fhwedel
UNIVERSITY OF APPLIED SCIENCES

Informatik-Seminar WS07/08

Computeralgebra

bei Prof. Dr. Sebastian Iwanowski

Langzahlarithmetik: Vereinfachen von Brüchen und Faktorisierung

28.11.2007

Referent: Thomas Stuht (winf2967)

Agenda

1. Motivation
2. Konzepte
3. Rationale Arithmetik
4. Faktorisierung
5. Anwendung
6. Fazit

Kapitel 1

Motivation

- Vereinfachen von Brüchen
  - $\frac{2318068701}{2436944019}$ ohne Hilfsmittel nicht zu kürzen
  - GGT-Berechnung
    - Euklidischer Algorithmus
    - **Erweiterter Euklidischer Algorithmus**
  - Konzepte als Hilfssätze in Algorithmen und Beweisen

- Faktorisierung
  - Grundlegend für die Computeralgebra
  - Einfache Verfahren sind intuitiv und leicht anzuwenden, aber nur für Zahlen bis $\leq 10^6$ effizient
  - In der Praxis viel größere Zahlen, die **effiziente Verfahren** erfordern
  - Wichtig für die Kryptografie

Schwerpunkte

- Wie wird der erweiterte Euklidische Algorithmus verwendet?
- Wie können Brüche vereinfacht werden?
- Wie lassen sich ganze Zahlen zerlegen?

Kapitel 2

Konzepte

Euklidischer Algorithmus

Erweiterter Euklidischer Algorithmus

GGT für n ganze Zahlen

Euklidischer Algorithmus: Einführung

- Wichtiger Algorithmus in der Zahlentheorie
- Berechnet GGT zweier ganzer Zahlen **schneller**, als durch Primfaktorzerlegung mittels Faktorisierung
- Laufzeit: $O(n \cdot m)$ für zwei ganze Zahlen der Länge n und m
- Wiederholte Anwendung einer **Division mit Rest**
- Für $x, y \in \mathbb{Z}$ gilt $x = q \cdot y + r \quad mit \quad 0 \leq r < y$ für ein Paar $(Quotient\ q, Rest\ r)$
- **Grundlegende Beziehung** $GGT(x, y) = GGT(y, r)$

Euklidischer Algorithmus: Pseudocode

- Listing

```
1  public GGT( x∈ℤ , y∈ℤ){
2    if (y = 0) return GGT=x;
3    do {
4          q = x DIV y;
5          r = x MOD y;
6          x = y; y = r;
7    } while (r != 0);
8    return GGT = y;
9  }
```

- Zahlen sind *teilerfremd*, wenn $GGT = 1$

Euklidischer Algorithmus: Beispiel

- Beispiel
  - Gesucht: GGT von $x=3564$ und $y=2727$

$$
\begin{array}{rcrcrcr}
x & = & q & \cdot & y & + & r \\
3564 & = & 1 & \cdot & 2727 & + & 837 \\
2727 & = & 3 & \cdot & 837 & + & 216 \\
837 & = & 3 & \cdot & 216 & + & 189 \\
216 & = & 1 & \cdot & 189 & + & \mathbf{27} \\
189 & = & 7 & \cdot & \mathbf{27} & + & \underline{0}
\end{array}
$$

  - $GGT = \mathbf{27}$
  - 5 Iterationen notwendig = *Euklidische Länge*

Erweiterter Euklidischer Algorithmus: Einführung

- **Anwendung** in vielen Algorithmen der Zahlentheorie
  - z. B. Berechnung von **inversen Elementen** innerhalb von Restklassenringen *(Chinesischer Restsatz/ Koepf s. Kapitel 4)*
  - Auch zur Schlüsselerzeugung in der **Kryptografie**
- Berechnet ebenfalls den **GGT**, stellt ihn aber zusätzlich als **Linearkombination** der Parameter dar:
  $GGT(x, y) = x \cdot s + y \cdot t \ \ mit \ \ x, y, s, t \in \mathbb{Z}$
- s und t heißen *Bézoutkoeffizienten*
- Ergebnis des einfachen Euklidischen Algorithmus kann auch zur Darstellung des GGT als Linearkombination genutzt werden (*s. Koepf*)
  - **Nachträgliches** Umformen der Gleichungen: $x = q \cdot y + r \Leftrightarrow x - q \cdot y = r$
  - Ausgehend von letzter Zeile aufwärts rekursiv einsetzen

Erweiterter Euklidischer Algorithmus: Beispiel

- Beispiel 1 (mit einfachem Euklidischen Algorithmus)
  - Gesucht: GGT und Linearkombination von $x=3564$ und $y=2727$
  - Nachträgliches Umformen der Gleichungen: $x=q\cdot y+r \Leftrightarrow x-q\cdot y=r$

$$
\begin{array}{rcrcrcrcrcrcr}
3564 & = & 1 & \cdot & 2727 & + & 837 & \Leftrightarrow & 3564 & - & 1 & \cdot & 2727 & = & 837 \\
2727 & = & 3 & \cdot & 837 & + & 216 & \Leftrightarrow & 2727 & - & 3 & \cdot & 837 & = & 216 \\
837 & = & 3 & \cdot & 216 & + & 189 & \Leftrightarrow & 837 & - & 3 & \cdot & 216 & = & 189 \\
216 & = & 1 & \cdot & 189 & + & \mathbf{27} & \Leftrightarrow & 216 & - & 1 & \cdot & 189 & = & 27 \\
189 & = & 7 & \cdot & \mathbf{27} & + & \underline{0} & & & & & & & &
\end{array}
$$

  - Ausgehend von letzter Zeile aufwärts rekursiv einsetzen

$$
\begin{array}{rcl}
27 & = & 216-1\cdot\underline{189} \\
27 & = & 216-1\cdot(837-3\cdot 216) = 216-837+3\cdot 216 = 4\cdot\underline{216}-837 \\
27 & = & 4\cdot(2727-3\cdot 837)-837 = 4\cdot 2727-13\cdot\underline{837} \\
\mathbf{27} & = & 4\cdot 2727-13\cdot(3564-1\cdot 2727) = \mathbf{3564\cdot(-13)+2727\cdot 17}
\end{array}
$$

  - Linearkombination: $3564\cdot(-13)+2727\cdot 17=27$

Erweiterter Euklidischer Algorithmus: Vergleich

- **Einschränkungen** durch Einsatz des einfachen Euklidischen Algorithmus:
  - **Zusätzlicher** Rechenaufwand notwendig
  - Erst *nach* Berechnung des GGT durchführbar (2. Schritt)
  - **Nur GGT als Linearkombination** der Parameter darstellbar
- **Verbesserung** durch Einsatz der „Erweiterung“:
  - Linearkombination **parallel** zur GGT-Berechnung
  - Linearkombination aller Zwischen-Restwerte und des GGT

Erweiterter Euklidischer Algorithmus: Pseudocode

- Listing

```
1   public GGT( x∈ℤ , y∈ℤ ) {
2       r_0 = x; s_0 = 1; t_0 = 0; // init
3       r_1 = y; s_1 = 0; t_1 = 1;
4       i = 1;
5       while (r_i != 0) {
6           q_i   = r_i-1 DIV r_i;
7           r_i+1 = r_i-1 - q_i * r_i;
8           s_i+1 = s_i-1 - q_i * s_i;
9           t_i+1 = t_i-1 - q_i * t_i;
10          i++;
11      }
12      return GGT = r_i-1;
13  }
```

Erweiterter Euklidischer Algorithmus: Beispiel

- Beispiel 2 (mit erweitertem Euklidischen Algorithmus)
  - Gesucht: GGT und Linearkombination von $x=3564$ und $y=2727$

i	q_i	r_i	s_i	t_i	$x \cdot s_i + y \cdot t_i = r_i$
0		3564	1	0	$3564 \cdot 1 + 2727 \cdot 0 = 3564$
1	1	2727	0	1	$3564 \cdot 0 + 2727 \cdot 1 = 2727$
2	3	837	1	-1	$3564 \cdot 1 + 2727 \cdot (-1) = 837$
3	3	216	-3	4	$3564 \cdot (-3) + 2727 \cdot 4 = 216$
4	1	189	10	-13	$3564 \cdot 10 + 2727 \cdot (-13) = 189$
5	7	**27**	-13	17	$3564 \cdot (-13) + 2727 \cdot 17 = 27$
6		0	101	-132	$3564 \cdot 101 + 2727 \cdot (-132) = 0$

  - $GGT = \mathbf{27}$
  - Linearkombination: $3564 \cdot (-13) + 2727 \cdot 17 = 27$

Erweiterter Euklidischer Algorithmus: Bewertung

- **Vorteile** des erweiterten Euklidischen Algorithmus
  - Linearkombination als „Seiteneffekt“
  - Für **jede Zeile** gilt $x \cdot s_i + y \cdot t_i = r_i$
  - Linearkombinationen der Zwischen-Restwerte werden in anderen Algorithmen genutzt
  - Vergleichsweise geringer Aufwand gegenüber einfachem Euklidischen Algorithmus
- Laufzeit: $O(n \cdot m)$ für zwei ganze Zahlen der Länge n und m
  - Schlimmster Fall wären zwei aufeinander folgende Fibonaccizahlen ($F_n = F_{n-1} + F_{n-2}$ mit $F_0 = 0$, $F_1 = 1$ und $n \geq 2$)
  - Alle Quotienten sind 1, als Reste ergeben sich der Reihe nach alle kleineren Fibonaccizahlen
  - $GGT = 1$

GGT für n ganze Zahlen: Pseudocode

- **Verallgemeinerung** des GGT von zwei Zahlen
- $GGT(u_1, u_2, \ldots, u_n) = GGT(u_1, GGT(u_2, \ldots, u_n))$
- Es gilt: $GGT(u_1, u_2, \ldots, u_n, 1) = 1$
- Laufzeit: $O(n \cdot m^2)$ für n ganze Zahlen, die alle Länge m besitzen
- Listing

```
1  public NZahlenGGT(u_1 .. u_n ∈ℤ){
2      d = u_n; k = n-1; // init
3      while (d != 1 && k > 0) {
4          d = GGT(u_k,d);
5          k--;
6      }
7      return NZahlenGGT = d;
8  }
```

GGT für n ganze Zahlen: Beispiel

- Beispiel 1
  - Gesucht: GGT($u_1=14,\ u_2=147,\ u_3=42\ ,u_4=294,\ u_5=693$)

d	k
$u_5 = 693$	4
$GGT(u_4,d) = GGT(294,693) = 21$	3
$GGT(u_3,d) = GGT(42,21) = 21$	2
$GGT(u_2,d) = GGT(147,21) = 21$	1
$GGT(u_1,d) = GGT(14,21) = 7$	0

  - $k=0 \quad \Rightarrow \quad GGT=7$

GGT für n ganze Zahlen: Beispiel

- Beispiel 2
  - Gesucht: GGT($u_1=14,\ u_2=147,\ u_3=\underline{113},\ u_4=294,\ u_5=693$)

d	k
$u_5 = 693$	4
$GGT(u_4,d) = GGT(294,693) = 21$	3
$GGT(u_3,d) = GGT(113,21) = 1$	2

  - $d=1 \quad \Rightarrow \quad GGT=\mathbf{1}$ (teilerfremd)

Kapitel 3

Rationale Arithmetik

Vereinfachen von Brüchen

Rechenoperationen

Vereinfachen von Brüchen: Einführung

- Verwendung **bekannter Verfahren** zur Bestimmung des GGT von Zähler und Nenner
- **Kürzen** während einer Berechnung führt u. U. zu einer **besseren Lesbarkeit** und **effizienteren Weiterverarbeitung**
- **Computeralgebrasysteme** liefern meist automatisch die gekürzten Brüche als Ergebnis

Maxima (Open-Source/ GNU General Public License (GPL))
http://maxima.sourceforge.net

Vereinfachen von Brüchen: Beispiel

- Beispiel (Kürzen von $\frac{2318068701}{2436944019}$)
  - Verwendung des erweiterten Euklidischen Algorithmus
    - Gesucht: GGT von $x = 2318068701$ und $y = 2436944019$

i	q_i	r_i	s_i	t_i	$x \cdot s_i + y \cdot t_i = r_i$
0		2318068701	1	0	$2318068701 \cdot 1 + 2436944019 \cdot 0 = 2318068701$
1	0	2436944019	0	1	$2318068701 \cdot 0 + 2436944019 \cdot 1 = 2436944019$
2	1	2318068701	1	0	$2318068701 \cdot 1 + 2436944019 \cdot 0 = 2318068701$
3	19	118875318	-1	1	$2318068701 \cdot (-1) + 2436944019 \cdot 1 = 118875318$
4	2	**59437659**	20	-19	$2318068701 \cdot 20 + 2436944019 \cdot (-19) = 59437659$
5		0	-41	39	$2318068701 \cdot (-41) + 2436944019 \cdot 39 = 0$

- $GGT = \mathbf{59437659}$
- Teilen von Zähler und Nenner durch den GGT liefert $\frac{39}{41}$

Rechenoperationen: Vereinfachen der Addition

- Sei angenommen $\frac{a}{b}$ und $\frac{c}{d}$ sind bereits gekürzt und die Nenner seien verschieden
  - **Addition**: $\frac{a}{b}+\frac{c}{d}=\frac{a}{b}\cdot\frac{d}{d}+\frac{c}{d}\cdot\frac{b}{b}=\frac{a\cdot d+b\cdot c}{b\cdot d}$
  - Durch Multiplikation treten „große“ Zahlen auf ⇒ **Kürzen** sinnvoll
  - Teilen von **Zähler** und **Nenner** durch $GGT(a\cdot d+b\cdot c, b\cdot d)$
  - Computer berechnet: eine Addition, 3 Multiplikationen, ein GGT und 2 Divisionen zum Kürzen
- Beispiel (Addition von $\frac{a}{b}=\frac{16}{21}$ und $\frac{c}{d}=\frac{5}{9}$)
  - Berechnung von $GGT(a\cdot d+b\cdot c, b\cdot d)=GGT(16\cdot 9+21\cdot 5, 21\cdot 9)=GGT(249, 189)=3$
  - Teilen des Zählers und Nenners von $\frac{249}{189}$ durch den GGT
  - Ergebnis: $\frac{83}{63}$

Rechenoperationen: Vereinfachen der Subtraktion

- Sei angenommen $\frac{a}{b}$ und $\frac{c}{d}$ sind bereits gekürzt und die Nenner seien verschieden
  - **Subtraktion**: $\frac{a}{b}-\frac{c}{d}=\frac{a}{b}\cdot\frac{d}{d}-\frac{c}{d}\cdot\frac{b}{b}=\frac{a\cdot d-b\cdot c}{b\cdot d}$
  - Durch Multiplikation treten „große“ Zahlen auf ⇒ **Kürzen** sinnvoll
  - Teilen von **Zähler** und **Nenner** durch $GGT(a\cdot d-b\cdot c, b\cdot d)$
  - Computer berechnet: eine Subtraktion, 3 Multiplikationen, ein GGT und 2 Divisionen zum Kürzen
- Beispiel (Subtraktion von $\frac{a}{b}=\frac{16}{21}$ und $\frac{c}{d}=\frac{5}{9}$)
  - Berechnung von $GGT(a\cdot d-b\cdot c, b\cdot d)=GGT(16\cdot 9-21\cdot 5, 21\cdot 9)=GGT(39, 189)=3$
  - Teilen des Zählers und Nenners von $\frac{39}{189}$ durch den GGT
  - Ergebnis: $\frac{13}{63}$

Rechenoperationen: Vereinfachen der Multiplikation

- Sei angenommen $\frac{a}{b}$ und $\frac{c}{d}$ sind bereits gekürzt und die Nenner seien verschieden
  - **Multiplikation**: $\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$
  - Durch Multiplikation treten „große“ Zahlen auf ⇒ **Kürzen** sinnvoll
  - Teilen von **Zähler** und **Nenner** durch $GGT(a \cdot c, b \cdot d)$
  - Computer berechnet: 2 Multiplikationen, ein GGT und 2 Divisionen zum Kürzen
- Beispiel (Multiplikation von $\frac{a}{b} = \frac{16}{21}$ und $\frac{c}{d} = \frac{5}{9}$)
  - Berechnung von $GGT(a \cdot c, b \cdot d) = GGT(16 \cdot 5, 21 \cdot 9) = GGT(80, 189) = 1$
  - Zähler und Nenner sind *teilerfremd*
  - Ergebnis: $\frac{80}{189}$ lässt sich nicht weiter kürzen

Rechenoperationen: Vereinfachen der Division

- Sei angenommen $\frac{a}{b}$ und $\frac{c}{d}$ sind bereits gekürzt und die Nenner seien verschieden
  - **Division**: $\frac{a}{b} : \frac{c}{d} = \frac{a}{b} \cdot \frac{d}{c} = \frac{a \cdot d}{b \cdot c}$
  - Durch Multiplikation treten „große“ Zahlen auf ⇒ **Kürzen** sinnvoll
  - Teilen von **Zähler** und **Nenner** durch $GGT(a \cdot d, b \cdot c)$
  - Computer berechnet: 2 Multiplikationen, ein GGT und 2 Divisionen zum Kürzen
- Beispiel (Division von $\frac{a}{b} = \frac{16}{21}$ und $\frac{c}{d} = \frac{5}{9}$)
  - Berechnung von $GGT(a \cdot d, b \cdot c) = GGT(16 \cdot 9, 21 \cdot 5) = GGT(144, 105) = 3$ *(Bildung des Kehrwertes somit berücksichtigt)*
  - Teilen des Zählers und Nenners von $\frac{144}{105}$ durch den GGT
  - Ergebnis: $\frac{48}{35}$

Kapitel 4

Faktorisierung

Einführung

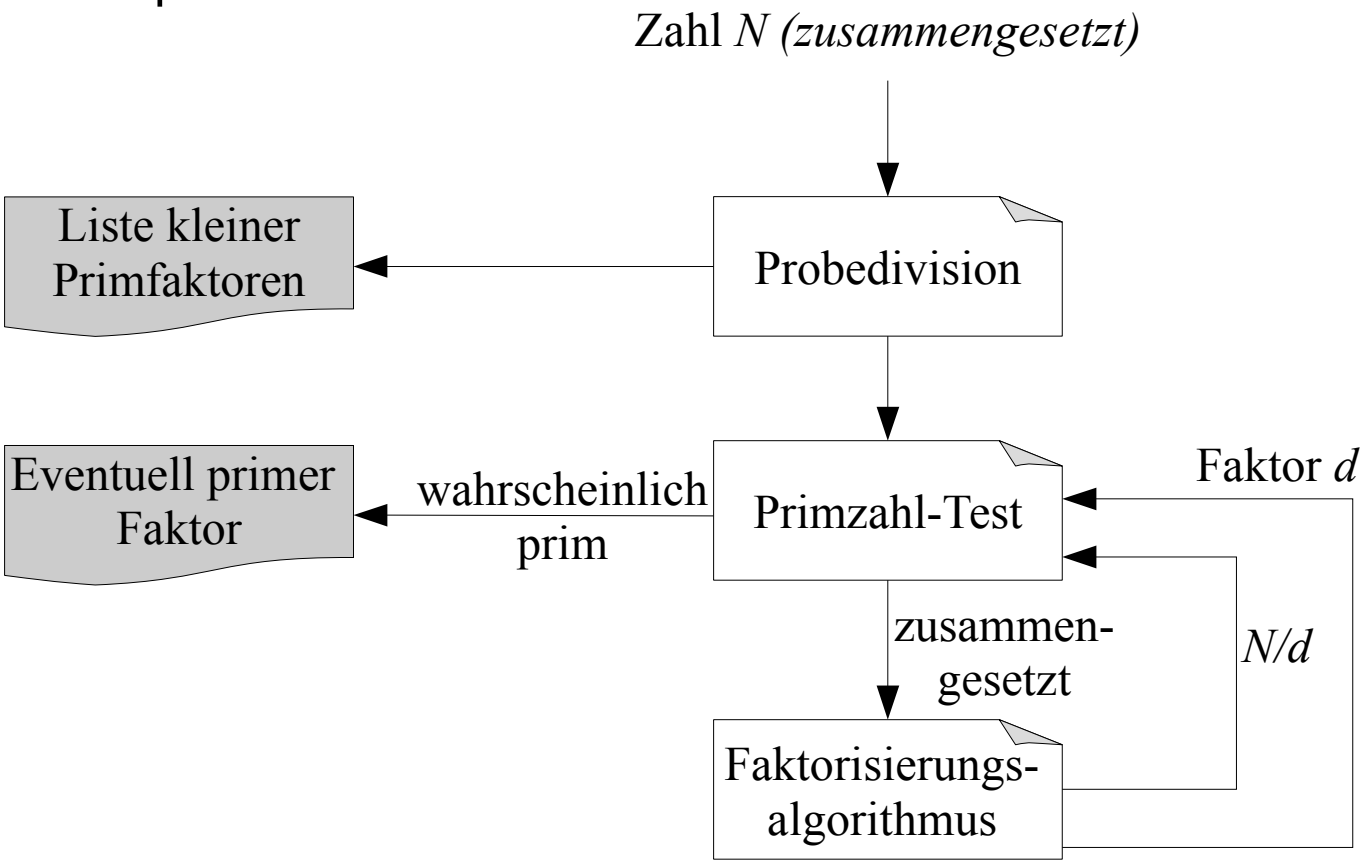
Probedivision

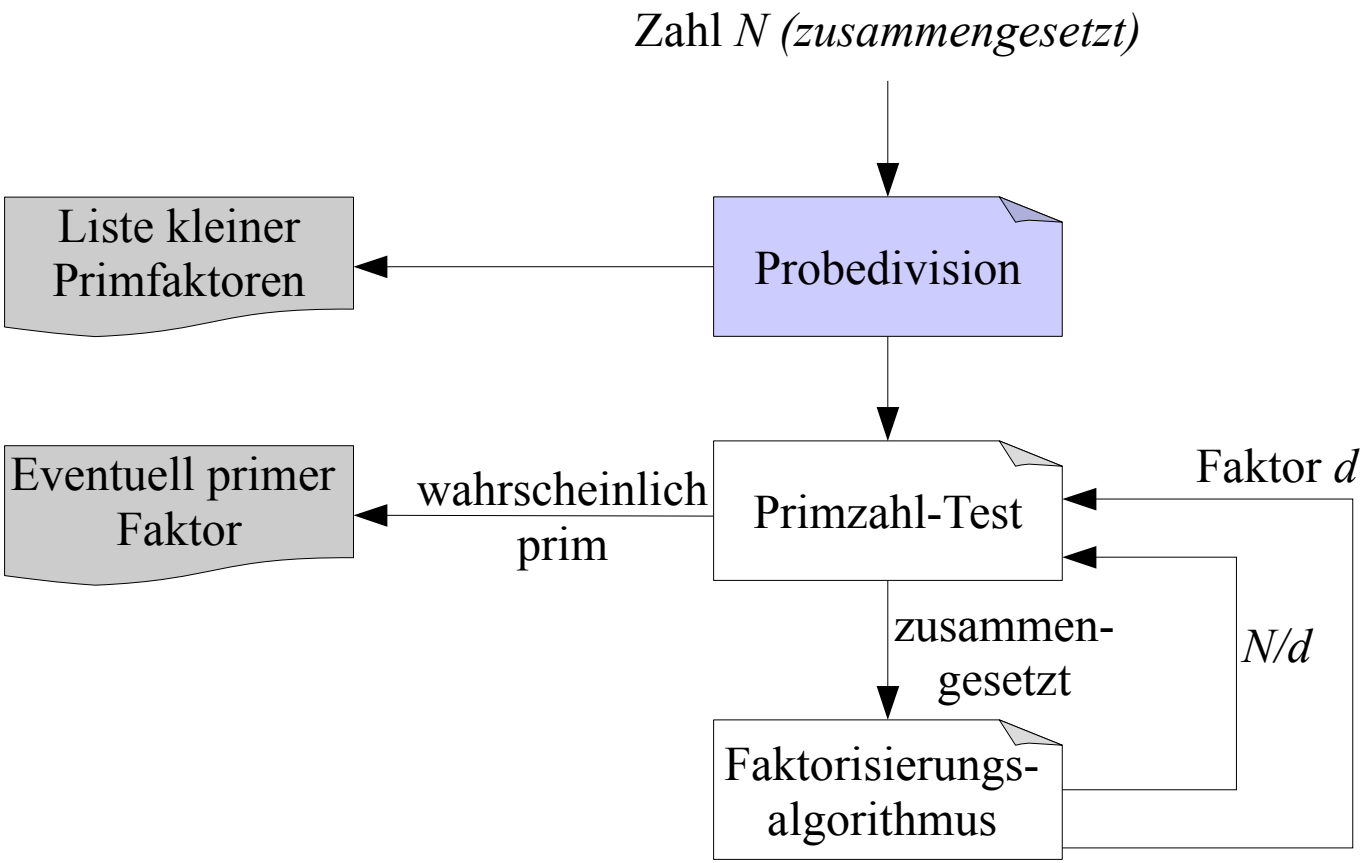
Primzahltest

Moderne Faktorisierungsalgorithmen

Ablauf

- **Eindeutige** Zerlegung $N = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_s^{e_s}$ (Primzahlen p_i, Exponenten e_i)
- Zerlegung in der Praxis **nicht in polynomialer Zeit** durchführbar
- Ablaufplan:




Zahl *N (zusammengesetzt)*
Liste kleiner Primfaktoren
Probedivision
Eventuell primer Faktor
wahrscheinlich prim
Primzahl-Test
Faktor *d*
zusammen-gesetzt
N/d
Faktorisierungs-algorithmus

Probedivision: Einführung

- **Einfache Methode** zur Faktorisierung einer Zahl N , meist $\leq 10^6$
- Beschränkung auf kleine Primfaktoren (p_t) wegen **exponentieller Laufzeit**
- Zur Durchführung wird eine **Liste von Divisoren** (d_1 .. d_i) benötigt
  - Für ein Listenelement muss $d_k \geq \sqrt{N}$ gelten, sonst $\leq \sqrt{N}$
  - Verschiedene Ansätze zum **Aufbau der Liste**

    a) Feste **Speicherung** zu prüfender Primzahlen

    b) **Berechnung** der Primzahlen, z. B. Sieb des Eratosthenes

    c) alle positiven Zahlen der **Form** $6 \cdot t \pm 1 \; mit \; t \geq 1$ (sowie 2 und 3)
    (liefert 2,3,5,7,11,13,17,19,23,25,29,31,35,...)

Probedivision: Pseudocode

- Listing

```
1  public Probedivision(N∈ℕ , d_1 .. d_i ∈ℕ){
2    t = 0; k = 0; n = N;
3    //es gilt: n=N/(p_1*..*p_t) && n hat keine Primfaktoren <d_k
4    while (n != 1) {
5      q = n DIV d_k; r = n MOD d_k;
6      if (r = 0) { // Faktor gefunden
7        t++; n = q; p_t = d_k; // Faktor speichern
8      } else {
9        if (q > d_k) { k++; } // nächsten Faktor probieren
10       else {
11         t++; p_t = n; // n ist prim
12         return; // Ende des Algorithmus
13       }
14     }
15   }
16 }
```

Probedivision: Beispiel

- *Beispiel (Zerlegung $N=4242$)*
  - Divisoren der Form $6 \cdot t \pm 1 \; mit \; t \geq 1$ (sowie 2 und 3)
    (2,3,5,7,11,13,17,19,23,25,29,31,35,...67 $\geq \sqrt{4242}$)

t	k	d_k	n	q	r	p_t
0	0	2	4242	2121	0	$\Rightarrow p_1=2$
1	0	2	2121	1060	1	
1	1	3	2121	707	0	$\Rightarrow p_2=3$
2	1	3	707	235	2	
2	2	5	707	141	2	
2	3	7	707	101	0	$\Rightarrow p_3=7$
3	3	7	101	14	3	
3	4	11	101	9	2	$\Rightarrow p_4=101$

  - Algorithmus terminiert, da $q \leq d_k \Leftrightarrow 9 \leq 11 \Rightarrow$ 101 Primfaktor
  - Ergebnis: $4242=2 \cdot 3 \cdot 7 \cdot 101$ in 8 Schritten
  - Für die Zahl $N=4243$ (Primzahl) mindestens 19 Schritte erforderlich

Probedivision: Existenz kleiner Teiler

- **Test auf Existenz** von kleinen Teilern
  - Probedivision kann **übersprungen** werden, falls keine kleinen Teiler vorhanden sind
  - Bildung des **Produkts** P **der Divisoren** bis zu einer oberen Grenze
  - Bei gleich bleibender Grenze als **Konstante** realisierbar
  - $GGT(P, N)$ ermittelt vergleichsweise **schnell** , ob N mindestens einen Teiler aus der Divisorenliste besitzt
    - $GGT > 1$: es existiert **mindestens ein** Teiler $\Rightarrow$ Probedivision
    - $GGT = 1$: **keine** Teiler vorhanden $\Rightarrow$ Probedivision überspringen

Probedivision: Beispiel

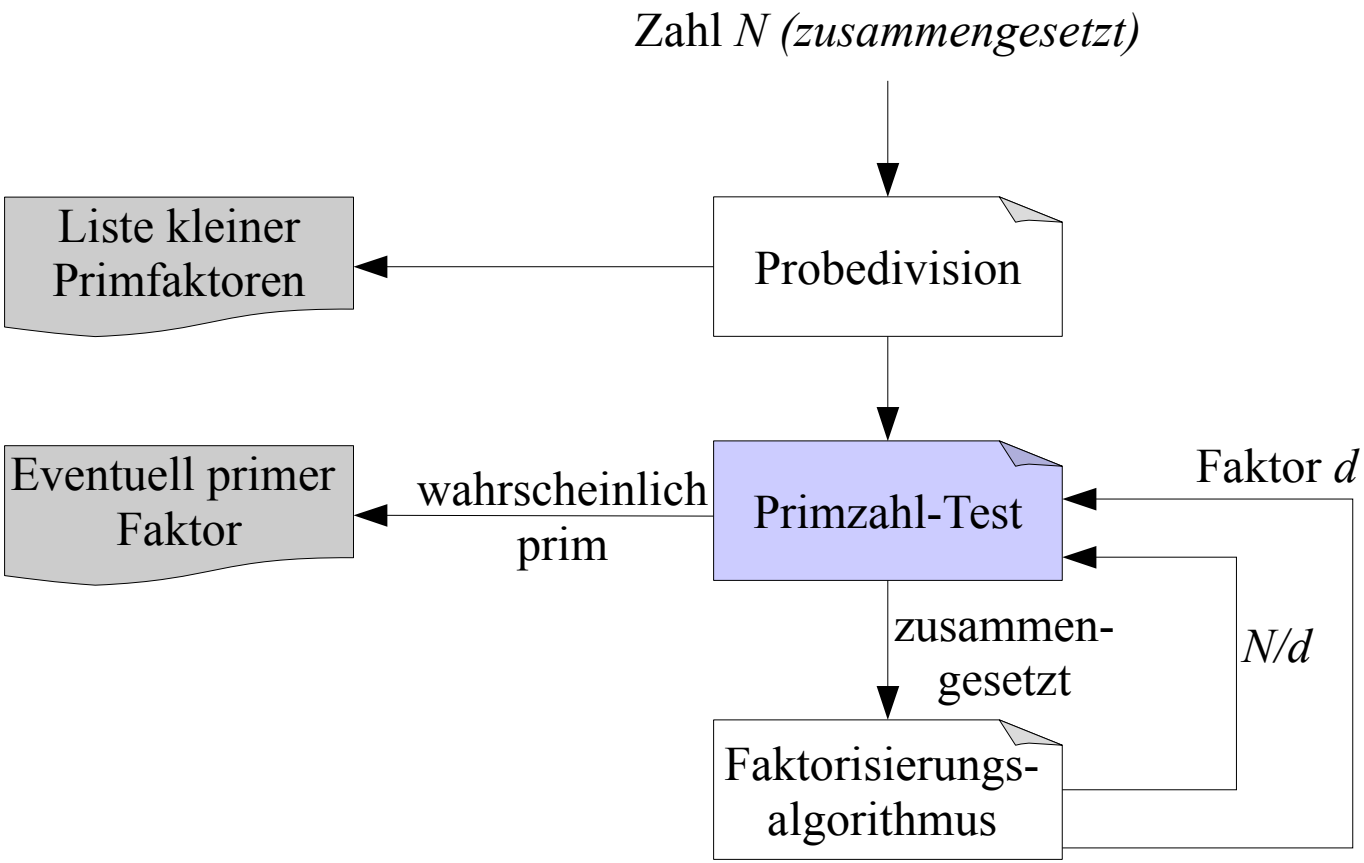
- Betrachtung der Teiler bis obere Grenze 71
  $P = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71$
  $P = 557940830126698960967415390$

- Beispiel (Test, ob Zahl $N = 731771 = 53 \cdot 13807$ kleine Teiler enthält)
  - Berechnung des GGT
    $GGT(P, N) = GGT(557940830126698960967415390,\ 731771) = 53$
  - $GGT > 1 \Rightarrow$ Probedivision durchführen

- Beispiel (Test, ob Zahl $N = 544909 = 163 \cdot 3343$ kleine Teiler enthält)
  - Berechnung des GGT
    $GGT(P, N) = GGT(557940830126698960967415390,\ 544909) = 1$
  - $GGT = 1 \Rightarrow$ Probedivision überspringen, da keine Teiler bis 71

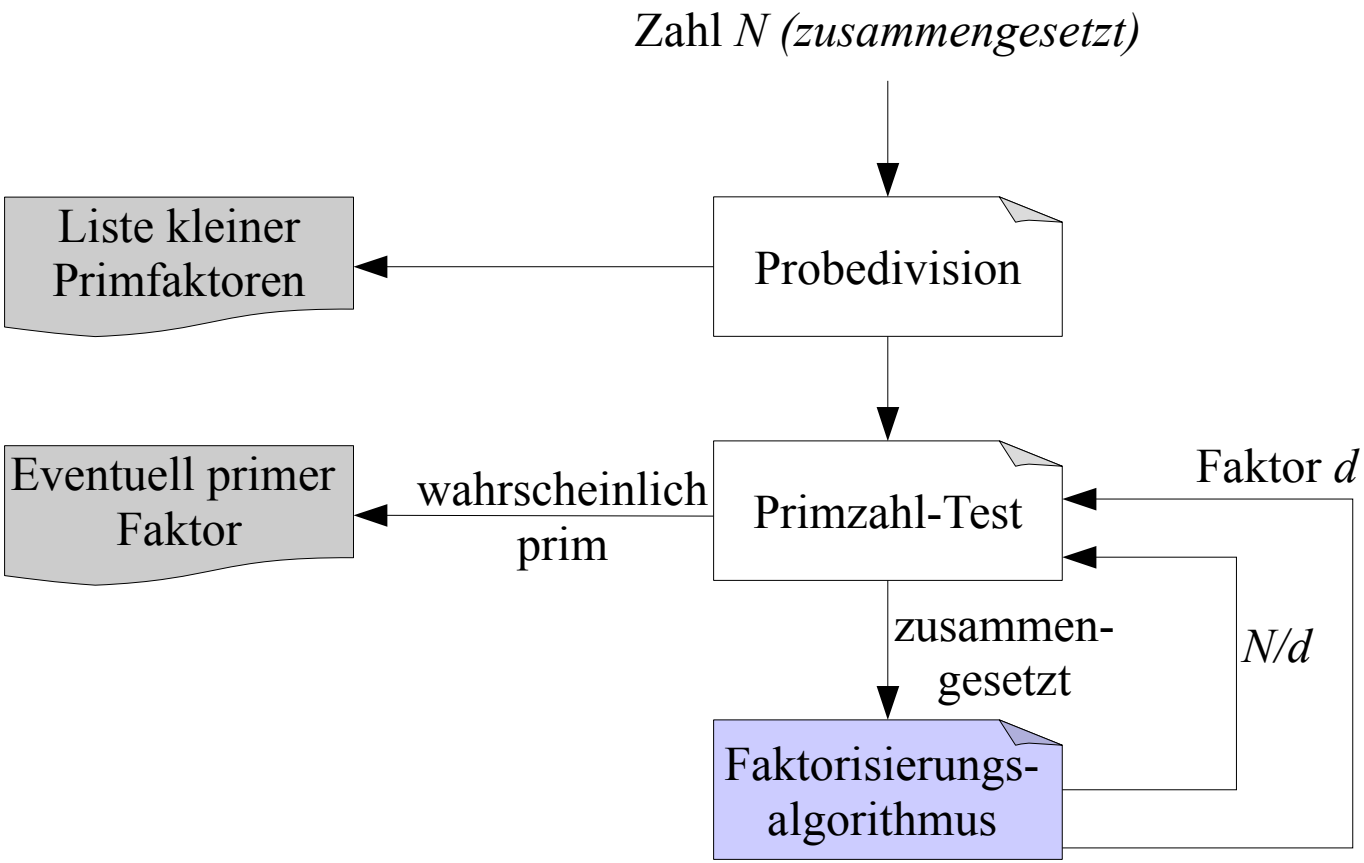
Probedivision: Bewertung

- Geeignet für Zahlen $\leq 10^6$, da nur Divisoren bis $\sqrt{10^6} = 1000$
- Bei **größeren Zahlen** nur Prüfung bis zu einer **oberen Grenze**
  - Ermittlung **kleiner Primfaktoren**
  - **Effizientere Verfahren** zur Zerlegung der restlichen Zahl
- Laufzeit: $O(2^{n/2})$ für Länge n der Zahl N


Zahl *N (zusammengesetzt)*
Liste kleiner Primfaktoren
Probedivision
Eventuell primer Faktor
wahrscheinlich prim
Primzahl-Test
Faktor *d*
zusammen-gesetzt
N/d
Faktorisierungs-algorithmus

Primzahl-Test

- **Faktorisierungsverfahren** arbeiten vergleichsweise **langsam**
- Vorab **prüfen**, ob Zahl **zusammengesetzt** ist
  - Berechnung erfolgt **schnell**
  - Aussage „*ist zusammengesetzt*“ ist sicher **richtig**
  - Aussage „*ist Primzahl*“ ist mit **geringer Wahrscheinlichkeit falsch** (*strenge Pseudoprimzahlen*)
  - Keine Aussage über Teiler
  - z. B. *Fermattest* oder *Rabin-Miller-Test* (*Koepf s. Kapitel 4)*


Zahl *N (zusammengesetzt)*
Liste kleiner Primfaktoren
Probedivision
Eventuell primer Faktor
wahrscheinlich prim
Primzahl-Test
Faktor *d*
zusammen-gesetzt
N/d
Faktorisierungs-algorithmus

Moderne Faktorisierungsalgorithmen: Einführung

- **Kein praxistaugliches Verfahren** mit polynomialer Laufzeit
- **Theoretischer Ansatz** um polynomiale Laufzeit erreichen zu *können*
- Viele unterschiedliche Verfahren
  - **Allgemeine**: keine Annahmen über zu zerlegende Zahl
  - **Spezielle**: erfordern besondere Annahmen über die Form, ...
- Meist **hintereinander angewendet**, da *ein* Verfahren nicht alle Faktoren mit *einem* Durchlauf liefert

Moderne Faktorisierungsalgorithmen: Exkurs

- Derzeitiger **Weltrekord**: Zerlegung einer **200-stelligen** Zahl (Mai 2005)
  - Ein 2,2 GHz-Pentium Rechner hätte **55 Jahre** gebraucht
  - Größter Rechenschritt dauerte **3 Monate** mit einem Cluster von 80 2,2 GHz-Pentium u. a. an der Universität Bonn (Projekt startete Ende 2003)
  - RSA-200:
    27997833911221327870829467638722601621070446786955428537560009929326128400107609345671052955360856061822351910951365788637105954482006576775098580557613579098734950144178863178946295187237869221823983
  - Faktoren:
    35324619344027701212726049781984643686711974001976250236493034687761212536794232000585479565280883491
    und
    79258699544783330333470858414800596877379758573642199607343303414557678728181521353814093047401854671

Moderne Faktorisierungsalgorithmen: Shor-Algorithmus

- *Shor-Algorithmus* als theoretischer Ansatz mit **polynomialer Laufzeit**
  - 1994 entwickelt
  - Arbeitet auf **Quantencomputer** mit *Quanten-Bits*
    - Quanten-Bits: Atome mit verschiedenen **Quantenzuständen**
    - Quanten-Bits arbeiten als **Prozessor und Speicher**
  - Laufzeit: $O((\log N)^2(\log\log N)(\log\log\log N))$ auf Quantencomputer und $O(\log N)$ zur Nachverarbeitung auf normalen Computer
  - 2001 konnte IBM mit einigem Aufwand die **Zahl 15** zerlegen (Quantencomputer mit 7 Quanten-Bits)
  - Verfahren (noch) nicht praktisch anwendbar, da Quantencomputer **technisch sehr aufwendig** sind

Kapitel 5

Anwendung

Kryptografie

Kryptografie

- E-Mails, E-Banking, Filetransfer in einem **offenen System** (Internet)
  - **Gefahr** vor dem Zugriff Dritter
  - **Verschlüsselung** notwendig (z. B. Public-Key-Verfahren *RSA*)
  - Nutzung von Konzepten der **Zahlentheorie**
    - **Erweiterter Euklidischer Algorithmus** zur Schlüsselerzeugung
    - Schwierigkeit der **Faktorisierung** als „Garant“ für die Sicherheit
- Schlüsselerzeugung bei RSA
  - Wahl zweier gleichgroßer Primzahlen (mindestens 512 Bit)
  - Privater Schlüssel mit erweitertem Euklidischen Algorithmus berechenbar
- **Verschlüsselung** als mathematische Berechnung mittels öffentlichem Schlüssel
- **Entschlüsselung** nur mittels privatem Schlüssel durchführbar

Kryptografie

- **Sicherheit** ist gewährleistet, da die Faktorisierung (noch) nicht in polynomialer Laufzeit berechenbar ist
  - (fast) **unmöglich**, den privaten aus dem öffentlichen Schlüssel zu berechnen
  - *Einwegfunktion*: Multiplikation leicht, Faktorisierung schwer lösbar
- Schlüssel mindestens 1024 Bit = $1024 \cdot \log(2) : \log(10) \approx 309$ Stellen
- **Offenes Problem**, ob geheimer Schlüssel notwendig ist, zum Entschlüsseln einer Nachricht
- RSA wäre zu **ersetzen**, wenn sich die Faktorisierung in Zukunft als leicht lösbares Problem herausstellen sollte

Kapitel 6

Fazit

- **Euklidischer Algorithmus** seit 300 v. Chr. bekannt
- Nur geringe Neuerungen wie der **erweiterte Euklidische Algorithmus** (499 n. Chr.)
- Algorithmen noch heute von **großem Interesse**
  - Schnelle **GGT**-Berechnung
  - Darstellung des GGT als **Linearkombination**
  - **Vereinfachung von Brüchen**
  - Nutzung in Beweisen anderer Algorithmen und als Hilfssätze z. B. *Chinesischer Restsatz (Koepf s. Kapitel 4)*

- Faktorisierung ist eine sehr **dynamische** Disziplin
  - **Ab 1970** erste Algorithmen zur „schnellen“ Zerlegung von Zahlen bis 40 Stellen
  - **Nur 30 Jahre später** existieren Verfahren für 200-stellige Zahlen
- Zunahme der **Leistungsfähigkeit** als Bedrohung für Verschlüsselung
  - Annahme, dass keine Zerlegung in polynomialer Zeit möglich ist
- Faktorisierung ist ein wichtiges Gebiet der Computeralgebra in dem **weitere Verfahren und Techniken** zu erwarten sind

Vielen Dank für die Aufmerksamkeit ! ! !

Fragen . . . ? ? ?