

# SEMINARARBEIT

in der Fachrichtung

Wirtschaftsinformatik

Thema:

## Langzahlarithmetik: Vereinfachen von Brüchen und Faktorisierung

Eingereicht von: Thomas Stuht  
Krautstücken 2  
22589 Hamburg  
Tel.: 040 – 879 32 209  
thomas.stuht@web.de

Matrikel-Nummer: winf2967

Erarbeitet im: 5. Semester

Abgegeben am: 5. Dezember 2007

Seminar: Informatik-Seminar  
Computer-Algebra

Dozent: Prof. Dr. Sebastian Iwanowski  
Fachhochschule Wedel  
Feldstraße 143  
22880 Wedel

## Inhaltsverzeichnis

Inhaltsverzeichnis.....	II
Abbildungen.....	III
Listings.....	III
Akronyme.....	III
Einleitung.....	1
1 Grundlegende Konzepte der Algebra .....	2
1.1 Euklidischer Algorithmus.....	2
1.2 Erweiterter Euklidischer Algorithmus.....	3
1.3 GGT für $n$ ganze Zahlen.....	8
2 Rationale Arithmetik.....	10
2.1 Vereinfachen von Brüchen.....	10
2.2 Rechenoperationen.....	11
3 Algorithmen zur Faktorisierung.....	15
3.1 Probedivision.....	16
3.2 Primzahl-Test.....	20
3.3 Moderne Faktorisierungsalgorithmen.....	20
3.3.1 Übersicht.....	20
3.3.2 Shor-Algorithmus.....	21
4 Fazit und Ausblick.....	22
Anhang: Computeralgebra in der Praxis: Kryptografie.....	23
A Motivation der Kryptografie.....	23
B RSA.....	24
B.1 Funktionsweise.....	24
B.2 (Un)Sicherheit.....	26
Literaturverzeichnis.....	28

## Abbildungen

Faktorisierung von ganzen Zahlen.....	15
---------------------------------------	----

## Listings

Euklidischer Algorithmus.....	2
Erweiterter Euklidischer Algorithmus.....	5
GGT für n ganze Zahlen.....	8
Probedivision.....	17

## Akronyme

GGT	Größter Gemeinsamer Teiler
KGV	Kleinstes Gemeinsame Vielfache

## Einleitung

Grundlegende Konzepte wie der bekannte Euklidische Algorithmus dienen nicht nur zur GGT-Berechnung und als Hilfssätze in Beweisen. Sie sind auch wichtig für Vereinfachungen von Brüchen, wie sie in dieser Arbeit gezeigt werden. Durch „Hingucken“ oder Intuition sind Brüche wie  $\frac{2318068701}{2436944019}$  nicht zu kürzen.

In der Computeralgebra spielt die Faktorisierung ganzer Zahlen eine zentrale Rolle. Zwar gibt es Verfahren, die intuitiv und einfach anzuwenden sind, aber nur für Zahlen bis  $\leq 10^6$  effizient funktionieren. Da in der Praxis jedoch mit viel größeren Zahlen gearbeitet wird, ist die Laufzeitkomplexität eine entscheidende Größe. Verschiedene Verfahren wurden zu diesem Zweck entwickelt, um die Zerlegung möglichst effizient zu gestalten.

Im ersten Kapitel der Ausarbeitung werden grundlegende mathematische Konzepte erläutert und anhand von Beispielen präsentiert.

Im zweiten Kapitel wird die rationale Arithmetik<sup>1</sup> erläutert und die Vereinfachung von Brüchen vorgestellt.

Im dritten Kapitel wird ein ausgewähltes Verfahren zur Faktorisierung erläutert und anhand eines Beispiels dargestellt. Außerdem wird auf ein Verfahren der aktuellen Forschung eingegangen und dessen Bedeutung beschrieben.

Am Ende der Ausarbeitung befindet sich ein Fazit in Bezug auf die große Bedeutung, die unter anderem die Faktorisierung besitzt.

Im Anhang wird eine Praxisanwendung für die vorgestellten Verfahren gezeigt, indem das bekannte Verschlüsselungsverfahren RSA diskutiert wird.

Die Algorithmen werden in einer Java-ähnlichen Syntax notiert und anhand von Beispielen eingeführt. Zudem liegt der Ausarbeitung eine Implementierung des erweiterten Euklidischen Algorithmus für das Computeralgebrasystem Maxima (<http://maxima.sourceforge.net/>; Open-Source) bei.

Hamburg, Dezember 2007

Thomas Stucht

---

<sup>1</sup> Aus didaktischen Gründen wird dieses Thema vor der Faktorisierung behandelt, da die rationale Arithmetik direkt Bezug auf Kapitel 1 nimmt.

# 1 Grundlegende Konzepte der Algebra

Dieses Kapitel beschreibt ausgewählte Algorithmen, die in vielen Gebieten der Computeralgebra Anwendung finden. Sie bilden die Basis für weitere Verfahren oder werden in deren Beweisen genutzt. So wird der erweiterte Euklidische Algorithmus (siehe Kapitel 1.2) in der modularen Arithmetik unter anderem zur Berechnung von inversen Elementen innerhalb von Restklassenringen<sup>2</sup> verwendet. Es folgt eine Beschreibung der „Grundwerkzeuge der Algebra“.

**Grund-  
konzepte**

## 1.1 Euklidischer Algorithmus

Einer der wichtigsten Algorithmen in der Zahlentheorie ist der Euklidische Algorithmus. Durch wiederholtes Anwenden einer Division mit Rest<sup>3</sup> lässt sich der GGT (und somit auch das KGV<sup>4</sup>) berechnen. Er ist deutlich schneller in der Berechnung des GGT als zum Beispiel die Primfaktorzerlegung durch Faktorisierung (siehe Kapitel 3). Die Laufzeit beträgt  $O(n \cdot m)$  für zwei ganze Zahlen der Länge  $n$  und  $m$ .

**Division mit  
Rest**

**Laufzeit**

Seien  $x$  und  $y$  beliebige ganzen Zahlen, dann gilt die Gleichung

$$x = q \cdot y + r \quad \text{mit } 0 \leq r < y \quad (1)$$

für genau ein Paar (*Quotient*  $q$ , *Rest*  $r$ ). Weiterhin gilt die Beziehung

$$\text{GGT}(x, y) = \text{GGT}(y, r) \quad . \quad (2)$$

Die Berechnung<sup>5</sup> erfolgt nun wie folgt:

---

```

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

```

---

**Algorithmus**

*Listing 1: Euklidischer Algorithmus*

<sup>2</sup> vgl. [Koe2006], S. 91

<sup>3</sup> vgl. [Koe2006], S. 64 ff.

<sup>4</sup> Es gilt:  $\text{GGT}(x, y) \cdot \text{KGV}(x, y) = x \cdot y$  (vgl. [Iwa2007], S. 5)

<sup>5</sup> in Anlehnung: [Iwa2007], S. 7

Gilt  $GGT=1$ , dann werden die beiden Zahlen *teilerfremd* genannt.

**Definition**

Beispiel A: Das folgende einführende Beispiel wird in späteren Kapiteln erneut aufgegriffen. Gesucht wird der GGT von  $x=3564$  und  $y=2727$ .

**Beispiel**

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

Für die Berechnung des GGT waren 5 Iterationen notwendig. Hierbei wird auch von der Euklidischen Länge<sup>6</sup> gesprochen. Der GGT beträgt im Beispiel 27.

## 1.2 Erweiterter Euklidischer Algorithmus

Der „einfache“ Euklidische Algorithmus (siehe Kapitel 1.1) berechnet *nur* den GGT ganzer Zahlen. Der erweiterte Euklidische Algorithmus kann darüber hinaus auch den GGT (und alle Zwischenrestwerte) als Linearkombination der beiden Eingabeparameter darstellen<sup>7</sup>.

**Linearkombination**

Der erweiterte Euklidische Algorithmus ist grundlegend für viele weitere Algorithmen in der Zahlentheorie. Besondere Anwendung findet dieses Verfahren bei der Berechnung von inversen Elementen innerhalb von Restklassenringen<sup>8</sup>, wie sie in Kapitel 4 des Buches „Computeralgebra“ von Wolfram Koepf beschrieben sind und im Chinesischen Restsatz verwendet werden. Auch in der Kryptografie zur Schlüsselerzeugung (siehe Anhang) ist dieser Algorithmus von großem Interesse.

**Motivation**

Das Verfahren liefert folgende Darstellung, mit  $s$  und  $t$  als ganze Zahlen,

$$GGT(x, y) = x \cdot s + y \cdot t \text{ mit } x, y, s, t \in \mathbb{Z} . \quad (3)$$

$s$  und  $t$  werden auch als Bézoutkoeffizienten<sup>9</sup> bezeichnet.

**Bézoutkoeffizienten**

<sup>6</sup> vgl. [Koe2006], S. 64 ff.

<sup>7</sup> vgl. [Koe2006], S. 70 ff.

<sup>8</sup> vgl. [Coh1993], S. 18

<sup>9</sup> vgl. [Koe2006], S. 70

Es lässt sich jedoch auch das Ergebnis des einfachen Euklidischen Algorithmus nutzen, um in einem zweiten Schritt den GGT als Linearkombination darzustellen. Dies stellt somit ein *alternatives Vorgehen* zum erweiterten Euklidischen Algorithmus dar.

**Alternative**

Im Folgenden wird zunächst erklärt, wie mithilfe des einfachen Euklidischen Algorithmus die Linearkombination des GGT berechnet werden kann. Im Anschluss wird dann die „echte“ Erweiterung vorgestellt.

**Einfacher Euklidischer Algorithmus**

Wird der einfache Euklidische Algorithmus genutzt, dann müssen nachträglich die einzelnen Gleichungen nach  $r$  umgeformt werden<sup>10</sup>, sodass

$$x = q \cdot y + r \Leftrightarrow x - q \cdot y = r \quad (4)$$

gilt. Diese Umformung ist jedoch erst möglich, wenn der Algorithmus den GGT als Ergebnis wie in Beispiel A (siehe Kapitel 1.1) berechnet hat. Ausgehend von der letzten Zeile aufwärts, werden nun rekursiv die Restwerte durch die umgeformten Gleichungen ersetzt. Bildlich gesprochen werden die Gleichungen von unten nach oben verwendet, bis die oberste Zeile das gewünschte Endergebnis liefert.

Beispiel B1: Ausgehend vom Ergebnis aus Beispiel A soll nun der ermittelte GGT als Linearkombination der Eingangsparameter dargestellt werden. Im Beispiel B2 wird dann die Erweiterung des Algorithmus verwendet, um vergleichen zu können, ob sie eine Verbesserung darstellt.

**Beispiel**

Das bisherige Ergebnis für den GGT von  $x=3564$  und  $y=2727$  wird zunächst entsprechend Gleichung (4) nach  $r$  umgeformt:

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

Begonnen wird nun mit der letzten Zeile, also mit der umgeformten Darstellung des GGT. Es werden schrittweise die Restwertdarstellungen von unten nach oben eingesetzt. Die ersetzten Werte sind jeweils durch Unterstreichung gekennzeichnet.

<sup>10</sup> vgl. [vGG2003], S. 45ff.

Dieses Vorgehen führt zu folgender Rechnung:

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

Die gesuchte Linearkombination lautet daher  $3564 \cdot (-13) + 2727 \cdot 17 = 27$ .

Wird der einfache Euklidische Algorithmus verwendet, so ist ein zusätzlicher Rechenaufwand notwendig. Es muss erst der GGT vollständig berechnet werden, um dann in einem *zweiten Schritt* diesen als Linearkombination der Eingabeparameter darzustellen. Auch festzuhalten ist, dass ausschließlich der GGT als Linearkombination von  $x$  und  $y$  dargestellt werden kann.

**Einschränkungen**

Im Folgenden wird gezeigt, dass der erweiterte Euklidische Algorithmus eine echte Verbesserung darstellt. Er kann nicht nur den GGT berechnen, sondern auch den GGT und alle Zwischen-Restwerte als Linearkombination von  $x$  und  $y$  darstellen. Diese Berechnung erfolgt parallel zur GGT-Berechnung, indem die Bézoutkoeffizienten bei jedem Durchlauf so verändert werden, dass am Ende jedes Iterationsschrittes  $x \cdot s_i + y \cdot t_i = r_i$  gilt. Ein nachträgliches Umformen von Gleichungen ist nicht mehr notwendig.

**Verbesserung**

Seien  $x$  und  $y$  beliebige Werte aus dem Bereich der ganzen Zahlen, dann verläuft die Berechnung<sup>11</sup> des erweiterten Euklidischen Algorithmus wie folgt:

---

```

1 public GGT( x∈ℤ , y∈ℤ ) {
2   r0 = x; s0 = 1; t0 = 0; // init
3   r1 = y; s1 = 0; t1 = 1;
4   i = 1;
5   while (ri != 0) {
6     qi = ri-1 DIV ri;
7     ri+1 = ri-1 - qi * ri;
8     si+1 = si-1 - qi * si;
9     ti+1 = ti-1 - qi * ti;
10    i++;
11  }
12  return GGT = ri-1;
13 }

```

---

**Algorithmus**

*Listing 2: Erweiterter Euklidischer Algorithmus*

<sup>11</sup> in Anlehnung an: [vGG2003], S. 45ff.

Die im Codefragment verwendeten  $q_i$  entsprechen den Quotienten,  $r_i$  den Restwerten des „einfachen“ Euklidischen Algorithmus. Dies verdeutlicht die enge Verwandtschaft zwischen dem erweiterten und dem einfachen Euklidischen Algorithmus. Neu sind die Koeffizienten  $s_i$  und  $t_i$ , die für die Linearkombination benötigt werden. Vereinfachend gibt der Pseudocode nur den GGT aus, für die Linearkombinationen wären die Koeffizienten entsprechend zwischenspeichern.

**Beziehung zum Euklidischen Algorithmus**

Beispiel B2: Es wird erneut das Beispiel A aufgegriffen, aber nun der eben beschriebene, erweiterte Euklidische Algorithmus verwendet. Gesucht wird der GGT von  $x=3564$  und  $y=2727$ , sowie dessen Linearkombination. Das Verfahren liefert folgende Tabelle:

**Beispiel**

$i$	$q_i$	$r_i$	$s_i$	$t_i$	$x \cdot s_i + y \cdot t_i = r_i$
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	$3564 \cdot 101 + 2727 \cdot (-132) = 0$

Nicht überraschend ergibt sich auch hier als Ergebnis  $GGT=27$ , welches aus der vorletzten Zeile abgelesen werden kann. Das Besondere an dieser Erweiterung ist jedoch die Darstellung des berechneten GGT als Linearkombination der beiden Parameter  $x=3564$  und  $y=2727$ . Auch diese kann der vorletzten Zeile entnommen werden:  $3564 \cdot (-13) + 2727 \cdot 17 = 27$ .

Die Linearkombination wird sozusagen als „Seiteneffekt“ des Algorithmus gleich mitgeliefert. Ein nachträgliches, wiederholtes Einsetzen der Restwertdarstellungen ist hier nicht erforderlich. Außerdem fällt eine weitere besondere Eigenschaft an dieser Auflistung auf: für die Werte von jeder Zeile gilt die folgende Vorschrift:  $x \cdot s_i + y \cdot t_i = r_i$ .

**Vorteile der „Erweiterung“**

Der Restwert  $r_i$  von *jedem* Zwischenschritt wird so automatisch als Linearkombination von  $x$  und  $y$  dargestellt.

Zusammenfassend lässt sich sagen, dass auch mithilfe des einfachen Euklidischen Algorithmus eine Linearkombination berechenbar ist, aber nur mit einem größeren Aufwand. Außerdem muss zunächst der GGT vollständig berechnet sein, um danach von unten nach oben zurückzurechnen.

**Vergleich**  
„Einfach“ und  
„Erweitert“

Die Laufzeit des erweiterten Euklidischen Algorithmus beträgt  $O(n \cdot m)$  für zwei ganze Zahlen der Länge  $n$  und  $m$ <sup>12</sup>. Der schlimmste Fall wären zwei aufeinander folgende Fibonaccizahlen als Eingabeparameter. Die Fibonaccizahlen sind definiert als  $F_n = F_{n-1} + F_{n-2}$  mit  $F_0 = 0$ ,  $F_1 = 1$ ,  $n \geq 2$ .

**Laufzeit**

Beispiel B3: Wird das Verfahren mit den Fibonaccizahlen  $F_8=21$  und  $F_7=13$  gestartet, dann liefert es das folgende Ergebnis:

**Beispiel**

$i$	$q_i$	$r_i$	$s_i$	$t_i$	$x \cdot s_i + y \cdot t_i = r_i$
0		21	1	0	$21 \cdot 1 + 13 \cdot 0 = 21$
1	1	13	0	1	$21 \cdot 0 + 13 \cdot 1 = 13$
2	1	8	1	-1	$21 \cdot 1 + 13 \cdot (-1) = 8$
3	1	5	-1	2	$21 \cdot (-1) + 13 \cdot 2 = 5$
4	1	3	2	-3	$21 \cdot 2 + 13 \cdot (-3) = 3$
5	1	2	-3	5	$21 \cdot (-3) + 13 \cdot 5 = 2$
6	2	1	5	-8	$21 \cdot 5 + 13 \cdot (-8) = 1$
7		0	-13	21	$21 \cdot (-13) + 13 \cdot (21) = 0$

Alle Quotienten, mit Ausnahme der letzten Zeile, betragen 1 und als Reste ergeben sich der Reihe nach alle kleineren Fibonaccizahlen. Obwohl  $F_1 = F_2 = 1$  ist, kommt die 1 in den Restwerten nur einmalig vor, da die Restwerte stetig fallend sind. Der GGT beträgt stets 1 (teilerfremd).

<sup>12</sup> vgl. [Koe2006], S. 72-73

### 1.3 GGT für n ganze Zahlen

Bislang wurde stets der GGT von zwei ganzen Zahlen berechnet. Dies lässt sich jedoch verallgemeinern, indem der GGT von n ganzen Zahlen berechnet wird.

**Verallgemeinerung**

Diesem Verfahren liegt folgende Gleichung zugrunde:

$$GGT(u_1, u_2, \dots, u_n) = GGT(u_1, GGT(u_2, \dots, u_n)) \quad (5)$$

Des Weiteren gilt, dass  $GGT(u_1, u_2, \dots, u_n, 1) = 1$ . Folglich müssen in vielen Fällen gar nicht alle Berechnungen durchgeführt werden, sondern es kann bereits vorab aufgehört werden, wenn bekannt ist, dass mindestens zwei Zahlen teilerfremd sind.

Das Gesamtproblem wird somit auf mehrere Ausführungen der „einfachen“ Algorithmen zur GGT-Berechnung zweier Zahlen herunter gebrochen. So lassen sich diese Verfahren wiederverwenden.

**Wiederverwendung**

Die Laufzeit dieses Verfahrens wird im Wesentlichen durch die Komplexität der GGT-Berechnung bestimmt, da in jedem Durchlauf ein GGT zweier ganzer Zahlen zu berechnen ist. Vereinfachend sei angenommen, alle Zahlen sind gleichlang und haben die Länge  $m$ . Für den GGT von  $n$  Zahlen beträgt die Laufzeit des Verfahrens somit  $O(n \cdot m^2)$ .

**Laufzeit**

Wird der GGT von  $u_1, u_2, \dots, u_n$  mit  $n \geq 1$  gesucht, dann lässt er sich wie folgt berechnen:

---

```

1 public NZahlenGGT( $u_1 \dots u_n \in \mathbb{Z}$ ) {
2    $d = u_n$ ;  $k = n-1$ ; // init
3
4   while ( $d \neq 1 \ \&\& \ k > 0$ ) {
5      $d = GGT(u_k, d)$ ;
6      $k--$ ;
7   }
8   return  $NZahlenGGT = d$ ;
9 }
```

---

**Algorithmus**

Listing 3: GGT für n ganze Zahlen

Beispiel C1: Gesucht wird der GGT von den Zahlen  $u_1=14$  ,  $u_2=147$  , **Beispiel**  
 $u_3=42$  ,  $u_4=294$  und  $u_5=693$  .

$d$	$k$
$u_5 = 693$	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

Die Verarbeitung ist beendet, da  $k=0$  ist. Der GGT dieser Zahlen beträgt 7.

Beispiel C2: In diesem Beispiel wird der GGT von den Zahlen  $u_1=14$  , **Beispiel**  
 $u_2=147$  ,  $u_3=113$  (!) ,  $u_4=294$  und  $u_5=693$  gesucht.

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

Die Verarbeitung ist beendet, da  $d=1$  ist. Bereits nach der dritten Zahl ist bekannt, dass diese fünf Zahlen teilerfremd sind. Die restlichen Zahlen müssen nicht mehr betrachtet werden. Der GGT dieser Zahlen beträgt 1.

## 2 Rationale Arithmetik

In Kapitel 1 wurde der einfache und der erweiterte Euklidische Algorithmus vorgestellt, womit der GGT berechnet werden kann. Dieses Wissen kann nun zum Vereinfachen von Brüchen verwendet werden.

**Praxisnutzen**

### 2.1 Vereinfachen von Brüchen

Unter Umständen ist es sinnvoll, während der Berechnung auftretende Brüche zu kürzen. Dies erhöht nicht nur die Lesbarkeit, sondern kann auch für eine effizientere Weiterverarbeitung in Computeralgebrasystemen genutzt werden. So lässt sich mit kleinen, gekürzten Brüchen einfacher weiterrechnen, als mit ungekürzten. Computeralgebrasysteme liefern meist automatisch die gekürzten Brüche als Ergebnis.

**Kürzen**

Es wird nun versucht, den in der Einleitung aufgeführten Bruch zu vereinfachen. Hierzu wird der erweiterte Euklidische Algorithmus<sup>13</sup> genutzt, um den GGT von Zähler und Nenner zu ermitteln.

**GGT berechnen**

Beispiel D: Gesucht ist also der GGT von  $x=2318068701$  und  $y=2436944019$  für den Bruch  $\frac{2318068701}{2436944019}$ . Der Algorithmus liefert die Tabelle:

**Beispiel**

$i$	$q_i$	$r_i$	$s_i$	$t_i$	$x \cdot s_i + y \cdot t_i = r_i$
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		<u>0</u>	-41	39	$2318068701 \cdot (-41) + 2436944019 \cdot 39 = 0$

<sup>13</sup> Der „einfache“ Euklidische Algorithmus würde natürlich auch zum Ziel führen

Das Ergebnis lautet also  $GGT=59437659$ . Der GGT kann zudem als Linearkombination der Eingangsparameter dargestellt werden:  $2318068701 \cdot (20) + 2436944019 \cdot (-19) = 59437659$ . Wird der Zähler und Nenner jeweils durch 59437659 geteilt, so ergibt sich der gekürzte Bruch  $\frac{39}{41}$ .

## 2.2 Rechenoperationen

Eine weitere Anwendung der GGT-Berechnung ist die Vereinfachung der Addition und der Subtraktion sowie der Multiplikation und der Division zweier Brüche<sup>14</sup>. Im Folgenden sollen die Brüche  $\frac{a}{b}$  und  $\frac{c}{d}$  als bereits gekürzt angenommen werden. Weiterhin soll gelten, dass die Brüche ungleichnamig, also die Nenner verschieden sind. **Vereinfachung**

Die Addition wird häufig intuitiv folgendermaßen verwendet: **Addition**

$$\frac{a}{b} + \frac{c}{d} = \frac{a}{b} \cdot \frac{d}{d} + \frac{c}{d} \cdot \frac{b}{b} = \frac{a \cdot d + b \cdot c}{b \cdot d} \quad (6)$$

Die Subtraktion ist analog beschrieben: **Subtraktion**

$$\frac{a}{b} - \frac{c}{d} = \frac{a}{b} \cdot \frac{d}{d} - \frac{c}{d} \cdot \frac{b}{b} = \frac{a \cdot d - b \cdot c}{b \cdot d} \quad (7)$$

Durch Erweitern der Brüche mit dem jeweils anderen Nenner können die Brüche auf einen gemeinsamen Nenner gebracht und zu einem Bruch zusammengefasst werden. Da durch die Multiplikation „große“ Zahlen auftreten, sollte das Ergebnis gekürzt werden. Hierzu wird der GGT von Zähler und Nenner gebildet, bei der Addition  $GGT(a \cdot d + b \cdot c, b \cdot d)$  beziehungsweise  $GGT(a \cdot d - b \cdot c, b \cdot d)$  bei der Subtraktion. Werden nun Zähler und Nenner durch den GGT geteilt, dann ergibt sich automatisch der gekürzte Bruch.

Ein Computer müsste für die Berechnung somit eine Addition beziehungsweise Subtraktion, drei Multiplikationen und eine GGT-Berechnung durchführen. Zum Kürzen sind noch zwei Divisionen erforderlich.

<sup>14</sup> vgl. [Koe2006], S. 79 ff.

Ein alternatives Vorgehen<sup>15</sup> wäre es, die großen Zahlen durch die frühe Multiplikation zu vermeiden und eine GGT-Berechnung vorab zu tätigen. Dafür wird  $g = GGT(b, d)$  berechnet. Dies liefert  $b' = \frac{b}{g}$  und  $d' = \frac{d}{g}$ . Nun gilt

**Alternatives Vorgehen****Addition**

$$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d' + b' \cdot c}{b' \cdot d' \cdot g} \quad (8)$$

für die Addition. Die Subtraktion ist wiederum analog definiert als

**Subtraktion**

$$\frac{a}{b} - \frac{c}{d} = \frac{a \cdot d' - b' \cdot c}{b' \cdot d' \cdot g} \quad (9)$$

Der Ergebnisbruch kann gegebenenfalls noch weiter gekürzt werden, sodass eine zweite GGT-Berechnung durchgeführt werden muss. Vereinfachend kann  $GGT(a \cdot d' + b' \cdot c, g)$  für die Addition beziehungsweise im Fall der Subtraktion  $GGT(a \cdot d' - b' \cdot c, g)$  gerechnet werden, da  $b'$  und  $d'$  garantiert keine Teiler von  $a \cdot d' + b' \cdot c$  beziehungsweise  $a \cdot d' - b' \cdot c$  enthalten. Mit diesem Vorgehen treten kleinere Zahlen auf, was eventuell zu einer effizienteren Verarbeitung führt.

Beispiel E1: Es sollen die Brüche  $\frac{a}{b} = \frac{16}{21}$  und  $\frac{c}{d} = \frac{5}{9}$  addiert werden. Die Berechnung von  $GGT(a \cdot d + b \cdot c, b \cdot d)$  nach der intuitiven Methode liefert  $GGT(16 \cdot 9 + 21 \cdot 5, 21 \cdot 9) = GGT(249, 189) = 3$ . Der Zähler und Nenner von  $\frac{249}{189}$  wird nun durch den GGT geteilt. Das Ergebnis beträgt  $\frac{83}{63}$ .

**Beispiel**

Das optimierte Verfahren würde zunächst  $g = GGT(b, d) = GGT(21, 9) = 3$  berechnen. Dies liefert  $b' = \frac{21}{3} = 7$  und  $d' = \frac{9}{3} = 3$  und somit  $\frac{16 \cdot 3 + 7 \cdot 5}{7 \cdot 3 \cdot 3} = \frac{83}{63}$ . Die abschließende Berechnung  $GGT(a \cdot d' - b' \cdot c, g) = GGT(16 \cdot 3 - 7 \cdot 5, 3) = 1$  ergibt, dass der Bruch nicht weiter gekürzt werden kann. Auch hier lautet das Ergebnis somit  $\frac{83}{63}$ , aber es wurde mit kleineren Zahlen gearbeitet.

Beispiel E2: Für die Subtraktion der Brüche  $\frac{a}{b} = \frac{16}{21}$  und  $\frac{c}{d} = \frac{5}{9}$  würde intuitiv  $GGT(a \cdot d - b \cdot c, b \cdot d)$  berechnet werden. Das Resultat der Berechnung  $GGT(16 \cdot 9 - 21 \cdot 5, 21 \cdot 9) = GGT(39, 189) = 3$  wird als Teiler von Zähler und Nenner des Bruchs  $\frac{39}{189}$  verwendet. Daraus folgt der gekürzte Bruch  $\frac{13}{63}$  als Endergebnis.

**Beispiel**

<sup>15</sup> vgl. [Kap2005], S. 88-89

Mit dem optimierten Verfahren würde  $g = GGT(b, d) = GGT(21, 9) = 3$  zu Beginn berechnet werden. Daraus resultieren  $b' = \frac{21}{3} = 7$  und  $d' = \frac{9}{3} = 3$ , sowie  $\frac{16 \cdot 3 - 7 \cdot 5}{7 \cdot 3 \cdot 3} = \frac{13}{63}$ . Auch in diesem Fall kann der Bruch nicht weiter vereinfacht werden, da Zähler und Nenner teilerfremd sind.

Die Multiplikation zweier Brüche würde intuitiv berechnet werden als:

**Multiplikation**

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d} \quad (10)$$

Für die Division gilt folgende Rechnung:

**Division**

$$\frac{a}{b} : \frac{c}{d} = \frac{a}{b} \cdot \frac{d}{c} = \frac{a \cdot d}{b \cdot c} \quad (11)$$

Im Fall der Multiplikation werden jeweils die Zähler und die Nenner miteinander multipliziert. Bei der Division wird zunächst der Kehrwert des Divisors gebildet, um anschließend ebenfalls die Zähler und Nenner miteinander zu multiplizieren. Da sich auch hier „große“ Zahlen ergeben, sollte das Ergebnis gekürzt werden. Es wird daher der GGT von Zähler und Nenner berechnet, bei der Multiplikation  $GGT(a \cdot c, b \cdot d)$  beziehungsweise  $GGT(a \cdot d, b \cdot c)$  bei der Division. Zähler und Nenner werden nun durch den GGT geteilt. Dies liefert den gekürzten Bruch als Resultat.

Für die Multiplikation beziehungsweise Division sind somit zwei Multiplikationen und eine GGT-Berechnung notwendig. Anschließend sind noch zwei Divisionen erforderlich.

Analog zur Addition beziehungsweise Subtraktion kann auch hier ein alternatives Vorgehen<sup>16</sup> gewählt werden. Ziel ist es ebenfalls große Zahlen zu vermeiden. Für den Fall der Multiplikation lässt sich das Produkt schreiben als

**Alternatives Vorgehen**

**Multiplikation**

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{a}{d} \cdot \frac{c}{b} \quad (12)$$

Dies zeigt, dass ein vorheriges Kürzen der Brüche  $\frac{a}{d}$  und  $\frac{c}{b}$  gegebenenfalls kleinere Zahlen ergibt. Es müssen also zwei GGT-Berechnungen (statt einer) durchgeführt werden, aber jeweils von kleineren Zahlen.

<sup>16</sup> vgl. [Kap2005], S. 88-89

Nachdem der Kehrwert des zweiten Bruchs gebildet wurde, lässt sich das **Division** Produkt bei der Division folgendermaßen schreiben:

$$\frac{a}{b} \cdot \frac{d}{c} = \frac{a}{c} \cdot \frac{d}{b} \quad (13)$$

Bei der Division sollten daher die Brüche  $\frac{a}{c}$  und  $\frac{d}{b}$  vorab gekürzt werden.

Beispiel E3: Jetzt wird die Multiplikation der Brüche  $\frac{a}{b} = \frac{16}{21}$  und  $\frac{c}{d} = \frac{5}{9}$  **Beispiel** durchgeführt. Intuitiv würde in diesem Fall  $GGT(a \cdot c, b \cdot d)$  berechnet werden. Es ergibt sich  $GGT(16 \cdot 5, 21 \cdot 9) = GGT(80, 189) = 1$ . Da Zähler und Nenner teilerfremd sind, lässt sich der Bruch  $\frac{80}{189}$  nicht weiter kürzen.

Die optimierte Methode würde zunächst  $GGT(a, d) = GGT(16, 9) = 1$  und  $GGT(c, b) = GGT(5, 21) = 1$  berechnen. Die beiden Brüche lassen sich somit nicht weiter kürzen. Daher bildet das Produkt  $\frac{a}{d} \cdot \frac{c}{b} = \frac{16}{9} \cdot \frac{5}{21} = \frac{80}{189}$  das Endergebnis.

Beispiel E4: Als erster Schritt der Division von  $\frac{a}{b} = \frac{16}{21}$  und  $\frac{c}{d} = \frac{5}{9}$  muss der **Beispiel** Kehrwert des zweiten Bruchs gebildet werden. Die Berechnung von  $GGT(a \cdot d, b \cdot c)$  nach dem intuitiven Verfahren liefert für dieses Beispiel  $GGT(16 \cdot 9, 21 \cdot 5) = GGT(144, 105) = 3$ . Der Zähler und Nenner von  $\frac{144}{105}$  wird nun durch den GGT geteilt. Dies führt zum gekürzten Bruch  $\frac{48}{35}$ .

Nach der optimierten Methode würde zunächst  $GGT(a, c) = GGT(16, 5) = 1$  und  $GGT(d, b) = GGT(9, 21) = 3$  berechnet werden. Das Kürzen des zweiten Bruchs liefert  $\frac{9}{21} = \frac{3}{7}$ . Der erste Bruch kann nicht vereinfacht werden. Danach wird das Produkt  $\frac{a}{c} \cdot \frac{d}{b} = \frac{16}{5} \cdot \frac{3}{7} = \frac{48}{35}$  ermittelt. Auch hier ergibt sich selbstverständlich das gleiche Ergebnis, aber es wurden Operationen mit kleineren Zahlen durchgeführt.

### 3 Algorithmen zur Faktorisierung

Ziel einer Faktorisierung ist es, eine Zahl in ihre Primfaktoren<sup>17</sup> zu zerlegen. **Motivation**  
 Jede Zahl  $N > 1$  lässt sich durch die Primzahlen  $p_i$  und den Exponenten  $e_i$  als  $N = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_s^{e_s}$  *eindeutig* darstellen. Eindeutig<sup>18</sup> meint, dass zwei Primfaktorzerlegungen sich nur in der Reihenfolge der Werte unterscheiden dürfen. Die Anzahl und natürlich die Werte selber müssen übereinstimmen.

Dieses ist mit den bis heute bekannten Verfahren eine sehr rechenaufwendige **Exponentielle Laufzeit**  
 Aufgabe, die in der Praxis nicht in polynomialer Zeit durchführbar ist. Es gibt derzeit mehrere effiziente Verfahren, die jedoch alle mit exponentieller Laufzeit arbeiten. Diese „Schwäche“ wird von verschiedenen Verschlüsselungsverfahren genutzt (siehe Anhang).

In den meisten Fällen werden beim Faktorisieren mehrere Verfahren **Ablaufplan**  
 hintereinander angewendet, da *ein* Verfahren nicht alle Faktoren mit *einem* Durchlauf liefert. Dieser allgemeine Ablauf ist in Abbildung 1 dargestellt. Der Faktorisierungsalgorithmus kann je Iteration gewechselt werden, da es verschieden effiziente Verfahren gibt.

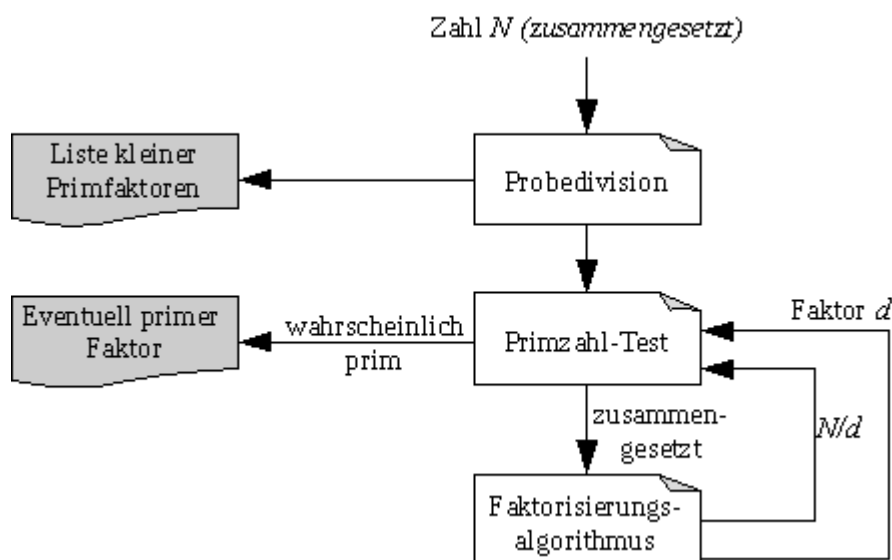


Abbildung 1: Faktorisierung von ganzen Zahlen

<sup>17</sup> vgl. [Iwa2007], S. 10

<sup>18</sup> vgl. [Koe2006], S. 73 ff.

Ziel ist es die Zahl  $N$  zu zerlegen. Um keine unnützen Berechnungen zu tätigen, kann bereits vorab getestet werden, ob die Zahl zusammengesetzt ist. Aufgrund von Effizienzbetrachtungen wird mit einer Probedivision (3.1) begonnen. So lassen sich bereits alle „kleinen“ Primfaktoren entfernen. Daran schließt sich eine Prüfung an, ob die verbleibende Zahl zusammengesetzt ist (3.2). Handelt es sich um eine zusammengesetzte Zahl, so lassen sich verschiedene Faktorisierungsalgorithmen (3.3) anwenden. Die Zahl wird dann in den nichttrivialen Faktor  $d$  und  $N/d$  zerlegt. Danach wird der Ablauf für beide Teile *einzel*n wiederholt – bis keine weitere Zerlegung mehr möglich ist.

### 3.1 Probedivision

Die Probedivision ist eine einfache Methode zur Faktorisierung einer Zahl  $N$ . Sie findet aufgrund der exponentiellen Laufzeit meist nur Anwendung bei der Ermittlung von kleinen Primfaktoren<sup>19</sup>. Je nach Umsetzung wird sich meist auf Zahlen  $\leq 10^6$  beschränkt.

**Faktorisieren  
durch  
Probieren**

Für die Durchführung des Verfahrens wird eine Liste von Divisoren benötigt. Für mindestens ein Element muss die Beziehung  $d_k \geq \sqrt{N}$  gelten, alle anderen Elemente sind  $\leq \sqrt{N}$ . Hierzu gibt es ganz unterschiedliche Ansätze<sup>20</sup>: eine Möglichkeit wäre es, die zu prüfenden Primzahlen in einer Tabelle fest zu speichern. Dies würde jedoch relativ viel Speicherplatz in Anspruch nehmen. Um daher auf eine Speicherung zu verzichten, ließen sich die Primzahlen (bis zu einer Grenze) auch berechnen. Hier könnte das bekannte Sieb des Eratosthenes<sup>21</sup> verwendet werden, welches alle Primzahlen zwischen 2 und einer frei wählbaren Grenze liefert. Zur Vermeidung des Rechenaufwands, verursacht durch das „Sieben“, könnten auch alle positiven Zahlen der Form  $6 \cdot t \pm 1$  mit  $t \geq 1$  (sowie 2 und 3) als Divisoren genutzt werden. Die Gleichung würde alle Primzahlen, aber auch zusammengesetzte Zahlen wie 25, 35, 49 ... liefern. Dies bedeutet zwar auf der einen Seite überflüssige Divisionen, aber auf der anderen Seite kann auf das „teure“ Sieben verzichtet werden. Durch „Implementationstricks“ wie das Überspringen von Vielfachen der Zahl 7 lässt sich die Liste verkleinern.

**Verschiedene  
Ansätze**

<sup>19</sup> vgl. [vGG2003], S. 533ff.

<sup>20</sup> vgl. [Rie1994], S. 141ff.

<sup>21</sup> vgl. [Iwa2007], S. 8

Die Durchführung der Probedivision verläuft wie folgt<sup>22</sup>:

Es soll die positive Zahl  $N$  in ihre Primfaktoren zerlegt werden. Außerdem muss eine Liste von Divisoren ( $d_1 \dots d_i$ ) verfügbar sein, die ausgehend von der obigen Gegenüberstellung nur Primzahlen oder auch weitere Zahlen enthält. Im Codefragment steht  $q$  für den Quotienten,  $r$  für den Restwert und  $p_t$  für die gefundenen Faktoren. Die Variable  $n$  beinhaltet den verbleibenden, noch nicht zerlegten Teil der Zahl  $N$ . Zu jeder Zeit gilt daher  $n = N / (p_1 \dots p_t)$ .

---

```

1  public Probedivision( N∈N , d1 .. di ∈N ) {
2    t = 0; k = 0; n = N;
3    // es gilt: n=N/(p1*...*pt) &&
4    // n hat keine Primfaktoren < dk
5    while (n != 1) {
6      q = n DIV dk; r = n MOD dk;
7      if (r = 0) { // Faktor gefunden
8        t++; n = q;
9        pt = dk; // Faktor speichern
10     } else {
11       if (q > dk) {
12         k++; // nächsten Faktor probieren
13       } else {
14         t++;
15         pt = n; // n ist prim
16         return; // Ende des Algorithmus
17       }
18     }
19   }
20 }

```

---

**Algorithmus**

*Listing 4: Probedivision*

Beispiel F1: Es wird nun der obige Algorithmus auf die Zahl  $N=4242$  angewendet. Die Divisorenliste ( $d_1 \dots d_i$ ) enthält die Zahlen der Form  $6 \cdot t \pm 1$  mit  $t \geq 1$  sowie 2 und 3, also:

**Beispiel**

$\{2,3,5,7,11,13,17,19,23,25,29,31,35, \dots, 67\}$ .

Aufgrund  $d_k \leq \sqrt{4242}$  brauchen nur Elemente bis  $\leq 65$  enthalten sein. Zusätzlich muss ein Element  $\geq 65$  in der Liste vorkommen, hier 67.

<sup>22</sup> vgl. [Knu1998], S. 380ff.

Es entsteht folgende Tabelle:

$t$	$k$	$d_k$	$n$	$q$	$r$	$p_t$
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	$\Rightarrow p_4=101$

Der Algorithmus terminiert, da nun  $q \leq d_k \Leftrightarrow 9 \leq 11$  gilt. Folglich ist  $n=101$  prim und wird der Liste der Faktoren hinzugefügt. Es ergibt sich die Zerlegung  $4242=2 \cdot 3 \cdot 7 \cdot 101$ .

In diesem Beispiel wurden 8 Schritte für die Berechnung des Endergebnisses benötigt. Für die Zahl  $N=4243$ , die eine Primzahl ist, hätte das Verfahren mindestens 19 Schritte<sup>23</sup> benötigt, um festzustellen, dass es sich um eine Primzahl handelt.

Mit einem „Trick“<sup>24</sup> lässt sich prüfen, ob eine Probedivision bis zu einer (beliebig festgelegten) oberen Grenze überhaupt Teiler finden würde. Angenommen es gäbe bis zu dieser Grenze keine „kleinen“ Teiler, so ließe sich der Rechenaufwand der Probedivision sparen – und gleich mit einem anderen Faktorisierungsverfahren beginnen.

**Test auf  
Existenz von  
kleinen Teilern**

Für die Prüfung ist das Produkt der Divisoren (je nach Ansatz Primzahlen und weitere Zahlen zum Beispiel 25, 35 oder 49, wie bereits zu Beginn diskutiert) zu bilden, die kleiner als die festgelegte Grenze sind. Bei gleich bleibender Grenze kann das Produkt  $P$  einmalig berechnet und dauerhaft im Code als Konstante gespeichert werden. Nun ließe sich durch die Berechnung von  $GGT(P, N)$

**GGT des  
Produktes  
und der Zahl**

<sup>23</sup> je nach Umfang der Divisorenliste

<sup>24</sup> vgl. [Kap2005], S. 168

vergleichsweise schnell feststellen, ob die Zahl  $N$  mindestens einen Teiler aus der Divisorenliste besitzt. Gilt  $GGT > 1$ , so existiert mindestens ein Teiler und die Probedivision wäre durchzuführen. Bei  $GGT = 1$  sind die Zahlen teilerfremd und die Probedivision würde keine „kleinen“ Teiler finden, sodass sie übersprungen werden kann.

Beispiel F2: Im Folgenden werden die Teiler bis zur oberen Grenze 71 betrachtet. Das berechnete Produkt kann als Konstante gespeichert werden. **Beispiel**

$$P = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71$$

$$P = 557940830126698960967415390$$

Es soll geprüft werden, ob die Zahl  $N = 731771 = 53 \cdot 13807$  kleine Teiler enthält. Die Berechnung von  $GGT(P, N)$  liefert als Ergebnis  $GGT(557940830126698960967415390, 731771) = 53$ . Da  $GGT > 1$ , enthält die Zahl sicher kleine Teiler bis 71 und eine Probedivision ist durchzuführen. Der kleinste Faktor ist in diesem Beispiel 53.

Beispiel F3: Auch für dieses Beispiel wird die zuvor berechnete Konstante  $P$  verwendet. Für die Zahl  $N = 544909 = 163 \cdot 3343$  ist festzustellen, ob sie kleine Teiler enthält. Das Ergebnis von  $GGT(P, N)$  beträgt in diesem Fall  $GGT(557940830126698960967415390, 544909) = 1$ . **Beispiel**

Die Probedivision kann übersprungen werden, da  $GGT = 1$  und somit keine kleinen Teiler bis 71 enthalten sind. Der kleinste Faktor ist hier 163.

Das Verfahren bietet sich für Zahlen  $\leq 10^6$  an, da nur Divisoren bis einschließlich  $\sqrt{(10^6)} = 1000$  zu verwalten sind. Doch auch für größere Zahlen lässt sich dieses Verfahren sinnvoll verwenden, indem eine obere Grenze<sup>25</sup> für die Prüfung integriert und nicht die gesamte Zahl mit diesem Verfahren zerlegt<sup>26</sup> wird. So lassen sich vorab bereits die „kleinen Primfaktoren“ ermitteln. Anschließend werden effizientere Verfahren genutzt, um die restliche Zahl weiter zu zerlegen, falls es sich nicht zufällig bereits um eine Primzahl handelt. **Bewertung**

Die Laufzeit<sup>27</sup> des Verfahrens beträgt  $O(2^{n/2})$  für eine Zahl  $N$  mit Länge  $n$ . **Laufzeit**

<sup>25</sup> je nach Implementierung; der Algorithmus bricht ab, wenn  $d_k$  diese Grenze überschreitet

<sup>26</sup> vgl. [Coh1993], S. 413

<sup>27</sup> vgl. [vGG2003], S. 531

## 3.2 Primzahl-Test

Wie bereits bei der Probedivision gesehen, arbeiten Faktorisierungsalgorithmen vergleichsweise langsam. Daher ist es sinnvoll vorab zu testen, ob eine Zahl zusammengesetzt ist. Die Berechnung erfolgt sehr schnell. Macht das Verfahren die Aussage „ist zusammengesetzt“, dann ist dieses Ergebnis sicher richtig. Die Aussage „ist Primzahl“ ist jedoch mit geringer Wahrscheinlichkeit falsch. Zahlen, die fälschlicherweise als Primzahlen identifiziert werden, heißen *strenge Pseudoprimzahlen*. Zudem sagen die Verfahren nichts über mögliche Teiler aus<sup>28</sup>. Verbreitete Verfahren sind hierbei zum Beispiel der Fermat-Test oder der Rabin-Miller-Test. Aufgrund der Themenabgrenzung sei an dieser Stelle auf das Kapitel 4.6 des Buches „Computeralgebra“ von Wolfram Koepf verwiesen.

**Zusammen-  
gesetzte  
Zahlen  
erkennen**

**Rabin-Miller-  
Test**

## 3.3 Moderne Faktorisierungsalgorithmen

### 3.3.1 Übersicht

Die Faktorisierung einer zusammengesetzten, ganzen Zahl  $N$ , ist wie gesehen sehr komplex und rechenaufwendig. Bis heute ist kein praxistaugliches Verfahren bekannt, mit dem die Zerlegung in polynomialer Zeit durchführbar ist. Die derzeit bekannten Verfahren arbeiten alle mit exponentieller Laufzeit. Mit dem Shor-Algorithmus (siehe Kapitel 3.3.2) gibt es jedoch einen theoretischen Ansatz, der die Zerlegung in polynomialer Zeit erreichen *könnte*. Dies hätte einen Einfluss auf die Kryptografie (siehe Anhang), da das RSA-Verfahren von der Annahme ausgeht, dass *keine* Zerlegung in polynomialer Zeit möglich ist.

**Rechen-  
aufwand**

Es werden zwei Klassen von Faktorisierungsalgorithmen unterschieden: *allgemeine* Verfahren erwarten keine bestimmte Form einer Zahl, im Gegensatz zu den *speziellen* Verfahren. Diese erfordern besondere Annahmen<sup>29</sup> über die zu zerlegende Zahl, wie zum Beispiel über die Form. Dafür arbeiten spezielle Verfahren unter Umständen effizienter. Für detaillierte Beschreibungen von Faktorisierungsverfahren sei auf das Kapitel 5 des Buches „Computeralgebra“ von Michael Kaplan verwiesen.

**Allgemeine  
und spezielle  
Verfahren**

<sup>28</sup> vgl. [Koe2006], S. 107 ff.

<sup>29</sup> vgl. [Rie1994], S. 173 ff.

Mithilfe des derzeit schnellsten Verfahrens, dem sogenannten Zahlkörpersieb, gelang es der Universität Bonn im Mai 2005 den Weltrekord<sup>30</sup> im Faktorisieren aufzustellen. Die Forscher begannen Ende 2003, die 200-stellige Zahl RSA-200

**Weltrekord**

$$\begin{aligned} &2799783391122132787082946763872260162107044678695 \\ &5428537560009929326128400107609345671052955360856 \\ &0618223519109513657886371059544820065767750985805 \\ &57613579098734950144178863178946295187237869221823983 \end{aligned} \quad (14)$$

zu zerlegen und nutzen dafür ein Cluster von 80 2,2 GHz-Pentium-Rechnern. Der größte Rechenschritt des Verfahrens dauerte dabei 3 Monate. Für die gesamte Zerlegung hätte ein einzelner 2,2 GHz-Rechner 55 Jahre gebraucht. Die Forscher errechneten zwei Faktoren:

$$\begin{aligned} &35324619344027701212726049781984643686711974001976250 \\ &23649303468776121253679423200058547956528088349 \end{aligned} \quad (15)$$

und

$$\begin{aligned} &79258699544783330333470858414800596877379758573642 \\ &19960734330341455767872818152135381409304740185467 \end{aligned} \quad (16)$$

### 3.3.2 Shor-Algorithmus

Der 1994 entwickelte Shor-Algorithmus<sup>31</sup> ist ein theoretischer Ansatz, mit dem ganze Zahlen in polynomialer Laufzeit zerlegt werden können. Das Verfahren arbeitet auf einem Quantencomputer mit Quanten-Bits. Dies sind Atome mit besonderen Quantenzuständen. Aufgrund ihrer Leistungsfähigkeit können die Quanten-Bits gleichzeitig als Prozessor und Speicher arbeiten. 2001 konnte IBM mit einigem Aufwand die Zahl 15 erfolgreich zerlegen und nutzte dafür einen Quantencomputer mit 7 Quanten-Bits.

**Polynomiale Laufzeit**

Der Shor-Algorithmus benötigt für die Zerlegung einer ganzen Zahl  $N$  auf einem Quantencomputer die Laufzeit<sup>32</sup>  $O((\log N)^2(\log \log N)(\log \log \log N))$  und zusätzlich  $O(\log N)$  für die Nachverarbeitung auf einem normalen Computer zur Ausgabe der Faktoren. Da Quantencomputer technisch sehr aufwendig sind, ist das Verfahren (noch) nicht praktisch anwendbar.

**Laufzeit**

<sup>30</sup> vgl. [RSA2005]

<sup>31</sup> vgl. [IBM2001]

<sup>32</sup> vgl. [Sho1996]

## 4 Fazit und Ausblick

Obwohl die ersten Konzepte, wie der Euklidische Algorithmus bereits seit 300 v. Chr.<sup>33</sup> bekannt sind, entwickelten sich nur geringe Neuerungen, wie der erweiterte Euklidische Algorithmus (499 n. Chr.<sup>34</sup>). Dennoch besteht auch heute noch ein großes Interesse an ihnen.

Sie dienen zur schnellen GGT-Berechnung, zur Darstellung des GGT als Linearkombination und zum Vereinfachen von Brüchen. Oftmals werden sie auch für Beweise anderer Algorithmen oder als Hilfssätze verwendet, wie im Chinesischen Restsatz<sup>35</sup>.

Die Faktorisierung hingegen ist rückblickend betrachtet eine sehr dynamische Disziplin<sup>36</sup>: erste Algorithmen zur „schnellen“ Zerlegung von Zahlen bis 40 Stellen gab es zwar erst ab 1970, doch nur 30 Jahre später existieren Verfahren für 200-stellige Zahlen. Dies zeigt, dass die Entwicklung noch nicht zu Ende ist, sondern dieses Forschungsgebiet auch zukünftig für weitere Rekorde sorgen wird.

Das wiederum könnte Probleme für Verschlüsselungsverfahren wie das RSA-Verfahren bedeuten, da diese davon ausgehen, dass eine Zahl nicht in polynomialer Zeit zerlegt werden kann. Mit bisherigen Methoden ist dieses (noch) nicht möglich.

Die Entwicklungen im Bereich der Faktorisierung zeigen, wie praxisnah und bedeutsam dieses Gebiet der Computeralgebra ist. Zukünftig werden weitere Verfahren und Techniken zu erwarten sein.

---

33 vgl. [Knu1998], S. 335

34 vgl. [Knu1998], S. 343

35 vgl. [Koe2006], S. 96 ff.

36 vgl. [vGG2003], S. 531-532

## Anhang: Computeralgebra in der Praxis: Kryptografie

E-Mails, E-Banking, Filetransfer – alle diese Techniken sind für die Anwender besonders praktisch. Da sie jedoch nicht in einem geschlossenen, sondern in einem offenen System (Internet) ablaufen, sind sie dem Zugriff Dritter ständig ausgesetzt. Wer nicht möchte, dass seine Daten mitgelesen werden, ist auf eine sichere Verschlüsselung angewiesen.

**Offenes System**

Dieser Anhang zeigt die Beziehungen der Zahlentheorie zur modernen Kryptografie auf und vermittelt die Praxisrelevanz der mathematischen Konzepte in der heutigen Zeit. Für detaillierte (mathematische) Beschreibungen der Verfahren sei an dieser Stelle auf das Buch „Moderne Verfahren der Kryptographie“ von Albrecht Beutelspacher, Jörg Schwenk und Klaus-Dieter Wolfenstetter verwiesen, da diese nicht Thema der Ausarbeitung sind.

**Bezug zur Zahlentheorie**

Es wird gezeigt, dass der erweiterte euklidische Algorithmus eine Rolle bei der Schlüsselerzeugung spielt. Die Schwierigkeit der Faktorisierung dient hingegen als „Garant“ für die Sicherheit der Verschlüsselung.

### A Motivation der Kryptografie

Es lassen sich bestimmte Hauptaufgaben<sup>37</sup> der Kryptografie identifizieren: es muss gewährleistet sein, dass sich die Kommunikationspartner geheim unterhalten können (Geheimhaltung). Hier werden zwei Arten von Verschlüsselungen unterschieden: die symmetrischen Verfahren, wo beide Partner den gleichen (geheimen) Schlüssel nutzen. Bei asymmetrischen Verfahren (auch Public-Key-Verfahren genannt) verfügt eine Person im Gegensatz dazu über zwei unterschiedliche Schlüssel. Der private (geheime) Schlüssel ist nur dem Empfänger selber bekannt. Seinen zweiten (öffentlichen) Schlüssel kann jedem anderen bekannt sein – dieser dient zum Verschlüsseln der Nachricht an den Empfänger. Die Nachricht kann nur mit dem privaten Schlüssel entschlüsselt werden. Symmetrische Verfahren haben den Vorteil, dass sie meist schneller arbeiten, als asymmetrische, jedoch müssen viel mehr Schlüssel gespeichert werden. Prinzipiell müssten alle Partner jeweils ihre geheimen Schlüssel übertragen und speichern.

**Geheimhaltung**

<sup>37</sup> vgl. [BSW2006], S. 1 ff.

Bei der zweiten Aufgabe, der Authentikation, gilt es für den Sender der Nachricht nachzuweisen, dass er wirklich die richtige Person ist und nicht ein Dritter, der in seinem Namen etwas versendet. Dieses lösen sogenannte Signaturverfahren, die hier nicht näher besprochen werden.

**Authentikation**

Außerdem befasst sich die Kryptografie mit der Frage nach der Anonymität der kommunizierenden Personen. Ziel ist es, die eigene Identität an so wenigen Stellen wie möglich bekannt zu geben, am besten natürlich bei keiner Transaktion. Auch dieses wird im Folgenden nicht näher vertieft.

**Anonymität**

## **B RSA**

### ***B.1 Funktionsweise***

Für den Ablauf der Verschlüsselung ist Wissen aus dem Bereich der modularen Arithmetik notwendig, wie sie in Kapitel 4 des Buches „Computeralgebra“ von Wolfram Koepf beschrieben ist. Daher wird hier nur auf das grundsätzliche Verfahren eingegangen, um zu zeigen, wie die mathematischen Konzepte zur Sicherheit des Verfahrens beitragen.

Ron Rivest, Adi Shamir und Len Adleman wollten zeigen, dass Public-Key-Verfahren nicht funktionieren würden. Hierbei jedoch entwickelten sie den nach ihnen benannten RSA-Algorithmus zur Verschlüsselung. RSA zählt zu den bekanntesten Public-Key-Verfahren (asymmetrisch) und wird sehr häufig eingesetzt. Die Funktionsweise des Verfahrens ist wie folgt:

**Public-Key-Verfahren**

Zu Beginn erzeugt sich eine Person ein Schlüsselpaar<sup>38</sup> (privat und öffentlich). Hierzu werden zwei gleichgroße Primzahlen (mindestens 512-Bit-Zahlen<sup>39</sup>) benötigt. Diese ergeben das sogenannte RSA-Modul  $n = p \cdot q$  sowie die eulersche Funktion<sup>40</sup>  $\varphi(n) = (p-1) \cdot (q-1)$ .

**Schlüssel-erzeugung**

Anschließend wird eine beliebige Zahl  $e \in \mathbb{N}$  (encryption) gewählt, die teilerfremd zu  $\varphi(n)$  ist. Der öffentliche Schlüssel wird vom Paar  $(e, n)$  gebildet und zum Verschlüsseln von Nachrichten an den Empfänger genutzt. Der private Schlüssel  $d$  (decryption) lässt sich nun mithilfe des bereits

<sup>38</sup> vgl. [Buc2001], S. 115 ff.

<sup>39</sup>  $512 \cdot \log(2) : \log(10) \approx 155$  : 512 Bit entsprechen ungefähr 155 Dezimalstellen

<sup>40</sup> Die eulersche Funktion für eine Zahl  $n \in \mathbb{N}$  berechnet die Anzahl positiver natürlicher Zahlen, welche  $\leq n$  und teilerfremd zu  $n$  sind.

vorgestellten erweiterten euklidischen Algorithmus berechnen. Es muss die Gleichung  $e \cdot d \bmod \varphi(n) = 1$  gelten.

Beispiel 1: Als Primzahlen werden  $p=41$  und  $q=19$  gewählt. Daraus ergibt sich  $n=p \cdot q=41 \cdot 19=779$  und  $\varphi(n)=(p-1) \cdot (q-1)=40 \cdot 18=720$ .

Wird  $e=7$  gesetzt, dann lässt sich  $d$  wie folgt berechnen:

$i$	$q_i$	$r_i$	$d_i$ ( $s_i$ )	$k_i$ ( $t_i$ )	$e \cdot d_i + \varphi(n) \cdot k_i = r_i$ ( $x \cdot s_i + y \cdot t_i = r_i$ )
0		7	1	0	$7 \cdot 1 + 720 \cdot 0 = 7$
1	0	720	0	1	$7 \cdot 0 + 720 \cdot 1 = 720$
2	102	7	1	0	$7 \cdot 1 + 720 \cdot 0 = 7$
3	1	6	-102	1	$7 \cdot 1 + 720 \cdot (-102) = 6$
4	6	1	103	-1	$7 \cdot 103 + 720 \cdot (-1) = 1$
5		0	-720	7	$7 \cdot (-720) + 720 \cdot 7 = 0$

Der vorletzten Zeile wird  $d=103$  sowie die Linearkombination des GGT  $7 \cdot 103 + 720 \cdot (-1) = 1$  entnommen.

Die Zahlen  $p, q, \varphi(n)$  sollten aus Sicherheitsgründen nun gelöscht werden, da diese von einem Dritten zur Berechnung des geheimen Schlüssels  $d$  benutzt werden könnten.

Betrachtet wird nun, wie die Verschlüsselung<sup>41</sup> eines Nachrichtentextes im Allgemeinen abläuft, ohne auf die mathematischen Konzepte einzugehen, da dies nicht Gegenstand der Ausarbeitung ist. Vereinfachend wird angenommen, dass der zu verschlüsselnde Text  $m$  eine ganze Zahl ( $0 \leq m < n$ ) ist.

**Verschlüsselung**

Es werden hier zur Vereinfachung nur einfache Nachrichten betrachtet, da größere Nachrichten die Bildung von Blöcken erfordern würden.

Der codierte Text  $c$  ergibt sich aus dem öffentlichen Schlüssel ( $e, n$ ) und der Gleichung  $c = m^e \bmod n$ .

<sup>41</sup> vgl. [Buc2001], S. 116 ff.

Im Beispiel ist der öffentliche Schlüssel vom Partner B ( 7 , 779 ) und es soll die wichtige Zahl 42 verschlüsselt werden, damit nur B sie lesen kann. Die Berechnung ergibt  $c = m^e \bmod n = 42^7 \bmod 779 = 329$  .

Partner B erhält nun die verschlüsselte Nachricht und möchte den Inhalt lesen. Dafür wird die Entschlüsselungsfunktion  $m = c^d \bmod n$  genutzt, um den ursprünglichen Inhalt wieder zu berechnen. Im Beispiel ergibt sich  $m = c^d \bmod n = 329^{103} \bmod 779 = 42$  . Nun hat der Empfänger die wichtige Zahl erhalten, ohne dass ein Dritter mit der Nachricht hätte etwas anfangen können.

**Entschlüsselung**

### **B.2 (Un)Sicherheit**

Die Sicherheit des RSA Verfahrens basiert darauf, dass die Faktorisierung einer ganzen Zahl nicht in polynomialer Laufzeit durchführbar ist. Mit den bekannten Faktorisierungsalgorithmen ist bestenfalls eine exponentielle Laufzeit zu erreichen. Jedoch gibt es mit dem Shor-Algorithmus (siehe Kapitel 3.3.2) zumindest theoretische Ansätze, die Berechnung drastisch zu beschleunigen.

**Faktorisierung**

Doch auch die modernen Computer werden immer besser und können mittlerweile immer größere Zahlen faktorisieren. Folglich sind derzeit 1024 Bit<sup>42</sup> Schlüssel mindestens erforderlich.

Die Sicherheit<sup>43</sup> besteht nun darin, dass es (fast) *nicht* möglich ist, den privaten Schlüssel aus dem öffentlichen Schlüssel zu berechnen. Das Gegenteil würde dazu führen, dass das RSA-Verfahren wirkungslos wäre.

**Berechenbarkeit**

Es lässt sich zeigen, dass die Berechnung des privaten aus dem öffentlichen Schlüssel genauso schwer ist, wie die Faktorisierung von der Zahl  $n$  in  $n = p \cdot q$  (die ja bekannt gemacht wird).

Dies wird auch als Einwegfunktion bezeichnet, da die eine Richtung (Multiplikation) sehr einfach realisierbar ist, wohingegen die andere Richtung (fast) unmöglich ist (Zerlegung).

**Einwegfunktion**

<sup>42</sup>  $1024 \cdot \log(2) : \log(10) \approx 309$  : 1024 Bit entsprechen ungefähr 309 Dezimalstellen

<sup>43</sup> vgl. [Buc2001], S. 119 ff.

Ein offenes Problem ist, ob überhaupt der geheime Schlüssel benötigt wird beziehungsweise die Zahl  $n$  zerlegt werden muss, um aus dem verschlüsselten Text den Ausgangstext zu berechnen<sup>44</sup>. **Offenes Problem**

Das RSA-Verfahren beruht also auf der Schwierigkeit der Faktorisierung<sup>45</sup>. **Zukunft**  
Doch wenn sich dieses Problem in Zukunft als leicht lösbar herausstellen sollte, müsste dieser Sicherungsmechanismus durch einen besseren ausgetauscht werden. Mit derzeitiger Technik ist dies ein sehr schwieriges Problem.

---

44 vgl. [Buc2001], S. 121

45 vgl. [Buc2001], S. 114 ff.

## Literaturverzeichnis

- [BSW2006] Beutelspacher, Albrecht/ Schwenk, Jörg/ Wolfenstetter, Klaus-Dieter: Moderne Verfahren der Kryptographie - Von RSA zu Zero-Knowledge, 6., verb. Aufl., Wiesbaden: Vieweg, 2006
- [Buc2001] Buchmann, Johannes: Einführung in die Kryptographie, 2., erw. Aufl., Berlin, Heidelberg, New York: Springer, 2001
- [Coh1993] Cohen, Henri: A Course in Computational Algebraic Number Theory - Graduate Texts in Mathematics 138, Berlin, Heidelberg, New York: Springer, 1993
- [IBM2001] IBM Research Division, IBM's Test-Tube Quantum Computer Makes History - First Demonstration Of Shor's Historic Factoring Algorithm: ScienceDaily, Internet <http://www.sciencedaily.com/releases/2001/12/011220081620.htm> Stand 2001-12-20, Abruf 2007-11-09
- [Iwa2007] Iwanowski, Sebastian: Vorlesungsfolien 'Diskrete Mathematik' - Kapitel 4, Wedel: FH Wedel, 2007
- [Kap2005] Kaplan, Michael: Computeralgebra, Berlin, Heidelberg, New York: Springer, 2005
- [Knu1998] Knuth, Donald E.: The Art of Computer Programming - Vol. 2 Seminumerical Algorithms, 3. Aufl., Boston, San Francisco, New York: Addison Wesley, 1998
- [Koe2006] Koepf, Wolfram: Computeralgebra - Eine algorithmisch orientierte Einführung, Berlin, Heidelberg, New York: Springer, 2006
- [Rie1994] Riesel, Hans: Prime Numbers and Computer Methods for Factorization - Progress in mathematics Vol. 126, 2. Aufl., Boston, Basel, Stuttgart: Birkhäuser, 1994
- [RSA2005] The RSA Factoring Challenge - RSA-200 is factored: RSA Security, Internet <http://www.rsa.com/rsalabs/node.asp?id=2879> Stand 2005-05-10, Abruf 2007-11-20
- [Sho1996] Shor, Peter: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, Internet <http://www.arxiv.org/abs/quant-ph/9508027> Stand 1996-01-25, Abruf 2007-11-20
- [vGG2003] von zur Gathen, Joachim/ Gerhard, Jürgen: Modern Computer Algebra, 2. Aufl., Cambridge: Cambridge University Press, 2003