Springer-Lehrbuch

Einführung in die Kryptographie

Bearbeitet von
Johannes Buchmann

5. Aufl. 2010. Taschenbuch. xxiv, 280 S. Paperback
ISBN 978 3 642 11185 3
Format (B x L): 15,5 x 23,5 cm
Gewicht: 462 g

2. Ganze Zahlen

Ganze Zahlen spielen eine fundamentale Rolle in der Kryptographie. In diesem Kapitel stellen wir grundlegende Eigenschaften der ganzen Zahlen zusammen und beschreiben fundamentale Algorithmen.

2.1 Grundlagen

Wir schreiben, wie üblich, $\mathbb{N} = \{1, 2, 3, 4, 5, \ldots\}$ für die *natürlichen Zahlen* und $\mathbb{Z} = \{0, \pm 1, \pm 2, \pm 3, \ldots\}$ für die *ganzen Zahlen*. Die rationalen Zahlen werden mit $\mathbb{Q}$ bezeichnet und die reellen Zahlen mit $\mathbb{R}$.

Es gilt $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$. Reelle Zahlen (also auch natürliche, ganze und rationale Zahlen) kann man addieren und multiplizieren. Das wird als bekannt vorausgesetzt.

Außerdem werden folgende grundlegende Regeln benutzt:

Wenn das Produkt von zwei reellen Zahlen Null ist, dann ist wenigstens ein Faktor Null. Es kann also nicht sein, dass beide Faktoren von Null verschieden sind, aber das Produkt Null ist. Wir werden später sehen, dass es Zahlbereiche gibt, in denen das nicht gilt.

Reelle Zahlen kann man vergleichen. Z.B. ist $\sqrt{2}$ kleiner als 2, aber größer als 1. Wenn eine reelle Zahl α kleiner als eine andere reelle Zahl β ist, schreiben wir $\alpha < \beta$. Wenn α kleiner als β oder α gleich β ist, schreiben wir $\alpha \leq \beta$. Wenn α größer als β ist, schreiben wir $\alpha > \beta$. Wenn α größer als β oder α gleich β ist, schreiben wir $\alpha \geq \beta$. Ist γ eine weitere reelle Zahl, dann folgt aus $\alpha < \beta$ auch $\alpha + \gamma < \beta + \gamma$. Entsprechendes gilt für $\leq, >$ und $\geq$. Wenn $0 < \alpha$ und $0 < \beta$, dann folgt $0 < \alpha\beta$.

Eine Menge M von reellen Zahlen heißt *nach unten beschränkt*, wenn es eine reelle Zahl γ gibt, so dass alle Elemente von M größer als γ sind. Man sagt dann auch, dass M nach unten durch γ beschränkt ist. Die Menge der natürlichen Zahlen ist z.B. nach unten durch 0 beschränkt. Die Menge der geraden ganzen Zahlen ist aber nicht nach unten beschränkt. Eine wichtige Eigenschaft der ganzen Zahlen ist, dass jede nach unten beschränkte Menge ganzer Zahlen ein kleinstes Element besitzt. Z.B. ist die kleinste natürliche Zahl 1. Entsprechend definiert man nach oben beschränkte Mengen reeller Zahlen. Jede nach oben beschränkte Menge ganzer Zahlen hat ein größtes Element.

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-11186-0_2,

Für eine reelle Zahl α schreiben wir

$$\lfloor \alpha \rfloor = \max\{b \in \mathbb{Z} : b \leq \alpha\}.$$

Die Zahl $\lfloor \alpha \rfloor$ ist also die größte ganze Zahl, die kleiner als oder gleich α ist. Diese Zahl existiert, weil die Menge $\{b \in \mathbb{Z} : b \leq \alpha\}$ nach oben beschränkt ist.

Beispiel 2.1.1. Es ist $\lfloor 3.43 \rfloor = 3$ und $\lfloor -3.43 \rfloor = -4$.

Schließlich benötigen wir noch das Prinzip der *vollständigen Induktion*. Ist eine Aussage, in der eine unbestimmte natürliche Zahl n vorkommt, richtig für $n = 1$ und folgt aus ihrer Richtigkeit für alle natürlichen Zahlen m mit $1 \leq m \leq n$ (oder auch nur für n) ihre Richtigkeit für $n+1$, so ist die Aussage richtig für jede natürliche Zahl n.

In diesem Kapitel bezeichnen kleine lateinische Buchstaben ganze Zahlen.

2.2 Teilbarkeit

Definition 2.2.1. *Man sagt a* teilt *n, wenn es eine ganze Zahl b gibt mit $n = ab$.*

Wenn a die Zahl n teilt, dann heißt a *Teiler* von n und n *Vielfaches* von a und man schreibt $a \mid n$. Man sagt auch, n ist durch a *teilbar*. Wenn a kein Teiler von n ist, dann schreibt man $a \nmid n$.

Beispiel 2.2.2. Es gilt $13 \mid 182$, weil $182 = 14 * 13$ ist. Genauso gilt $-5 \mid 30$, weil $30 = (-6) * (-5)$ ist. Die Teiler von 30 sind ± 1, ± 2, ± 3, ± 5, ± 6, ± 10, ± 15, ± 30.

Jede ganze Zahl a teilt 0, weil $0 = a * 0$ ist. Die einzige ganze Zahl, die durch 0 teilbar ist, ist 0 selbst, weil aus $a = 0 * b$ folgt, dass $a = 0$ ist.

Wir beweisen einige einfache Regeln.

Theorem 2.2.3. *1. Aus $a \mid b$ und $b \mid c$ folgt $a \mid c$.*
2. Aus $a \mid b$ folgt $ac \mid bc$ für alle c.
3. Aus $c \mid a$ und $c \mid b$ folgt $c \mid da + eb$ für alle d und e.
4. Aus $a \mid b$ und $b \neq 0$ folgt $|a| \leq |b|$.
5. Aus $a \mid b$ und $b \mid a$ folgt $|a| = |b|$.

Beweis. 1. Wenn $a \mid b$ und $b \mid c$, dann gibt es f, g mit $b = af$ und $c = bg$. Also folgt $c = bg = (af)g = a(fg)$. 2. Wenn $a \mid b$, dann gibt es f mit $b = af$. Also folgt $bc = (af)c = f(ac)$. 3. Wenn $c \mid a$ und $c \mid b$, dann gibt es f, g mit $a = fc$ und $b = gc$. Also folgt $da + eb = dfc + egc = (df + eg)c$. 4. Wenn $a \mid b$ und $b \neq 0$, dann gibt es $f \neq 0$ mit $b = af$. Also ist $|b| = |af| \geq |a|$. 5. Gelte $a \mid b$ und $b \mid a$. Wenn $a = 0$, dann gilt $b = 0$ und umgekehrt. Wenn $a \neq 0$ und $b \neq 0$, dann folgt aus 4., dass $|a| \leq |b|$ und $|b| \leq |a|$, also $|a| = |b|$ gilt. □

Das folgende Ergebnis ist sehr wichtig. Es zeigt, dass Division mit Rest von ganzen Zahlen möglich ist.

Theorem 2.2.4. *Wenn a, b ganze Zahlen sind, $b > 0$, dann gibt es eindeutig bestimmte ganze Zahlen q und r derart, dass $a = qb + r$ und $0 \leq r < b$ ist, nämlich $q = \lfloor a/b \rfloor$ und $r = a - bq$.*

Beweis. Gelte $a = qb + r$ und $0 \leq r < b$. Dann folgt $0 \leq r/b = a/b - q < 1$. Dies impliziert $a/b - 1 < q \leq a/b$, also $q = \lfloor a/b \rfloor$. Umgekehrt erfüllen $q = \lfloor a/b \rfloor$ und $r = a - bq$ die Behauptung des Satzes. □

In der Situation von Theorem 2.2.4 nennt man q den (ganzzahligen) *Quotient* und r den *Rest* der Division von a durch b. Man schreibt $r = a \bmod b$. Wird a durch $a \bmod b$ ersetzt, so sagt man auch, dass a modulo b *reduziert* wird.

Beispiel 2.2.5. Wenn $a = 133$ und $b = 21$ ist, dann erhält man $q = 6$ und $r = 7$, d.h. $133 \bmod 21 = 7$. Entsprechend gilt $-50 \bmod 8 = 6$.

2.3 Darstellung ganzer Zahlen

In Büchern werden ganze Zahlen üblicherweise als Dezimalzahlen geschrieben. In Computern werden ganze Zahlen in Binärentwicklung gespeichert. Allgemein kann man ganze Zahlen mit Hilfe der sogenannten g-adischen Darstellung aufschreiben. Diese Darstellung wird jetzt beschrieben. Für eine natürliche Zahl $g > 1$ und eine positive reelle Zahl α bezeichnen wir mit $\log_g \alpha$ den Logarithmus zur Basis g von α. Für eine Menge M bezeichnet M^k die Menge aller Folgen der Länge k mit Gliedern aus M.

Beispiel 2.3.1. Es ist $\log_2 8 = 3$, weil $2^3 = 8$ ist. Ferner ist $\log_8 8 = 1$, weil $8^1 = 8$ ist.

Beispiel 2.3.2. Die Folge $(0, 1, 1, 1, 0)$ ist ein Element von $\{0, 1\}^5$. Ferner ist $\{1, 2\}^2 = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$.

Theorem 2.3.3. *Sei g eine natürliche Zahl, $g > 1$. Für jede natürliche Zahl a gibt es eine eindeutig bestimmte natürliche Zahl k und eine eindeutig bestimmte Folge*

$$(a_1, \ldots, a_k) \in \{0, \ldots, g-1\}^k$$

mit $a_1 \neq 0$ und

$$a = \sum_{i=1}^{k} a_i g^{k-i}. \tag{2.1}$$

Dabei gilt $k = \lfloor \log_g a \rfloor + 1$, und a_i ist der ganzzahlige Quotient der Division von $a - \sum_{j=1}^{i-1} a_j g^{k-j}$ durch g^{k-i} für $1 \leq i \leq k$.

Beweis. Sei a eine natürliche Zahl. Wenn es eine Darstellung von a wie in (2.1) gibt, dann gilt $g^{k-1} \leq a = \sum_{i=1}^{k} a_i g^{k-i} \leq (g-1)\sum_{i=1}^{k} g^{k-i} = g^k - 1 < g^k$. Also ist $k = \lfloor \log_g a \rfloor + 1$. Dies beweist die Eindeutigkeit von k. Wir beweisen die Existenz und Eindeutigkeit der Folge $(a_1, \ldots, a_k)$ durch Induktion über k.

Für $k = 1$ setze $a_1 = a$. Dann ist (2.1) erfüllt, und eine andere Wahl für a_1 hat man nicht.

Sei $k > 1$. Wir beweisen zuerst die Eindeutigkeit. Wenn es eine Darstellung wie in (2.1) gibt, dann gilt $0 \leq a - a_1 g^{k-1} < g^{k-1}$ und daher $0 \leq a/g^{k-1} - a_1 < 1$. Damit ist a_1 der ganzzahlige Quotient der Division von a durch g^{k-1}, also eindeutig bestimmt. Setze $a' = a - a_1 g^{k-1} = \sum_{i=2}^{k} a_i g^{k-i}$. Entweder ist $a' = 0$. Dann ist $a_i = 0$, $2 \leq i \leq n$. Oder $a' = \sum_{i=2}^{k} a_i g^{k-i}$ ist die eindeutig bestimmte Darstellung von a' nach Induktionsannahme. Es ist jetzt auch klar, dass eine Darstellung (2.1) existiert. Man braucht nur $a_1 = \lfloor a/g^{k-1} \rfloor$ zu setzen und die anderen Koeffizienten aus der Darstellung von $a' = a - a_1 g^{k-1}$ zu nehmen.

Definition 2.3.4. *Die Folge* $(a_1, \ldots, a_k)$ *aus Theorem 2.3.3 heißt* g-adische Entwicklung *von* a. *Ihre Glieder heißen* Ziffern. *Ihre* Länge *ist* $k = \lfloor \log_g a \rfloor + 1$. *Falls* $g = 2$ *ist, heißt diese Folge* Binärentwicklung *von* a . *Falls* $g = 16$ *ist, heißt die Folge* Hexadezimalentwicklung *von* a.

Die g-adische Entwicklung einer natürlichen Zahl ist nur dann eindeutig, wenn man verlangt, dass die erste Ziffer von Null verschieden ist. Statt $(a_1, \ldots, a_k)$ schreibt man auch $a_1 a_2 \ldots a_k$.

Beispiel 2.3.5. Die Folge `10101` ist die Binärentwicklung der Zahl $2^4 + 2^2 + 2^0 = 21$. Wenn man Hexadezimaldarstellungen aufschreibt, verwendet man für die Ziffern $10, 11, \ldots, 15$ die Buchstaben A, B, C, D, E, F. So ist `A1C` die Hexadezimaldarstellung von $10 * 16^2 + 16 + 12 = 2588$.

Theorem 2.3.3 enthält ein Verfahren zur Berechnung der g-adischen Entwicklung einer natürlichen Zahl. Das wird im nächsten Beispiel angewandt.

Beispiel 2.3.6. Wir bestimmen die Binärentwicklung von 105. Da $64 = 2^6 < 105 < 128 = 2^7$ ist, hat sie die Länge 7. Wir erhalten: $a_1 = \lfloor 105/64 \rfloor = 1$. $105 - 64 = 41$. $a_2 = \lfloor 41/32 \rfloor = 1$. $41 - 32 = 9$. $a_3 = \lfloor 9/16 \rfloor = 0$. $a_4 = \lfloor 9/8 \rfloor = 1$. $9 - 8 = 1$. $a_5 = a_6 = 0$. $a_7 = 1$. Also ist die Binärentwicklung von 105 die Folge 1101001.

Die Umwandlung von Hexadezimalentwicklungen in Binärentwicklungen und umgekehrt ist besonders einfach. Sei $(h_1, h_2, \ldots, h_k)$ die Hexadezimalentwicklung einer natürlichen Zahl n. Für $1 \leq i \leq k$ sei $(b_{1,i}, b_{2,i}, b_{3,i}, b_{4,i})$ der Bitstring der Länge 4, der h_i darstellt, also $h_i = b_{1,i}2^3 + b_{2,i}2^2 + b_{3,i}2 + b_{4,i}$, dann ist $(b_{1,1}, b_{2,1}, b_{3,1}, b_{4,1}, b_{1,2}, \ldots, b_{4,k})$ die Binärentwicklung von n.

Beispiel 2.3.7. Betrachte die Hexadezimalzahl $n = 6EF$. Die auf Länge 4 normierten Binärentwicklungen der Ziffern sind $6 = 0110$, $E = 1110$, $F = 1111$. Daher ist 011011101111 die Binärentwicklung von n.

Die Länge der Binärentwicklung einer natürlichen Zahl wird auch als ihre *binäre Länge* bezeichnet. Die binäre Länge von 0 wird auf 1 gesetzt. Die binäre Länge einer ganzen Zahl ist die binäre Länge ihres Absolutbetrags. Die binäre Länge einer ganzen Zahl a wird auch mit $\mathrm{size}(a)$ oder $\mathrm{size}\, a$ bezeichnet.

2.4 O- und Ω-Notation

Beim Design eines kryptographischen Algorithmus ist es nötig, abzuschätzen, welchen Berechnungsaufwand er hat und welchen Speicherplatz er benötigt. Um solche Aufwandsabschätzungen zu vereinfachen, ist es nützlich, die O- und die Ω-Notation zu verwenden.

Seien k eine natürliche Zahl, $X, Y \subset \mathbb{N}^k$ und $f : X \to \mathbb{R}$, $g : Y \to \mathbb{R}$ Funktionen. Man schreibt $f = O(g)$, falls es positive reelle Zahlen B und C gibt derart, dass für alle $(n_1, \ldots, n_k) \in \mathbb{N}^k$ mit $n_i > B$, $1 \leq i \leq k$, folgendes gilt:

1. $(n_1, \ldots, n_k) \in X \cap Y$, d.h. $f(n_1, \ldots, n_k)$ und $g(n_1, \ldots, n_k)$ sind definiert,
2. $f(n_1, \ldots, n_k) \leq Cg(n_1, \ldots, n_k)$.

Das bedeutet, dass fast überall $f(n_1, \ldots, n_k) \leq Cg(n_1, \ldots, n_k)$ gilt. Man schreibt dann auch $g = \Omega(f)$. Ist g eine Konstante, so schreibt man $f = O(1)$.

Beispiel 2.4.1. Es ist $2n^2 + n + 1 = O(n^2)$, weil $2n^2 + n + 1 \leq 4n^2$ ist für alle $n \geq 1$. Außerdem ist $2n^2 + n + 1 = \Omega(n^2)$, weil $2n^2 + n + 1 \geq 2n^2$ ist für alle $n \geq 1$.

Beispiel 2.4.2. Ist g eine natürliche Zahl, $g > 2$ und bezeichnet $f(n)$ die Länge der g-adischen Entwicklung einer natürlichen Zahl n, so gilt $f(n) = O(\log n)$, wobei $\log n$ der natürliche Logarithmus von n ist. Diese Länge ist nämlich $\lfloor \log_g n \rfloor + 1 \leq \log_g n + 1 = \log n / \log g + 1$. Für $n > 3$ ist $\log n > 1$ und daher ist $\log n / \log g + 1 < (1/\log g + 1) \log n$.

2.5 Aufwand von Addition, Multiplikation und Division mit Rest

In vielen kryptographischen Verfahren werden lange ganze Zahlen addiert, multipliziert und mit Rest dividiert. Um die Laufzeit solcher Verfahren abschätzen zu können, muss man untersuchen, wie lange diese Operationen brauchen. Man legt dazu ein Rechenmodell fest, das den tatsächlichen Computern möglichst ähnlich ist. Dies wird sehr sorgfältig und ausführlich in [3]

und [4] gemacht. Hier wird nur ein naives Modell beschrieben, das aber eine gute Abschätzung für die benötigte Rechenzeit liefert.

Es seien a und b natürliche Zahlen, die durch ihre Binärentwicklungen gegeben seien. Die binäre Länge von a sei m und die binäre Länge von b sei n. Um $a+b$ zu berechnen, schreibt man die Binärentwicklungen von a und b untereinander und addiert Bit für Bit mit Übertrag.

Beispiel 2.5.1. Sei $a = 10101$, $b = 111$. Wir berechnen $a + b$.

	1	0	1	0	1
+			1	1	1
Übertrag		1	1	1	
	1	1	1	0	0

Wir nehmen an, dass die Addition von zwei Bits Zeit $O(1)$ braucht. Dann braucht die gesamte Addition Zeit $O(\max\{m,n\})$. Entsprechend zeigt man, dass man b von a in Zeit $O(\max\{m,n\})$ subtrahieren kann. Daraus folgt, dass die Addition zweier ganzer Zahlen a und b mit Binärlänge m und n Zeit $O(\max\{m,n\})$ kostet.

Auch bei der Multiplikation gehen wir ähnlich vor wie in der Schule.

Beispiel 2.5.2. Sei $a = 10101$, $b = 101$. Wir berechnen $a * b$.

1	0	1	0	1	*	1	0	1	
				1	0	1	0	1	
+		1	0	1	0	1			
Übertrag			1		1				
		1	1	0	1	0	0	1	

Man geht b von hinten nach vorn durch. Für jede 1 schreibt man a auf und zwar so, dass das am weitesten rechts stehende Bit von a unter der 1 von b steht. Dann addiert man dieses a zu dem vorigen Ergebnis. Jede solche Addition kostet Zeit $O(m)$ und es gibt höchstens $O(n)$ Additionen. Die Berechnung kostet also Zeit $O(mn)$. In [3] wird die Methode von Schönhage und Strassen erläutert, die zwei n-Bit-Zahlen in Zeit $O(n \log n \log\log n)$ multipliziert. In der Praxis ist diese Methode für Zahlen, die eine kürzere binäre Länge als 10000 haben, aber langsamer als die Schulmethode.

Um a durch b mit Rest zu dividieren, verwendet man ebenfalls die Schulmethode.

Beispiel 2.5.3. Sei $a = 10101$, $b = 101$. Wir dividieren a mit Rest durch b.

1	0	1	0	1	=	1	0	1	*	1	0	0	+	1
1	0	1												
	0	0	0											
	0	0	0											
		0	0	1										
		0	0	0										
				1										

Analysiert man diesen Algorithmus, stellt man folgendes fest: Sei k die Anzahl der Bits des Quotienten. Dann muss man höchstens k-mal zwei Zahlen mit binärer Länge $\leq n+1$ voneinander abziehen. Dies kostet Zeit $O(kn)$.

Zusammenfassend erhalten wir folgende Schranken, die wir in Zukunft benutzen wollen.

1. Die Addition von a und b erfordert Zeit $O(\max\{\operatorname{size} a, \operatorname{size} b\})$.
2. Die Multiplikation von a und b erfordert Zeit $O((\operatorname{size} a)(\operatorname{size} b))$.
3. Die Division mit Rest von a durch b erfordert Zeit $O((\operatorname{size} b)(\operatorname{size} q))$, wobei q der Quotient ist.

Der benötigte Platz ist $O(\operatorname{size} a + \operatorname{size} b)$.

2.6 Polynomzeit

Bei der Analyse eines kryptographischen Verfahrens muss man zeigen, dass das Verfahren in der Praxis effizient funktioniert, aber nicht effizient gebrochen werden kann. Wir präzisieren den Begriff "effizient".

Angenommen, ein Algorithmus bekommt als Eingabe ganze Zahlen z_1, ..., z_n. Man sagt, dass dieser Algorithmus *polynomielle Laufzeit* hat, wenn es nicht negative ganze Zahlen $e_1, \ldots, e_n$ gibt, so dass der Algorithmus die Laufzeit

$$O((\operatorname{size} z_1)^{e_1}(\operatorname{size} z_2)^{e_2} \cdots (\operatorname{size} z_n)^{e_n})$$

hat. Der Algorithmus gilt als effizient, wenn er polynomielle Laufzeit hat. Man muss allerdings beachten, dass der Algorithmus nur dann als praktisch effizient gelten kann, wenn die O-Konstanten und die Exponenten e_i klein sind.

2.7 Größter gemeinsamer Teiler

Wir führen den größten gemeinsamen Teiler zweier ganzer Zahlen ein.

Definition 2.7.1. *Ein* gemeinsamer Teiler *von a und b ist eine ganze Zahl c, die sowohl a als auch b teilt.*

Theorem 2.7.2. *Unter allen gemeinsamen Teilern zweier ganzer Zahlen a und b, die nicht beide gleich 0 sind, gibt es genau einen (bezüglich $\leq$) größten. Dieser heißt* größter gemeinsamer Teiler (ggT) *von a und b und wird mit $\gcd(a,b)$ bezeichnet. Die Abkürzung gcd steht für greatest common divisor.*

Beweis. Sei $a \neq 0$. Nach Theorem 2.2.3 sind alle Teiler von a durch $|a|$ beschränkt. Daher muss es unter allen Teilern von a und damit unter allen gemeinsamen Teilern von a und b einen größten geben. □

Der Vollständigkeit halber wird der größte gemeinsame Teiler von 0 und 0 auf 0 gesetzt, also $\gcd(0,0) = 0$. Der größte gemeinsame Teiler zweier ganzer Zahlen ist also nie negativ.

Beispiel 2.7.3. Der größte gemeinsame Teiler von 18 und 30 ist 6. Der größte gemeinsame Teiler von -10 und 20 ist 10. Der größte gemeinsame Teiler von -20 und -14 ist 2. Der größte gemeinsame Teiler von 12 und 0 ist 12.

Der größte gemeinsame Teiler von ganzen Zahlen $a_1, \ldots, a_k$, $k \geq 1$, wird entsprechend definiert: Ist wenigstens eine der Zahlen a_i von Null verschieden, so ist $\gcd(a_1, \ldots, a_k)$ die größte natürliche Zahl, die alle a_i teilt. Sind alle a_i gleich 0, so wird $\gcd(a_1, \ldots, a_k) = 0$ gesetzt.

Wir geben als nächstes eine besondere Darstellung des größten gemeinsamen Teilers an. Dazu brauchen wir eine Bezeichnung.

Sind $\alpha_1, \ldots, \alpha_k$ reelle Zahlen, so schreibt man

$$\alpha_1\mathbb{Z} + \ldots + \alpha_k\mathbb{Z} = \{\alpha_1 z_1 + \ldots + \alpha_k z_k : z_i \in \mathbb{Z}, 1 \leq i \leq k\}.$$

Dies ist die Menge aller *ganzzahligen Linearkombinationen* der α_i.

Beispiel 2.7.4. Die Menge der ganzzahligen Linearkombinationen von 3 und 4 ist $3\mathbb{Z} + 4\mathbb{Z}$. Sie enthält die Zahl $1 = 3 * (-1) + 4$. Sie enthält auch alle ganzzahligen Vielfachen von 1. Also ist diese Menge gleich $\mathbb{Z}$.

Der nächste Satz zeigt, dass das Ergebnis des vorigen Beispiels kein Zufall ist.

Theorem 2.7.5. *Die Menge aller ganzzahligen Linearkombinationen von a und b ist die Menge aller ganzzahligen Vielfachen von* $\gcd(a,b)$*, also*

$$a\mathbb{Z} + b\mathbb{Z} = \gcd(a,b)\mathbb{Z}.$$

Beweis. Für $a = b = 0$ ist die Behauptung offensichtlich korrekt. Also sei angenommen, dass a oder b nicht 0 ist.

Setze

$$I = a\mathbb{Z} + b\mathbb{Z}.$$

Sei g die kleinste positive ganze Zahl in I. Wir behaupten, dass $I = g\mathbb{Z}$ gilt. Um dies einzusehen, wähle ein von Null verschiedenes Element c in I. Wir müssen zeigen, dass $c = qg$ für ein q gilt. Nach Theorem 2.2.4 gibt es q, r mit $c = qg + r$ und $0 \leq r < g$. Also gehört $r = c - qg$ zu I. Da aber g die kleinste positive Zahl in I ist, muss $r = 0$ und $c = qg$ gelten.

Es bleibt zu zeigen, dass $g = \gcd(a,b)$ gilt. Da $a, b \in I$ ist, folgt aus $I = g\mathbb{Z}$, dass g ein gemeinsamer Teiler von a und b ist. Da ferner $g \in I$ ist, gibt es x, y mit $g = xa + yb$. Ist also d ein gemeinsamer Teiler von a und b, dann ist d auch ein Teiler von g. Daher impliziert Theorem 2.2.3, dass $|d| \leq g$ gilt. Damit ist g der größte gemeinsame Teiler von a und b. □

Das Ergebnis von Beispiel 2.7.4 hätte man direkt aus Theorem 2.7.5 folgern können. Es ist nämlich $\gcd(3,4) = 1$ und daher $3\mathbb{Z} + 4\mathbb{Z} = 1\mathbb{Z} = \mathbb{Z}$.

Theorem 2.7.5 hat einige wichtige Folgerungen.

Korollar 2.7.6. *Für alle a, b, n ist die Gleichung $ax + by = n$ genau dann durch ganze Zahlen x und y lösbar, wenn $\gcd(a,b)$ ein Teiler von n ist.*

Beweis. Gibt es ganze Zahlen x und y mit $n = ax + by$, dann gehört n zu $a\mathbb{Z} + b\mathbb{Z}$ und nach Theorem 2.7.5 damit auch zu $\gcd(a,b)\mathbb{Z}$. Man kann also $n = c\gcd(a,b)$ schreiben, und das bedeutet, dass n ein Vielfaches von $\gcd(a,b)$ ist.

Ist umgekehrt n ein Vielfaches von $\gcd(a,b)$, dann gehört n zu der Menge $\gcd(a,b)\mathbb{Z}$. Nach Theorem 2.7.5 gehört n also auch zu $a\mathbb{Z}+b\mathbb{Z}$. Es gibt daher ganze Zahlen x und y mit $n = ax + by$. □

Korollar 2.7.6 sagt uns, dass die Gleichung

$$3x + 4y = 123$$

eine Lösung hat, weil $\gcd(3,4) = 1$ ist und 123 ein Vielfaches von 1 ist. Wir kennen aber noch keine effiziente Methode, um eine Lösung x und y zu berechnen. Man kann das mit dem euklidischen Algorithmus machen, der im nächsten Abschnitt erklärt wird.

Korollar 2.7.7. *Es gibt ganze Zahlen x und y mit $ax + by = \gcd(a,b)$.*

Beweis. Weil $\gcd(a,b)$ ein Teiler von sich selbst ist, folgt die Behauptung unmittelbar aus Korollar 2.7.6. □

Wir geben noch eine andere nützliche Charakterisierung des größten gemeinsamen Teilers an. Diese Charakterisierung wird auch häufig als Definition des größten gemeinsamen Teilers verwendet.

Korollar 2.7.8. *Es gibt genau einen nicht negativen gemeinsamen Teiler von a und b, der von allen gemeinsamen Teilern von a und b geteilt wird. Dieser ist der größte gemeinsame Teiler von a und b.*

Beweis. Der größte gemeinsame Teiler von a und b ist ein nicht negativer gemeinsamer Teiler von a und b. Außerdem gibt es nach Korollar 2.7.7 ganze Zahlen x und y mit $ax+by = \gcd(a,b)$. Daher ist jeder gemeinsame Teiler von a und b auch ein Teiler von $\gcd(a,b)$. Damit ist gezeigt, dass es einen nicht negativen gemeinsamen Teiler von a und b gibt, der von allen gemeinsamen Teilern von a und b geteilt wird.

Sei umgekehrt g ein nicht negativer gemeinsamer Teiler von a und b, der von jedem gemeinsamen Teiler von a und b geteilt wird. Ist $a = b = 0$, so ist $g = 0$, weil nur 0 von 0 geteilt wird. Ist a oder b von Null verschieden, dann ist nach Theorem 2.2.3 jeder gemeinsame Teiler von a und b kleiner oder gleich g. Damit ist $g = \gcd(a,b)$. □

Es bleibt die Frage, wie $\gcd(a, b)$ berechnet wird und wie ganze Zahlen x und y bestimmt werden, die $ax + by = \gcd(a, b)$ erfüllen. Der Umstand, dass diese beiden Probleme effiziente Lösungen besitzen, ist zentral für fast alle kryptographische Techniken.

Beide Probleme werden mit dem euklidischen Algorithmus gelöst, der im nächsten Abschnitt erläutert wird.

2.8 Euklidischer Algorithmus

Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler zweier natürlicher Zahlen sehr effizient. Er beruht auf folgendem Satz:

Theorem 2.8.1. *1. Wenn $b = 0$ ist, dann ist $\gcd(a, b) = |a|$.*
2. Wenn $b \neq 0$ ist, dann ist $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$.

Beweis. Die erste Behauptung ist offensichtlich korrekt. Wir beweisen die zweite. Sei $b \neq 0$. Nach Theorem 2.2.4 gibt es eine ganze Zahl q mit $a = q|b| + (a \bmod |b|)$. Daher teilt der größte gemeinsame Teiler von a und b auch den größten gemeinsamen Teiler von $|b|$ und $a \bmod |b|$ und umgekehrt. Da beide größte gemeinsame Teiler nicht negativ sind, folgt die Behauptung aus Theorem 2.2.3. □

Wir erläutern den euklidischen Algorithmus erst an einem Beispiel.

Beispiel 2.8.2. Wir möchten $\gcd(100, 35)$ berechnen. Nach Theorem 2.8.1 erhalten wir $\gcd(100, 35) = \gcd(35, 100 \bmod 35) = \gcd(35, 30) = \gcd(30, 5) = \gcd(5, 0) = 5$.

Zuerst ersetzt der euklidische Algorithmus a durch $|a|$ und b durch $|b|$. Dies hat in unserem Beispiel keinen Effekt. Solange b nicht Null ist, ersetzt der Algorithmus a durch b und b durch $a \bmod b$. Sobald $b = 0$ ist, wird a zurückgegeben. In Abbildung 2.1 ist der euklidische Algorithmus im Pseudocode dargestellt.

Theorem 2.8.3. *Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler von a und b.*

Beweis. Um zu beweisen, dass der euklidische Algorithmus abbricht und dann tatsächlich den größten gemeinsamen Teiler von a und b zurückgibt, führen wir folgende Bezeichnungen ein: Wir setzen

$$r_0 = |a|, r_1 = |b| \tag{2.2}$$

und für $k \geq 1$ und $r_k \neq 0$

$$r_{k+1} = r_{k-1} \bmod r_k. \tag{2.3}$$

```
euclid(int a, int b, int gcd)
begin
  int r
  a = |a|
  b = |b|
  while (b != 0)
    r = a%b
    a = b
    b = r
  end while
  gcd = a
end
```

Abb. 2.1. Der euklidische Algorithmus

Dann ist $r_2, r_3, \ldots$ die Folge der Reste, die in der `while`-Schleife des euklidischen Algorithmus ausgerechnet wird. Außerdem gilt nach dem k-ten Durchlauf der `while`-Schleife im euklidischen Algorithmus

$$a = r_k, \quad b = r_{k+1}.$$

Aus Theorem 2.8.1 folgt, dass sich der größte gemeinsame Teiler von a und b nicht ändert. Um zu zeigen, dass der euklidische Algorithmus tatsächlich den größten gemeinsamen Teiler von a und b berechnet, brauchen wir also nur zu beweisen, dass ein r_k schließlich 0 ist. Das folgt aber daraus, dass nach (2.3) die Folge $(r_k)_{k\geq 1}$ streng monoton fallend ist. Damit ist die Korrektheit des euklidischen Algorithmus bewiesen. □

Der euklidische Algorithmus berechnet $\gcd(a, b)$ sehr effizient. Das ist wichtig für kryptographische Anwendungen. Um dies zu beweisen, wird die Anzahl der Iterationen im euklidischen Algorithmus abgeschätzt. Dabei wird der euklidische Algorithmus Schritt für Schritt untersucht. Zur Vereinfachung nehmen wir an, dass

$$a > b > 0$$

ist. Dies ist keine Einschränkung, weil der euklidische Algorithmus einen Schritt braucht, um $\gcd(a, b)$ zu bestimmen (wenn $b = 0$ ist) oder diese Situation herzustellen.

Sei r_n das letzte von Null verschiedene Glied der Restefolge (r_k). Dann ist n die Anzahl der Iterationen, die der euklidische Algorithmus braucht, um $\gcd(a, b)$ auszurechnen. Sei weiter

$$q_k = \lfloor r_{k-1}/r_k \rfloor, \quad 1 \leq k \leq n. \tag{2.4}$$

Die Zahl q_k ist also der Quotient der Division von r_{k-1} durch r_k und es gilt

$$r_{k-1} = q_k r_k + r_{k+1}. \tag{2.5}$$

Beispiel 2.8.4. Ist $a = 100$ und $b = 35$, dann erhält man die Folge

k	0	1	2	3	4
r_k	100	35	30	5	0
q_k		2	1	6	

Um die Anzahl n der Iterationen des euklidischen Algorithmus abzuschätzen, beweisen wir folgendes Hilfsresultat. Hierin ist $a > b > 0$ vorausgesetzt.

Lemma 2.8.5. *Es gilt $q_k \geq 1$ für $1 \leq k \leq n-1$ und $q_n \geq 2$.*

Beweis. Da $r_{k-1} > r_k > r_{k+1}$ gilt, folgt aus (2.5), dass $q_k \geq 1$ ist für $1 \leq k \leq n$. Angenommen, $q_n = 1$. Dann folgt $r_{n-1} = r_n$ und das ist nicht möglich, weil die Restefolge streng monoton fällt. Daher ist $q_n \geq 2$. □

Theorem 2.8.6. *Im euklidischen Algorithmus sei $a > b > 0$. Setze $\Theta = (1+\sqrt{5})/2$. Dann ist die Anzahl der Iterationen im euklidischen Algorithmus höchstens $(\log b)/(\log \Theta) + 1 < 1.441 * \log_2(b) + 1$.*

Beweis. Nach Übung 2.12.19 können wir annehmen, dass $\gcd(a, b) = r_n = 1$ ist. Durch Induktion wird bewiesen, dass

$$r_k \geq \Theta^{n-k}, \quad 0 \leq k \leq n \tag{2.6}$$

gilt. Dann ist insbesondere

$$b = r_1 \geq \Theta^{n-1}.$$

Durch Logarithmieren erhält man daraus

$$n \leq (\log b)/(\log \Theta) + 1,$$

wie behauptet.

Wir beweisen nun (2.6). Zunächst gilt

$$r_n = 1 = \Theta^0$$

und nach Lemma 2.8.5

$$r_{n-1} = q_n r_n = q_n \geq 2 > \Theta.$$

Sei $n - 2 \geq k \geq 0$ und gelte die Behauptung für $k' > k$. Dann folgt aus Lemma 2.8.5

$$\begin{aligned} r_k &= q_{k+1} r_{k+1} + r_{k+2} \geq r_{k+1} + r_{k+2} \\ &\geq \Theta^{n-k-1} + \Theta^{n-k-2} = \Theta^{n-k-1}\left(1 + \frac{1}{\Theta}\right) = \Theta^{n-k}. \end{aligned}$$

Damit sind (2.6) und das Theorem bewiesen. □

2.9 Erweiterter euklidischer Algorithmus

Im vorigen Abschnitt haben wir gesehen, wie man den größten gemeinsamen Teiler zweier ganzer Zahlen berechnen kann. In Korollar 2.7.7 wurde gezeigt, dass es ganze Zahlen x, y gibt, so dass $\gcd(a, b) = ax + by$ ist. In diesem Abschnitt erweitern wir den euklidischen Algorithmus so, dass er solche Koeffizienten x und y berechnet. Wie in Abschnitt 2.8 bezeichnen wir mit $r_0, \ldots, r_{n+1}$ die Restefolge und mit $q_1, \ldots, q_n$ die Folge der Quotienten, die bei der Anwendung des euklidischen Algorithmus auf a, b entstehen.

Wir erläutern nun die Konstruktion zweier Folgen (x_k) und (y_k), für die $x = (-1)^n x_n$ und $y = (-1)^{n+1} y_n$ die gewünschte Eigenschaft haben.

Wir setzen

$$x_0 = 1, x_1 = 0, y_0 = 0, y_1 = 1.$$

Ferner setzen wir

$$x_{k+1} = q_k x_k + x_{k-1}, \quad y_{k+1} = q_k y_k + y_{k-1}, \quad 1 \leq k \leq n. \tag{2.7}$$

Wir nehmen an, dass a und b nicht negativ sind.

Theorem 2.9.1. *Es gilt* $r_k = (-1)^k x_k a + (-1)^{k+1} y_k b$ *für* $0 \leq k \leq n + 1$.

Beweis. Es ist

$$r_0 = a = 1 * a - 0 * b = x_0 * a - y_0 * b.$$

Weiter ist

$$r_1 = b = -0 * a + 1 * b = -x_1 * a + y_1 * b.$$

Sei nun $k \geq 2$ und gelte die Behauptung für alle $k' < k$. Dann ist

$$\begin{aligned} r_k &= r_{k-2} - q_{k-1} r_{k-1} \\ &= (-1)^{k-2} x_{k-2} a + (-1)^{k-1} y_{k-2} b - q_{k-1}((-1)^{k-1} x_{k-1} a + (-1)^k y_{k-1} b) \\ &= (-1)^k a(x_{k-2} + q_{k-1} x_{k-1}) + (-1)^{k+1} b(y_{k-2} + q_{k-1} y_{k-1}) \\ &= (-1)^k x_k a + (-1)^{k+1} y_k b. \end{aligned}$$

Damit ist das Theorem bewiesen. □

Man sieht, dass insbesondere

$$r_n = (-1)^n x_n a + (-1)^{n+1} y_n b$$

ist. Damit ist also der größte gemeinsame Teiler von a und b als Linearkombination von a und b dargestellt.

Beispiel 2.9.2. Wähle $a = 100$ und $b = 35$. Dann kann man die Werte r_k, q_k, x_k und y_k aus folgender Tabelle entnehmen.

k	0	1	2	3	4
r_k	100	35	30	5	0
q_k		2	1	6	
x_k	1	0	1	1	7
y_k	0	1	2	3	20

Damit ist $n = 3$ und $\gcd(100, 35) = 5 = -1 * 100 + 3 * 35$.

Der erweiterte euklidische Algorithmus berechnet neben $\gcd(a, b)$ auch die Koeffizienten

$$x = (-1)^n x_n \quad y = (-1)^{n+1} y_n$$

Den erweiterten euklidischen Algorithmus findet man in Abbildung 2.2.

Die Korrektheit dieses Algorithmus folgt aus Theorem 2.9.1.

2.10 Analyse des erweiterten euklidischen Algorithmus

Als erstes werden wir die Größe der Koeffizienten x und y abschätzen, die der erweiterte euklidische Algorithmus berechnet. Das ist wichtig dafür, dass der erweiterte euklidische Algorithmus von Anwendungen effizient benutzt werden kann.

Wir brauchen die Matrizen

$$E_k = \begin{pmatrix} q_k & 1 \\ 1 & 0 \end{pmatrix}, \quad 1 \leq k \leq n,$$

und

$$T_k = \begin{pmatrix} y_k & y_{k-1} \\ x_k & x_{k-1} \end{pmatrix}, \quad 1 \leq k \leq n+1.$$

Es gilt

$$T_{k+1} = T_k E_k, \quad 1 \leq k \leq n$$

und da T_1 die Einheitsmatrix ist, folgt

$$T_{n+1} = E_1 E_2 \cdots E_n.$$

Setzt man nun

$$S_k = E_{k+1} E_{k+2} \cdots E_n, \quad 0 \leq k \leq n,$$

wobei S_n die Einheitsmatrix ist, so gilt

$$S_0 = T_{n+1}.$$

Wir benutzen die Matrizen S_k, um die Zahlen x_n und y_n abzuschätzen. Schreibt man

$$S_k = \begin{pmatrix} u_k & v_k \\ u_{k+1} & v_{k+1} \end{pmatrix}, \quad 0 \leq k \leq n,$$

```
xeuclid(int a, int b,int gcd, int x, int y) {
begin

  int q, r, xx, yy, sign
  int xs[2], ys[2]

  // Die Koeffizienten werden initialisiert.

  xs[0] = 1 xs[1] = 0
  ys[0] = 0 ys[1] = 1
  sign = 1

  // Solange b != 0 ist, wird a durch b und b durch a%b
  // ersetzt.
  // Ausserdem werden die Koeffizienten neu berechnet.

  while (b != 0)
    r = a%b
    q = a/b
    a = b
    b = r
    xx = xs[1]
    yy = ys[1]
    xs[1] = q*xs[1] + xs[0]
    ys[1] = q*ys[1] + ys[0]
    xs[0] = xx
    ys[0] = yy
    sign = -sign
  end while

  // Die Koeffizienten werden endgueltig berechnet.

  x = sign*xs[0]
  y = -sign*ys[0]

  // Der ggT wird berechnet

  gcd = a
end
```

Abb. 2.2. Der erweiterte euklidische Algorithmus

so gelten wegen

$$S_{k-1} = E_k S_k, \quad 1 \le k \le n$$

die Rekursionen

$$u_{k-1} = q_k u_k + u_{k+1}, \quad v_{k-1} = q_k v_k + v_{k+1}, \quad 1 \le k \le n. \tag{2.8}$$

Eine analoge Rekursion gilt auch für die Reste r_k, die im euklidischen Algorithmus berechnet werden.

Die Einträge v_k der Matrizen S_k werden jetzt abgeschätzt.

Lemma 2.10.1. *Es gilt* $0 \le v_k \le r_k/(2\gcd(a,b))$ *für* $0 \le k \le n$.

Beweis. Es gilt $0 = v_n < r_n/(2\gcd(a,b))$. Außerdem ist $q_n \ge 2$ nach Lemma 2.8.5 und $v_{n-1} = 1$. Daher ist $r_{n-1} = q_n r_n \ge 2\gcd(a,b) \ge 2\gcd(a,b)v_{n-1}$. Angenommen, die Behauptung stimmt für $k' \ge k$. Dann folgt $v_{k-1} = q_k v_k + v_{k+1} \le (q_k r_k + r_{k+1})/(2\gcd(a,b)) = r_{k-1}/(2\gcd(a,b))$. Damit ist die behauptete Abschätzung bewiesen. □

Aus Lemma 2.10.1 können wir Abschätzungen für die Koeffizienten x_k und y_k ableiten.

Korollar 2.10.2. *Es gilt* $x_k \le b/(2\gcd(a,b))$ *und* $y_k \le a/(2\gcd(a,b))$ *für* $1 \le k \le n$.

Beweis. Aus $S_0 = T_{n+1}$ folgt $x_n = v_1$ und $y_n = v_0$. Aus Lemma 2.10.1 folgt also die behauptete Abschätzung für $k = n$. Da aber $(x_k)_{k\ge1}$ und $(y_k)_{k\ge0}$ monoton wachsende Folgen sind, ist die Behauptung für $1 \le k \le n$ bewiesen. □

Für die Koeffizienten x und y, die der erweiterte euklidische Algorithmus berechnet, gewinnt man daraus die folgende Abschätzung:

Korollar 2.10.3. *Es gilt* $|x| \le b/(2\gcd(a,b))$ *und* $|y| \le a/(2\gcd(a,b))$.

Wir können auch noch die Koeffizienten x_{n+1} und y_{n+1} bestimmen.

Lemma 2.10.4. *Es gilt* $x_{n+1} = b/\gcd(a,b)$ *und* $y_{n+1} = a/\gcd(a,b)$.

Den Beweis dieses Lemmas überlassen wir dem Leser.

Wir können jetzt die Laufzeit des euklidischen Algorithmus abschätzen. Es stellt sich heraus, dass die Zeitschranke für die Anwendung des erweiterten euklidischen Algorithmus auf a und b von derselben Größenordnung ist wie die Zeitschranke für die Multiplikation von a und b. Das ist ein erstaunliches Resultat, weil der erweiterte euklidische Algorithmus viel aufwendiger aussieht als die Multiplikation.

Theorem 2.10.5. *Sind a und b ganze Zahlen, dann braucht die Anwendung des erweiterten euklidischen Algorithmus auf a und b Zeit* $O((\operatorname{size} a)(\operatorname{size} b))$.

Beweis. Wir nehmen an, dass $a > b > 0$ ist. Wir haben ja bereits gesehen, dass der erweiterte euklidische Algorithmus nach höchstens einer Iteration entweder fertig ist oder diese Annahme gilt. Es ist leicht einzusehen, dass dafür Zeit $O(\text{size}(a)\,\text{size}(b))$ nötig ist.

Im euklidischen Algorithmus wird die Restefolge $(r_k)_{2\le k\le n+1}$ und die Quotientenfolge $(q_k)_{1\le k\le n}$ berechnet. Die Zahl r_{k+1} ist der Rest der Division von r_{k-1} durch r_k für $1 \le k \le n$. Wie in Abschnitt 2.5 dargestellt, kostet die Berechnung von r_{k+1} höchstens Zeit $O(\text{size}(r_k)\,\text{size}(q_k))$, wobei q_k der Quotient der Division ist.

Wir wissen, dass $r_k \le b$, also $\text{size}(r_k) \le \text{size}(b)$ ist für $1 \le k \le n+1$. Wir wissen ferner, dass $\text{size}(q_k) \le \log(q_k) + 1$ ist für $1 \le k \le n$. Also benötigt der euklidische Algorithmus Zeit

$$T_1(a,b) = O(\text{size}(b)(n + \sum_{k=1}^{n} \log q_k)). \tag{2.9}$$

Nach Theorem 2.8.6 ist

$$n = O(\text{size}\, b). \tag{2.10}$$

Ferner ist

$$\begin{aligned} a = r_0 &= q_1 r_1 + r_2 \ge q_1 r_1 = q_1(q_2 r_2 + r_3) \\ &\ge q_1 q_2 r_2 > \ldots \ge q_1 q_2 \cdots q_n. \end{aligned}$$

Daraus folgt

$$\sum_{k=1}^{n} \log q_k = O(\text{size}\, a). \tag{2.11}$$

Setzt man (2.10) und (2.11) in (2.9) ein, ist die Laufzeitabschätzung für den einfachen euklidischen Algorithmus bewiesen.

Wir schätzen auch noch die Rechenzeit ab, die der erweiterte euklidische Algorithmus benötigt, um die Koeffizienten x und y zu berechnen. In der ersten Iteration wird

$$x_2 = q_1 x_1 + x_0 = 1, \quad y_2 = q_1 y_1 + y_0 = q_1$$

berechnet. Das kostet Zeit $O(\text{size}(q_1)) = O(\text{size}(a))$. Danach wird

$$x_{k+1} = q_k x_k + x_{k-1}, \quad y_{k+1} = q_k y_k + y_{k-1}$$

berechnet, und zwar für $2 \le k \le n$. Gemäß Lemma 2.10.2 ist $x_k, y_k = O(a)$ für $0 \le k \le n$. Damit ist die Laufzeit, die die Berechnung der Koeffizienten x und y braucht

$$T_2(a,b) = O(\text{size}(a)(1 + \sum_{k=2}^{n} \text{size}(q_k)) = O(\text{size}(a)(n + \sum_{k=2}^{n} \log q_k)). \tag{2.12}$$

Wie oben beweist man leicht

$$\prod_{k=2}^{n} q_k \leq b. \tag{2.13}$$

Setzt man dies in (2.12) ein, folgt die Behauptung. Damit ist das Theorem bewiesen. □

2.11 Zerlegung in Primzahlen

Ein zentraler Begriff in der elementaren Zahlentheorie ist der einer Primzahl. Primzahlen werden auch in vielen kryptographischen Verfahren benötigt. In diesem Abschnitt führen wir Primzahlen ein und beweisen, dass sich jede natürliche Zahl bis auf die Reihenfolge in eindeutiger Weise als Produkt von Primzahlen schreiben lässt.

Definition 2.11.1. *Eine natürliche Zahl $p > 1$ heißt* Primzahl, *wenn sie genau zwei positive Teiler hat, nämlich 1 und p.*

Die ersten neun Primzahlen sind $2, 3, 5, 7, 11, 13, 17, 19, 23$. Die Menge aller Primzahlen bezeichnen wir mit $\mathbb{P}$. Eine natürliche Zahl $a > 1$, die keine Primzahl ist, heißt *zusammengesetzt* . Wenn die Primzahl p die ganze Zahl a teilt, dann heißt p *Primteiler* von a.

Theorem 2.11.2. *Jede natürliche Zahl $a > 1$ hat einen Primteiler.*

Beweis. Die Zahl a besitzt einen Teiler, der größer als 1 ist, nämlich a selbst. Unter allen Teilern von a, die größer als 1 sind, sei p der kleinste. Die Zahl p muss eine Primzahl sein. Wäre sie nämlich keine Primzahl, dann besäße sie einen Teiler b, der

$$1 < b < p \leq a$$

erfüllt. Dies widerspricht der Annahme, dass p der kleinste Teiler von a ist, der größer als 1 ist. □

Das folgende Resultat ist zentral für den Beweis des Zerlegungssatzes.

Lemma 2.11.3. *Wenn eine Primzahl p ein Produkt zweier ganzer Zahlen teilt, so teilt p wenigstens einen der beiden Faktoren.*

Beweis. Angenommen, p teilt ab, aber nicht a. Da p eine Primzahl ist, muss $\gcd(a, p) = 1$ sein. Nach Korollar 2.7.7 gibt es x, y mit $1 = ax + py$. Daraus folgt

$$b = abx + pby.$$

Weil p ein Teiler von abx und pby ist, folgt aus Theorem 2.2.3, dass p auch ein Teiler von b ist. □

Korollar 2.11.4. *Wenn eine Primzahl p ein Produkt $\prod_{i=1}^{k} q_i$ von Primzahlen teilt, dann stimmt p mit einer der Primzahlen $q_1, q_2, \ldots, q_k$ überein.*

Beweis. Wir führen den Beweis durch Induktion über die Anzahl k. Ist $k = 1$, so ist p ein Teiler von q_1, der größer als 1 ist, und stimmt daher mit q_1 überein. Ist $k > 1$, dann ist p ein Teiler von $q_1(q_2 \cdots q_k)$. Nach Lemma 2.11.3 ist p ein Teiler von q_1 oder von $q_2 \cdots q_k$. Da beide Produkte weniger als k Faktoren haben, folgt die Behauptung des Korollars aus der Induktionsannahme. □

Jetzt wird der Hauptsatz der elementaren Zahlentheorie bewiesen.

Theorem 2.11.5. *Jede natürliche Zahl $a > 1$ kann als Produkt von Primzahlen geschrieben werden. Bis auf die Reihenfolge sind die Faktoren in diesem Produkt eindeutig bestimmt.*

Beweis. Wir beweisen den Satz durch Induktion über a. Für $a = 2$ stimmt der Satz. Angenommen, $a > 2$. Nach Theorem 2.11.2 hat a einen Primteiler p. Ist $a/p = 1$, so ist $a = p$ und der Satz ist bewiesen. Sei also $a/p > 1$. Da nach Induktionsvoraussetzung a/p Produkt von Primzahlen ist, kann auch a als Produkt von Primzahlen geschrieben werden. Damit ist die Existenz der Primfaktorzerlegung nachgewiesen. Es fehlt noch die Eindeutigkeit. Seien $a = p_1 \cdots p_k$ und $a = q_1 \cdots q_l$ Primfaktorzerlegungen von a. Nach Korollar 2.11.4 stimmt p_1 mit einer der Primzahlen $q_1, \ldots, q_k$ überein. Durch Umnummerierung erreicht man, dass $p_1 = q_1$ ist. Nach Induktionsannahme ist aber die Primfaktorzerlegung von $a/p_1 = a/q_1$ eindeutig. Also gilt $k = l$ und nach entsprechender Umnummerierung $q_i = p_i$ für $1 \leq i \leq k$. □

Die *Primfaktorzerlegung* einer natürlichen Zahl a ist die Darstellung der Zahl als Produkt von Primfaktoren. Effiziente Algorithmen, die die Primfaktorzerlegung einer natürlichen Zahl berechnen, sind nicht bekannt. Dies ist die Grundlage der Sicherheit des RSA-Verschlüsselungsverfahrens und auch anderer wichtiger kryptographischer Algorithmen. Es ist aber auch kein Beweis bekannt, der zeigt, dass das Faktorisierungsproblem schwer ist. Es ist daher möglich, dass es effiziente Faktorisierungsverfahren gibt, und dass die auch schon bald gefunden werden. Dann sind die entsprechenden kryptographischen Verfahren unsicher und müssen durch andere ersetzt werden.

Beispiel 2.11.6. Der französische Jurist Pierre de Fermat (1601 bis 1665) glaubte, dass die nach ihm benannten *Fermat-Zahlen*

$$F_i = 2^{2^i} + 1$$

sämtlich Primzahlen seien. Tatsächlich sind $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$ und $F_4 = 65537$ Primzahlen. Aber 1732 fand Euler heraus, dass $F_5 = 641 * 6700417$ zusammengesetzt ist. Die angegebene Faktorisierung ist auch die Primfaktorzerlegung der fünften Fermat-Zahl. Auch F_6, F_7, F_8 und

F_9 sind zusammengesetzt. Die Faktorisierung von F_6 wurde 1880 von Landry und Le Lasseur gefunden, die von F_7 erst 1970 von Brillhart und Morrison. Die Faktorisierung von F_8 wurde 1980 von Brent und Pollard gefunden und die von F_9 1990 von Lenstra, Lenstra, Manasse und Pollard. Einerseits sieht man an diesen Daten, wie schwierig das Faktorisierungsproblem ist; immerhin hat es bis 1970 gedauert, bis die 39-stellige Fermat-Zahl F_7 zerlegt war. Andererseits ist die enorme Weiterentwicklung daran zu erkennen, dass nur 20 Jahre später die 155-stellige Fermat-Zahl F_9 faktorisiert wurde.

2.12 Übungen

Übung 2.12.1. Sei α eine reelle Zahl. Zeigen Sie, dass $\lfloor \alpha \rfloor$ die eindeutig bestimmte ganze Zahl z ist mit $0 \leq \alpha - z < 1$.

Übung 2.12.2. Bestimmen Sie die Anzahl der Teiler von 2^n, $n \in \mathbb{Z}_{\geq 0}$.

Übung 2.12.3. Bestimmen Sie alle Teiler von 195.

Übung 2.12.4. Beweisen Sie folgende Modifikation der Division mit Rest: Sind a und b ganze Zahlen, $b > 0$, dann gibt es eindeutig bestimmte ganze Zahlen q und r mit der Eigenschaft, dass $a = qb + r$ und $-b/2 < r \leq b/2$ gilt. Schreiben Sie ein Programm, das den Rest r berechnet.

Übung 2.12.5. Berechnen Sie $1243 \bmod 45$ und $-1243 \bmod 45$.

Übung 2.12.6. Finden Sie eine ganze Zahl a mit $a \bmod 2 = 1$, $a \bmod 3 = 1$, und $a \bmod 5 = 1$.

Übung 2.12.7. Sei m eine natürliche Zahl und seien a, b ganze Zahlen. Zeigen Sie: Genau dann gilt $a \bmod m = b \bmod m$, wenn m die Differenz $b - a$ teilt.

Übung 2.12.8. Berechnen Sie die Binärdarstellung und die Hexadezimaldarstellung von 225.

Übung 2.12.9. Bestimmen Sie die binäre Länge der n-ten Fermat-Zahl $2^{2^n} + 1$, $n \in \mathbb{Z}_{\geq 0}$.

Übung 2.12.10. Schreiben Sie ein Programm, das für gegebenes $g \geq 2$ die g-adische Darstellung einer natürlichen Zahl n berechnet.

Übung 2.12.11. Sei $f(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_0$ ein Polynom mit reellen Koeffizienten, wobei $a_d > 0$ ist. Zeigen Sie, dass $f(n) = O(n^d)$ ist.

Übung 2.12.12. Sei $k \in \mathbb{N}$ und $X \subset \mathbb{N}^k$. Angenommen, $f, g, F, G : X \to \mathbb{R}_{\geq 0}$ mit $f = O(F)$ und $g = O(G)$. Zeigen Sie, dass $f \pm g = O(F + G)$ und $fg = O(FG)$ gilt.

Übung 2.12.13. Seien $a_1, \ldots, a_k$ ganze Zahlen. Beweisen Sie folgende Behauptungen.

1. Es ist $\gcd(a_1, \ldots, a_k) = \gcd(a_1, \gcd(a_2, \ldots, a_k))$.
2. Es ist $a_1\mathbb{Z} + \ldots + a_k\mathbb{Z} = \gcd(a_1, \ldots, a_k)\mathbb{Z}$.
3. Die Gleichung $x_1a_1 + \ldots + x_ka_k = n$ ist genau dann durch ganze Zahlen $x_1, \ldots, x_k$ lösbar, wenn $\gcd(a_1, \ldots, a_k)$ ein Teiler von n ist.
4. Es gibt ganze Zahlen $x_1, \ldots, x_k$ mit $a_1x_1 + \ldots + a_kx_k = \gcd(a_1, \ldots, a_k)$.
5. Der größte gemeinsame Teiler von $a_1, \ldots, a_k$ ist der eindeutig bestimmte nicht negative gemeinsame Teiler von $a_1, \ldots, a_k$, der von allen gemeinsamen Teilern von $a_1, \ldots, a_k$ geteilt wird.

Übung 2.12.14. Beweisen Sie, dass der eulidische Algorithmus auch funktioniert, wenn die Division mit Rest so modifiziert ist wie in Übung 2.12.4.

Übung 2.12.15. Berechnen Sie $\gcd(235, 124)$ samt seiner Darstellung mit dem erweiterten euklidischen Algorithmus.

Übung 2.12.16. Benutzen Sie den modifizierten euklidischen Algorithmus aus Übung 2.12.14, um $\gcd(235, 124)$ einschließlich Darstellung zu berechnen. Vergleichen Sie diese Berechnung mit der Berechnung aus Beispiel 2.12.15.

Übung 2.12.17. Beweisen Sie Lemma 2.10.4.

Übung 2.12.18. Sei $a > b > 0$. Beweisen Sie, dass der modifizierte euklidische Algorithmus aus Beispiel 2.12.14 $O(\log b)$ Iterationen braucht, um $\gcd(a, b)$ zu berechnen.

Übung 2.12.19. Seien a, b positive ganze Zahlen. Man zeige, dass die Anzahl der Iterationen und die Folge der Quotienten im euklidischen Algorithmus nur vom Quotienten a/b abhängt.

Übung 2.12.20. Finden Sie eine Folge $(a_i)_{i\geq 1}$ positiver ganzer Zahlen mit der Eigenschaft, dass der euklidische Algorithmus genau i Iterationen benötigt, um $\gcd(a_{i+1}, a_i)$ zu berechnen.

Übung 2.12.21. Zeigen Sie, dass aus $\gcd(a, m) = 1$ und $\gcd(b, m) = 1$ folgt, dass $\gcd(ab, m) = 1$ ist.

Übung 2.12.22. Berechnen Sie die Primfaktorzerlegung von 37800.

Übung 2.12.23. Zeigen Sie, dass jede zusammengesetzte Zahl $n > 1$ einen Primteiler $p \leq \sqrt{n}$ hat.

Übung 2.12.24. Das *Sieb des Eratosthenes* bestimmt alle Primzahlen unter einer gegebenen Schranke C. Es funktioniert so: Schreibe die Liste $2, 3, 4, 5, \ldots, \lfloor C \rfloor$ von ganzen Zahlen auf. Dann iteriere folgenden Prozeß für $i = 2, 3, \ldots, \lfloor \sqrt{C} \rfloor$. Wenn i noch in der Liste ist, lösche alle echten Vielfachen $2i, 3i, 4i, \ldots$ von i aus der Liste. Die Zahlen, die in der Liste bleiben, sind die gesuchten Primzahlen. Schreiben Sie ein Programm, dass diese Idee implementiert.