n & -D_n \end{bmatrix} \cdot \begin{bmatrix} DFT_n & 0_n \\ 0_n & DFT_n \end{bmatrix}$$

  - hat die Matrix $\widetilde{DFT}_{2n}$ $4n^2$ Koeffizienten $\neq 0$
  - haben die Matrizen auf der rechten Seite $4n + 2n^2$ Koeffizienten $\neq 0$
- FFT: diese Zerlegungsidee *rekursiv* durchführen
- Folgerung: Reduktion der Anzahl der Koeffizienten $\neq 0$ von $O(n^2)$ auf $O(n \log n)$

- FFT: der entscheidende "divide-and-conquer"-Trick:

  Rückführung von $DFT_{2n,\omega_{2n}}$ auf DFT_{n,ω_{2n}^2}

  - $\omega = \omega_{2n}$: primitive $2n$-te Einheitswurzel, d.h.

$$\mathcal{R}_{2n} = \{1 = \omega^0, \omega, \omega^2, \ldots, \omega^{2n-1}\}$$

  - $\eta = (\omega_{2n})^2 = \omega_n$: primitive n-te Einheitswurzel, d.h.

$$\mathcal{R}_n = \{1 = \eta^1, \eta, \eta^2, \ldots, \eta^{n-1}\}$$

  - Inklusion: $\mathcal{R}_n \subset \mathcal{R}_{2n}$
  - Genauer: unter der Abbildung $x \mapsto x^2$ wird $\mathcal{R}_{2n}$ 2-zu-1 auf $\mathcal{R}_n$ abgebildet:

$$\omega^k, \omega^{n+k} \mapsto \eta^k \quad (0 \leq k < n)$$

- Polynomialer Ansatz zur FFT
  - Zerlegung von Polynomen nach Parität der Exponenten

$$a(X) = \sum_{i=0}^{2n-1} a_i X^i \quad \leftrightarrow \quad \langle a_0, a_1, \ldots, a_{2n-2}, a_{2n-1} \rangle = \mathbf{a}$$

$$a^{[0]}(X) = \sum_{i=0}^{n-1} a_{2i} X^i \quad \leftrightarrow \quad \langle a_0, a_2, \ldots, a_{2n-4}, a_{2n-2} \rangle = \mathbf{a^{[0]}}$$

$$a^{[1]}(X) = \sum_{i=0}^{n-1} a_{2i+1} X^i \quad \leftrightarrow \quad \langle a_1, a_3, \ldots, a_{2n-3}, a_{2n-1} \rangle = \mathbf{a^{[1]}}$$

  - es gilt

$$a(X) = a^{[0]}(X^2) + X \cdot a^{[1]}(X^2)$$

- und daher für $0 \leq k < 2n$:

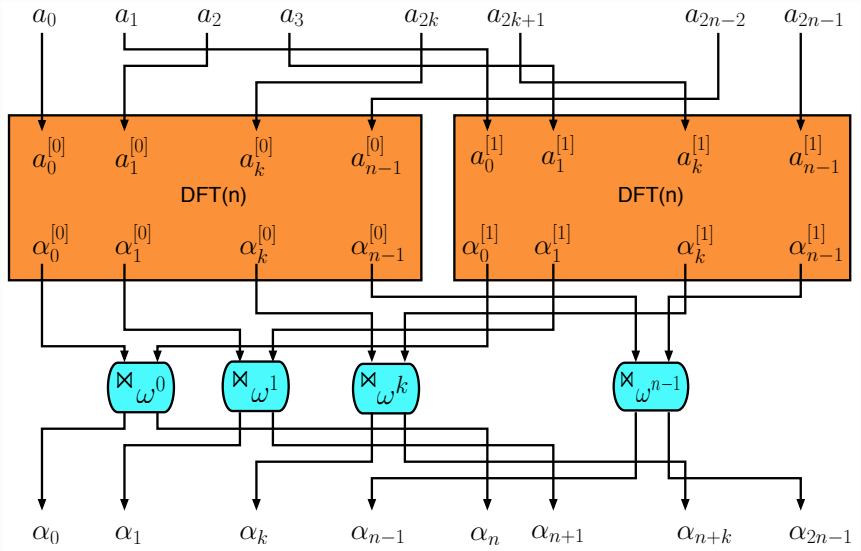
$$\begin{array}{rcccll} a(\omega^k) & = & a^{[0]}(\omega^{2k}) & + & \omega^k \cdot a^{[1]}(\omega^{2k}) & \\ & = & a^{[0]}(\eta^k) & + & \omega^k \cdot a^{[1]}(\eta^k) & (0 \leq k < n) \\ & = & a^{[0]}(\eta^{k-n}) & + & \omega^k \cdot a^{[1]}(\eta^{k-n}) & (n \leq k < 2n) \end{array}$$

- Folgerung:

$$DFT_{2n,\omega}(\mathbf{a}) = DFT_{n,\eta}(\mathbf{a^{[0]}}) \bowtie_{\omega^k} DFT_{n,\eta}(\mathbf{a^{[1]}})$$

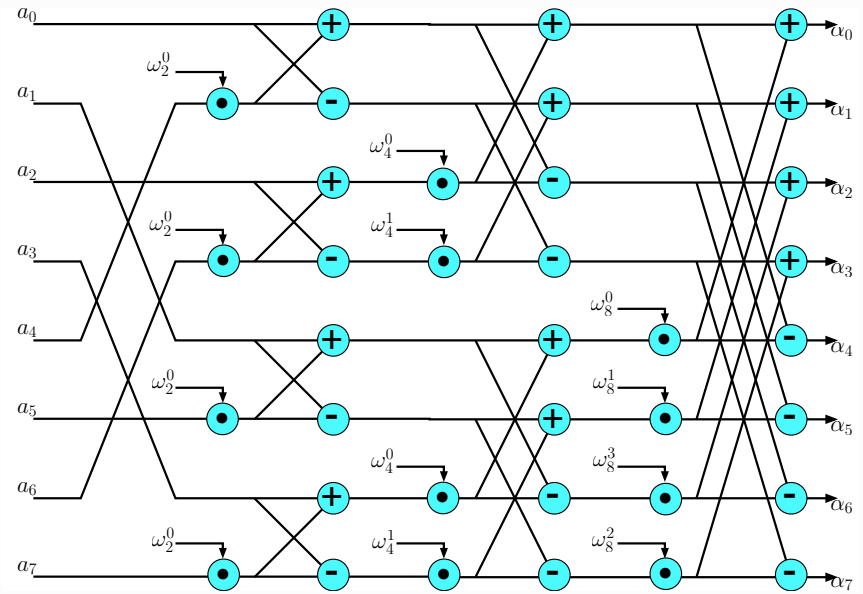
- $\bowtie_{\omega^k}$ ist die "butterfly"-Operation

$$\begin{pmatrix} a(\omega^k) \\ a(\omega^{n+k}) \end{pmatrix} = \begin{pmatrix} 1 & \omega^k \\ 1 & -\omega^k \end{pmatrix} \begin{pmatrix} a^{[0]}(\eta^k) \\ a^{[1]}(\eta^k) \end{pmatrix} \quad (0 \leq k < n)$$




```
FFT (real a[], int n, complex ω)
{
    if (n == 1) return (a_0);
    else
    {
        (α_0^[0], ..., α_{n-1}^[0]) = FFT((a_0, a_2 ..., a_{2n-4}, a_{2n-2}), n/2, ω^2);
        (α_0^[1], ..., α_{n-1}^[1]) = FFT((a_1, a_3 ..., a_{2n-3}, a_{2n-1}), n/2, ω^2);
        for (k = 0; k < n; k++)
            α_k = α_k^[0] + ω^k · α_k^[1];
        for (k = n; k < 2n; k++)
            α_k = α_{k-n}^[0] + ω^k · α_{k-n}^[1];
        return (α_0, α_1, ..., α_{2n-2}, α_{2n-1});
    }
}
```

```
FFT := proc (A,k)
n := 2^k;
if k=0 then RETURN(A) fi;
omega_n := exp(2*Pi*I/n);
omega := 1;
a_0 := [A[0],A[2],...,A[n-2]];
a_1 := [A[1],A[3],...,A[n-1]];
y_0 := FFT(a_0,k-1);
y_1 := FFT(a_1,k-1);
for t from 0 to (n/2)-1 do
   y[t] := y_0[t] + omega*y_1[t];
   y[t+(n/2)] := y_0[t]  - omega*y_1[t];
   omega := omega*omega_n
   od;
RETURN(y);
end;
```

- Komplexität der FFT
  - $F(n)$: Anzahl der komplexen arithmetischen Operationen zur Berechnung von $DFT(n)$ mittels FFT
  - "divide-and-conquer"-Rekursionsgleichung

$$F(2n) = 2 \cdot F(n) + O(n) \ , \quad F(1) = 0$$

  - Lösung:

$$F(n) \in \Theta(n \cdot \log n)$$

- Umkehrabbildung

$$DFT(n)^{-1} = \text{ "Interpolation" an den Stellen } \omega_n^k \in \mathcal{R}_n$$

- Inverses der Vandermonde-Matrix von $DFT(n)$

$$V_n = V_n(\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}) = \left(\omega_n^{ij}\right)_{0 \le i,j < n}$$

ist

$$\frac{1}{n} V_n(\overline{\omega_n}^0, \overline{\omega_n}^1 \ldots, \overline{\omega_n}^{n-1}) = \left(\frac{1}{n} \overline{\omega_n}^{ij}\right)_{0 \le i,j < n} = \left(\frac{1}{n} \omega_n^{-ij}\right)_{0 \le i,j < n}$$

wobei $\overline{\omega_n} = e^{-2\pi i/n} = \omega_n^{-1}$

  - Beweis: für $0 \le i, k < n$ gilt

$$\sum_{j=0}^{n-1} \omega_n^{ij} \cdot \overline{\omega_n}^{jk} = \sum_{j=0}^{n-1} \omega_n^{ij} \cdot \omega_n^{-jk} = \sum_{j=0}^{n-1} \omega_n^{(i-k)j} = \begin{cases} n & i = k \\ 0 & i \neq k \end{cases}$$

- Folgerung:
  - Die Rücktransformation $DFT(n)^{-1}$ ist

    — bis auf banale Änderungen:
    ω_n durch ω_n^{-1} ersetzen und alles durch n dividieren —

    die gleiche Operation wie $DFT(n)$
  - $DFT(n)^{-1}$ kann also ebenfalls mit $\Theta(n \log n)$ komplexen Operationen berechnet werden
  - Man kann die gleichen Programme/Hardware verwenden

- Anwendung: Multiplikation ("Faltung") von zwei Polynomen $a, b \in \mathbb{C}[X]_{<n}$ nach dem Schema der modularen Arithmetik
  - jeweils $2n$ Auswertungen mittels FFT

    $$a(\omega_{2n}^k), b(\omega_{2n}^k) \ (0 \leq k < 2n)$$
  - $2n$ komplexe Multiplikationen

    $$c(\omega_{2n}^k) = a(\omega_{2n}^k) \cdot b(\omega_{2n}^k) \ \ (0 \leq k < 2n)$$
  - Interpolation mittels FFT-Rücktransformation

    $$c(X) = DFT(2n)^{-1}([c(\omega_{2n}^0), \ldots, [c(\omega_{2n}^{2n-1})])$$

erfordert $3 \cdot \Theta(2n \cdot \log 2n) + O(2n) \in \Theta(n \cdot \log n)$ komplexe Operationen, ist also deutlich schneller als das klassische $O(n^2)$-Verfahren

- Ergänzende Bemerkungen
  - FFT kombiniert zwei Prinzipien algebraischer Natur:
    - Domain-Transformation (Koeffizientendarstellung vs. Wertedarstellung) mittels Evaluation $\leftrightarrow$ Interpolation
    - Rekursive Struktur der Einheitswurzeln als Lösungen von $X^n = 1$ und damit als Elemente einer zyklischen Gruppe
  - Zahlreiche Varianten und Variationen über Ringen, die genügend viele Einheitswurzeln enthalten, darunter
    - Ringe $\mathbb{Z}_n$ für geeignete n
    - endliche Körper (bestehen vollständig aus Einheitswurzeln!)
    - andere Zerlegungen als nach Zweierpotenzen
  - Beim Rechnen über $\mathbb{C}$: Rundungsfehler, numerische Stabilität? Andere Ringe/Körper mit exakter Arithmetik oft vorteilhaft
  - Schnellstes bekanntes Multiplikationsverfahren für ganze Zahlen (Schönhage-Strassen, $O(n \log n \log \log n)$) beruht auf FFT über Ringen $\mathbb{Z}_{2^k+1}$

Modulare Polynomarithmetik: Evaluation und Interpolation

Zwei Polynome f(X) und g(X) vom Grad ≤ 3 sollen miteinander multipliziert werden. Das Produkt h(X) = f(X) g(X) ist ein Polynom vom Grad ≤ 7.

```
> f := X -> 5*X^3+3*X^2-4*X+3;
```

$$f := X \to 5\,X^3 + 3\,X^2 - 4\,X + 3$$

```
> g := X -> 2*X^3-5*X^2+7*X-2;
```

$$g := X \to 2\,X^3 - 5\,X^2 + 7\,X - 2$$

```
> h := unapply(expand(f(X)*g(X)),X);
```

$$h := X \to 10\,X^6 - 19\,X^5 + 12\,X^4 + 37\,X^3 - 49\,X^2 + 29\,X - 6$$

Um h(X) nach dem Schema von Evaluation und Interpolation zu berechnen, genügen also 8 Interpolationsstellen, die im Prinzip beliebig gewählt werden könnnen.

Die Interpolationspunkte:

```
> points := [seq(X,X=0..7)];
```

$$points := [0, 1, 2, 3, 4, 5, 6, 7]$$

Die Werte der Polynome f(X) und g(X) an den Interpolationspunkten:

```
> valuesf := map(f,points);
```

$$valuesf := [3, 7, 47, 153, 355, 683, 1167, 1837]$$

```
> valuesg := map(g,points);
```

$$valuesg := [-2, 2, 8, 28, 74, 158, 292, 488]$$

Die Werte des Polynoms h(X) sind die Produkte der entsprechenden Werte von f(X) und g(X)

```
> zip((x,y)-> x*y,valuesf,valuesg);
```

$$[-6, 14, 376, 4284, 26270, 107914, 340764, 896456]$$

Durch Interpolation erhält man das gesuchte Polynom:

```
> interp(points,%,X);
```

$$10\,X^6 - 19\,X^5 + 12\,X^4 + 37\,X^3 - 49\,X^2 + 29\,X - 6$$

Man hätte die Interpolationspunkte auch zufällig wählen können. Beispielsweise mit Hilfe eines Zufallsgenerators. Man muss sich dann nur vergewissern, dass die Punkte paarweise verschieden sind.

```
> die := rand(-99..99);
```

$$die := \mathbf{proc}()\ \mathbf{local}\ t;\ \mathbf{global}\ \_seed;\ \_seed := \mathrm{irem}(a * \_seed,\ p)\,;\ t := \_seed\,;\ \mathbf{to}\ concats\ \mathbf{do}\ \_seed := \mathrm{irem}(a * \_seed,\ p)\,;\ t := s * t + \_seed\ \mathbf{end\ do}\,;\ \mathrm{irem}(t,\ divisor) + offset\ \mathbf{end\ proc}$$

```
> randpoints := [seq(die(),X=0..7)];
```

$$randpoints := [-85, -55, -37, -35, 97, 50, 79, 56]$$

```
> valuesf := map(f,randpoints);
```

$$valuesf := [-3048607, -822577, -249007, -210557, 4591207, 632303, 2483605, 887267]$$

```
> valuesg := map(g,randpoints);
```

$$valuesg := [-1264972, -348262, -108412, -92122, 1778978, 237848, 955424, 335942]$$

```
> zip((x,y)-> x*y,valuesf,valuesg);
```

$$[3856402494004, 286472311174, 26995346884, 19396931954, 8167656246446, 150392003944, 2372895823520, 298070250514]$$

```
> interp(randpoints,%,X);
```

$$10\,X^6 - 19\,X^5 + 12\,X^4 + 37\,X^3 - 49\,X^2 + 29\,X - 6$$

Bei der Diskreten Fouriertransformation der Ordnung n wählt man als Interpolationspunkte die sogenannten n-ten Einheitswurzeln in der komplexen Ebene . Das sind die n Lösungen der Gleichung $X^n = 1$, diese liegen auf dem Einheitskreis; es handelt sich um die n (verschiedenen) Potenzen der sogenannten "primitiven" n-ten Einheitwurzel $e^{2\pi i/n}$.

Im Falle des Beispiels arbeitet man mit $n=8$.

```
> dftpoints := [solve(X^8=1)];
```

$$dftpoints := [-I,\ I,\ -1,\ 1,\ -\frac{1}{2}\sqrt{2} - \frac{1}{2}\,I\sqrt{2},\ \frac{1}{2}\sqrt{2} + \frac{1}{2}\,I\sqrt{2},\ -\frac{1}{2}\sqrt{2} + \frac{1}{2}\,I\sqrt{2},\ \frac{1}{2}\sqrt{2} - \frac{1}{2}\,I\sqrt{2}]$$

Die Liste der 8 Punkte ist nicht in fortlaufenen Potenzen geordnet. Um das explizit zu machen:

```
> omega8 := (1+I)/sqrt(2);
```

$$\omega 8 := (\frac{1}{2} + \frac{1}{2}\,I)\,\sqrt{2}$$

```
> dftpoints := [seq(evalc(omega8^k),k=0..7)];
```

$$dftpoints := [1,\ \frac{1}{2}\sqrt{2} + \frac{1}{2}\,I\sqrt{2},\ I,\ -\frac{1}{2}\sqrt{2} + \frac{1}{2}\,I\sqrt{2},\ -1,\ -\frac{1}{2}\sqrt{2} - \frac{1}{2}\,I\sqrt{2},\ -I,\ \frac{1}{2}\sqrt{2} - \frac{1}{2}\,I\sqrt{2}]$$

Die Polynome f(X) und g(X) und des Produktes h(X) haben an diesen Stellen komplexe Werte.

```
> valuesf := map(evalc,map(f,dftpoints));
```

$$valuesf := [7,\ -\frac{9}{2}\sqrt{2}+3+I\,(\frac{1}{2}\sqrt{2}+3),\ -9\,I,\ \frac{9}{2}\sqrt{2}+3+I\,(\frac{1}{2}\sqrt{2}-3),\ 5,$$
$$\frac{9}{2}\sqrt{2}+3+I\,(-\frac{1}{2}\sqrt{2}+3),\ 9\,I,\ -\frac{9}{2}\sqrt{2}+3+I\,(-\frac{1}{2}\sqrt{2}-3)]$$

```
> valuesg := map(evalc,map(g,dftpoints));
```

$$valuesg := [2,\ \frac{5}{2}\sqrt{2}-2+I\,(\frac{9}{2}\sqrt{2}-5),\ 3+5\,I,\ -\frac{5}{2}\sqrt{2}-2+I\,(\frac{9}{2}\sqrt{2}+5),\ -16,$$
$$-\frac{5}{2}\sqrt{2}-2+I\,(-\frac{9}{2}\sqrt{2}-5),\ 3-5\,I,\ \frac{5}{2}\sqrt{2}-2+I\,(-\frac{9}{2}\sqrt{2}+5)]$$

```
> valuesh := map(evalc,map(h,dftpoints));
```

$$valuesh := [14,\ \frac{11}{2}\sqrt{2}-18+I\,(-59+\frac{85}{2}\sqrt{2}),\ 45-27\,I,\ -\frac{11}{2}\sqrt{2}-18+I\,(59+\frac{85}{2}\sqrt{2}),$$
$$-80,\ -\frac{11}{2}\sqrt{2}-18+I\,(-59-\frac{85}{2}\sqrt{2}),\ 45+27\,I,\ \frac{11}{2}\sqrt{2}-18+I\,(59-\frac{85}{2}\sqrt{2})]$$

Die Werte von h(X) erhält man auch durch Bildung der Produkte der entsprechenden Werte von f(X) und g(X)

```
> zip((x,y)-> evalc(x*y),valuesf,valuesg);
```

$$[14,\ \frac{11}{2}\sqrt{2}-18+I\,(-59+\frac{85}{2}\sqrt{2}),\ 45-27\,I,\ -\frac{11}{2}\sqrt{2}-18+I\,(59+\frac{85}{2}\sqrt{2}),\ -80,$$
$$-\frac{11}{2}\sqrt{2}-18+I\,(-59-\frac{85}{2}\sqrt{2}),\ 45+27\,I,\ \frac{11}{2}\sqrt{2}-18+I\,(59-\frac{85}{2}\sqrt{2})]$$

Die Berechnung (der Koeffizienten) von h(X) kann man in diesem Fall statt durch Interpolation viel einfacher durch Anwendung der inversen Fouriertransformation machen. Dazu interpretiert man die Folge der Funktionswerte als Koeffizienten eines Polynoms H(X). (Jetzt kommt es auf die richtige Reihenfolge an!)

```
> convert(zip((x,y)->x*y,valuesh,[seq(X^k,k=0..7)]),`+`);
```

$$14+(\frac{11}{2}\sqrt{2}-18+I\,(-59+\frac{85}{2}\sqrt{2}))\,X+(45-27\,I)\,X^2$$
$$+(-\frac{11}{2}\sqrt{2}-18+I\,(59+\frac{85}{2}\sqrt{2}))\,X^3-80\,X^4$$
$$+(-\frac{11}{2}\sqrt{2}-18+I\,(-59-\frac{85}{2}\sqrt{2}))\,X^5+(45+27\,I)\,X^6$$
$$+(\frac{11}{2}\sqrt{2}-18+I\,(59-\frac{85}{2}\sqrt{2}))\,X^7$$

```
> H := unapply(%,X);
```

$$H := X \to 14+(\frac{11}{2}\sqrt{2}-18+I\,(-59+\frac{85}{2}\sqrt{2}))\,X+(45-27\,I)\,X^2$$
$$+(-\frac{11}{2}\sqrt{2}-18+I\,(59+\frac{85}{2}\sqrt{2}))\,X^3-80\,X^4$$
$$+(-\frac{11}{2}\sqrt{2}-18+I\,(-59-\frac{85}{2}\sqrt{2}))\,X^5+(45+27\,I)\,X^6$$
$$+(\frac{11}{2}\sqrt{2}-18+I\,(59-\frac{85}{2}\sqrt{2}))\,X^7$$

Die Auswertung geschieht wiederum an den achten Einheitswurzeln, nun aber im umgekehrten Durchlaufungssinn:

```
> inversedftpoints :=[seq(evalc(omega8^(-k)),k=0..7)];
```

$$inversedftpoints :=$$
$$[1,\ \frac{1}{2}\sqrt{2}-\frac{1}{2}\,I\,\sqrt{2},\ -I,\ -\frac{1}{2}\sqrt{2}-\frac{1}{2}\,I\,\sqrt{2},\ -1,\ -\frac{1}{2}\sqrt{2}+\frac{1}{2}\,I\,\sqrt{2},\ I,\ \frac{1}{2}\sqrt{2}+\frac{1}{2}\,I\,\sqrt{2}]$$

```
> valuesH := map(expand,map(evalc,map(H,inversedftpoints)));
```

$$valuesH := [-48,\ 232,\ -392,\ 296,\ 96,\ -152,\ 80,\ 0]$$

Nach Divisions durch den Skalierungsfaktor 8 erhält man in der Tat die Koeffizienten des Polynoms h(X).

```
> map(x->1/8*x,%);
```

$$[-6,\ 29,\ -49,\ 37,\ 12,\ -19,\ 10,\ 0]$$

Im Beispielfall n=8 kann man exakt rechnen, da $\omega 8$ mittels Wurzeln exakt ausgedrückt werden kann. Das ist im allgemeinen nicht mehr der Fall und man muss die Rechnungen (sofern man nicht in endliche Körper und Ringe ausweicht) mit Fliesskamma-Arithmetik, also mit Rundungsfehlern behaftet, durchfhren. Zur Illustration folgt das gleiche Beispiel mit der in Maple eingebauten FFT-Funktion.

Komplexe FFT in Maple

Liste der Koeffizienten des Polynoms f(X):

```
> fcoeffs := array([seq(coeff(f(X),X,k),k=0..7)]);
```

$$fcoeffs := [3, -4, 3, 5, 0, 0, 0, 0]$$

Die Koeffizienten müssen in Real- und Imaginärteile zerlegt werden:

```
> reell := fcoeffs;
```

$$reell := fcoeffs$$

```
> imag := array([0$8]);
```

$$imag := [0, 0, 0, 0, 0, 0, 0, 0]$$

Durchführung der DFT der Ordnung $8 = 2^3$ mittels FFT:

```
> FFT(3,reell,imag);
```

$$8$$

Man erhält im Frequenzbereich die Liste der Werte auch nach Real- und Imaginärteil getrennt:

```
> Freell := copy(reell):
> print(reell);
```

[7.000000001, −3.363961017, 0., 9.363961017, 4.999999999, 9.363961017, 0., −3.363961017]

```
> Fimag := copy(imag):
> print(imag);
```

[0, −3.707106773, 8.999999995, 2.292893219, 0., −2.292893219, −8.999999995, 3.707106773]

Mittels inverser Fouriertransformation sollten sich die Koeffizienten von f(X) wiederherstellen lassen. Das gelingt nur bis auf Rundungsfehler!

```
> iFFT(3,reell,imag);
```

$$8$$

```
> op(reell);
```

[3.000000000, −3.999999992, 2.999999998, 4.999999989, 0., $-.2500000000\,10^{-8}$, $.2500000000\,10^{-8}$, $.6250000000\,10^{-8}$]

```
> op(imag);
```

[0., $-.7500000000\,10^{-9}$, 0., $-.5000000000\,10^{-9}$, 0., $.7500000000\,10^{-9}$, 0., $.5000000000\,10^{-9}$]

FFT der Koeffizientenliste von g(X):

```
> gcoeffs := array([seq(coeff(g(X),X,k),k=0..7)]);
```

$$gcoeffs := [-2, 7, -5, 2, 0, 0, 0, 0]$$

```
> reell := gcoeffs;
```

$$reell := gcoeffs$$

```
> imag := array([0$8]);
```

$$imag := [0, 0, 0, 0, 0, 0, 0, 0]$$

```
> FFT(3,reell,imag);
```

$$8$$

```
> Greell := reell:
> print(reell);
```

[2, 1.535533906, 3., −5.535533906, −16, −5.535533906, 3., 1.535533906]

```
> Gimag := imag:
> print(imag);
```

[0, −1.363961031, −5., −11.36396102, 0, 11.36396102, 5., 1.363961031]

Punktweise Multiplikation im Frequenzbereich:

```
> Fcoeffs := zip((x,y) -> x+I*y,Freell,Fimag);
```

$$Fcoeffs := [7.000000001, -3.363961017 - 3.707106773\,I, 0. + 8.999999995\,I, 9.363961017 + 2.292893219\,I, 4.999999999 + 0.\,I, 9.363961017 - 2.292893219\,I, 0. - 8.999999995\,I, -3.363961017 + 3.707106773\,I]$$

```
> Gcoeffs := zip((x,y) -> x+I*y,Greell,Gimag);
```

$$Gcoeffs := [2, 1.535533906 - 1.363961031\,I, 3. - 5.\,I, -5.535533906 - 11.36396102\,I, -16, -5.535533906 + 11.36396102\,I, 3. + 5.\,I, 1.535533906 + 1.363961031\,I]$$

```
> Hcoeffs := zip((x,y)-> evalc(x*y),Fcoeffs,Gcoeffs);
```

$$Hcoeffs := [14.00000000, -10.22182538 - 1.104076406\,I, 44.99999998 + 26.99999998\,I, -25.77817454 - 119.1040761\,I, -79.99999998 - 0.\,I, -25.77817454 + 119.1040761\,I, 44.99999998 - 26.99999998\,I, -10.22182538 + 1.104076406\,I]$$

Vergleich mit der Fouriertransformierten (in floats) H(X) der Produktpolynoms h(X):

```
> evalf(H(X));
```

$$14. - (10.22182541 - 1.10407638\,I)\,X + (45. - 27.\,I)\,X^2 - (25.77817459 - 119.1040764\,I)\,X^3 - 80.\,X^4 - (25.77817459 + 119.1040764\,I)\,X^5 + (45. + 27.\,I)\,X^6 - (10.22182541 + 1.10407638\,I)\,X^7$$

```
> Hreal := map(x->Re(x),Hcoeffs);
```

$$Hreal := [14.00000000, -10.22182538, 44.99999998, -25.77817454, -79.99999998, -25.77817454, 44.99999998, -10.22182538]$$

```
> Himag := map(x->Im(x),Hcoeffs);
```

$Himag :=$ [0., −1.104076406, 26.99999998, −119.1040761, −0., 119.1040761, −26.99999998, 1.104076406]

Rücktransformation mittels der Werte von H(X):

```
> iFFT(3,Hreal,Himag);
```

$$8$$

```
> op(Hreal);
```

[−5.999999982, 28.99999992, −48.99999991, 36.99999992, 11.99999998, −18.99999992, 9.999999938, $.5000000000\,10^{-7}$]

```
> op(Himag);
```

[0., $-.3750000000\,10^{-8}$, 0., $.6250000000\,10^{-8}$, 0., $.3750000000\,10^{-8}$, 0., $-.6250000000\,10^{-8}$]

Vergleich mit dem Produktpolynom h(X)

```
> h(X);
```

$$10\,X^6 - 19\,X^5 + 12\,X^4 + 37\,X^3 - 49\,X^2 + 29\,X - 6$$

Beispiel zur Schnellen Fourier-Transformation

```
> x := Array([-1,-1,-1,-1,1,1,1,1]);  # real parts of data
```
x := [-1, -1, -1, -1, 1, 1, 1, 1]

```
> y := Array([0,0,0,0,0,0,0,0]); # imaginary parts of data
```
y := [0, 0, 0, 0, 0, 0, 0, 0]

```
> FFT(3,x,y): # transform data
> x;
```
[0, -2.000000001, 0., -1.999999999, 0, -1.999999999, 0., -2.000000001]

```
> y; # imaginary parts of transformed data
```
[0, 4.828427122, 0., 0.828427124, 0, -.828427124, 0., -4.828427122]

```
> zip((a,b)->a+b*I, x, y):  convert(%, list);
```
[0, -2.000000001 + 4.828427122 I, 0. + 0. I, -1.999999999 + 0.828427124 I, 0, -1.999999999 - 0.828427124 I, 0. + 0. I, -2.000000001 - 4.828427122 I]

```
> iFFT(3,x,y): # check results
> x;
```
[-1.000000000, -.9999999990, -.9999999995, -.9999999985, 1.000000000, 0.9999999990, 0.9999999995, 0.9999999985]

```
> y;
```
[0., $2.500000000\ 10^{-10}$, 0., $-2.500000000\ 10^{-10}$, 0., $-2.500000000\ 10^{-10}$, 0., $2.500000000\ 10^{-10}$]

```
> y := map(fnormal, y);
```
y := [0., 0., 0., -0., 0., -0., 0., 0.]

Glättung einer verrauschten harmonischen Schwingung durch Faltung mit einer Gaussverteilung

Definition einer Störfunktion (Rauschen)

```
> noise := stats[random, normald]:
```

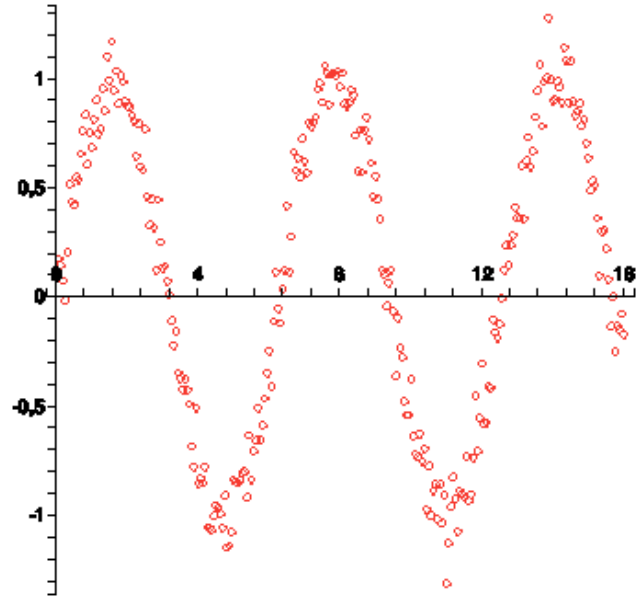
Harmonische Schwingung mit überlagertem Rauschen

```
> re_data := Array([seq(sin(0.0625*k)+0.1*noise(),
    k=1..2^8)]);
```
re_data := 256 Array, Data Type: anything, Storage: rectangular, Order: Fortran_order

```
> im_data := Array([seq(0, k=1..2^8)]):
> xcoords := Array([seq(0.0625*k, k=1..2^8)]):
```

Grafische Darstellung

```
> plotdata := convert(zip((a,b)->[a,b], xcoords,
    re_data), list):
> plot(plotdata, style=POINT);
```



Definition einer Gaussfunktion zur Faltung (Kern,kernel)

```
> re_kernel := Array([seq(exp(-100.0*(k/2^8)^2),
    k=1..2^8)]):
> im_kernel := Array([seq(0, k=1..2^8)]):
```

FFT in den Frequezbereich von verrauschten Daten und Kern

```
> FFT(8, re_data, im_data):
> FFT(8, re_kernel, im_kernel):
```

Punktweise Multiplikation der transformierten Daten

```
> data := zip((a,b)->(a+b*I), re_data, im_data):
> kernel := zip((a,b)->(a+b*I), re_kernel, im_kernel):
> newdata := zip((a,b)->a*b, data, kernel):
> new_re_data := map(Re, newdata):
> new_im_data := map(Im, newdata):
```

Rücktransformation mittels inverser FFT

```
> iFFT(8, new_re_data, new_im_data):
```

Grafische Drastellung der geglätteten Daten

```
> plotdata := convert(zip((a,b)->[a,b], xcoords,
    new_re_data), list):
> plot(plotdata, style=POINT);
```