Grundlagen

1

In diesem Kapitel präsentieren wir wichtige Grundlagen: Ganze Zahlen, elementare Wahrscheinlichkeitstheorie, elementare Komplexitätstheorie sowie grundlegende Algorithmen für ganze Zahlen.

1.1 Ganze Zahlen

Ganze Zahlen sind fundamental. In diesem Abschnitt beschreiben wir wichtige Eigenschaften ganzer Zahlen. Wir diskutieren insbesondere den Begriff der Teilbarkeit und des größten gemeinsamen Teilers und stellen verschiedene Möglichkeiten vor, ganze Zahlen darzustellen.

1.1.1 Grundbegriffe und Eigenschaften

Dieser Abschnitt erinnert die Leserinnen und Leser an wichtige Grundbegriffe und Eigenschaften von Zahlen.

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 Ergebnis heißt *Summe* bzw. *Produkt* der beiden Zahlen. Summen und Produkte über viele reelle Zahlen schreibt man so wie in den nächsten beiden Beispielen.

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n. \tag{1.1}$$

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-39775-2_1

$$\prod_{i=1}^{n} i = 1 * 2 * 3 * \ldots * n. \tag{1.2}$$

Reelle Zahlen können auch potenziert werden. Für eine relle Zahl α und eine natürliche Zahl n bezeichnet α^n die n-te *Potenz* von α, also das Ergebnis der Multiplikation von α n-mal mit sich selbst. So ist zum Beispiel $2^3 = 2 * 2 * 2 = 8$ die dritte Potenz von 2. Außerdem setzt man $\alpha^0 = 1$.

Wir werden folgende grundlegende Regeln benutzen: 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. Zum Beispiel ist $\sqrt{2}$ kleiner als 2 aber größer als 1. Wenn zwei reelle Zahlen α und β gleich sind, schreibt man $\alpha = \beta$. Andernfalls gilt $\alpha \neq \beta$. 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 $\alpha > 0$ und $\beta > 0$, dann folgt $\alpha\beta > 0$.

Sind α und β reelle Zahlen und ist $\beta \neq 0$, dann kann man α durch β dividieren. Das Ergebnis $\alpha/\beta = \frac{\alpha}{\beta}$ heißt *Quotient* mit *Zähler* α und *Nenner* β. Außerdem schreibt man $\beta^{-n} = 1/\beta^n$ für jede natürliche Zahl n.

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 oder gleich γ sind. Man sagt dann, dass M nach unten durch γ beschränkt ist und nennt γ *untere Schranke* für M. Eine untere Schranke für die Menge der natürlichen Zahlen ist z. B. die Zahl 0. Nach unten beschränkte Mengen haben nicht nur eine sondern unendlich viele untere Schranken. So sind zum Beispiel alle negativen ganzen Zahlen untere Schranken für $\mathbb{N}$. Aber viele Mengen reeller Zahlen sind nicht nach unten beschränkt, zum Beispiel die Menge der geraden ganzen Zahlen. Eine wichtige Eigenschaft der ganzen Zahlen ist, dass jede nach unten beschränkte Menge ganzer Zahlen ein kleinstes Element besitzt. Zum Beispiel ist die kleinste natürliche Zahl 1. Entsprechend definiert man *nach oben beschränkte* Mengen reeller Zahlen und *obere Schranken*. Außerdem hat jede nach oben beschränkte Menge ganzer Zahlen ein größtes Element.

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 höchstens so groß wie α ist. Diese Zahl existiert, weil die Menge $\{b \in \mathbb{Z} : b \leq \alpha\}$ aller ganzen Zahlen, die höchstens so groß wie α sind, nach oben beschränkt ist.

Beispiel 1.1 Es ist $\lfloor 3.43 \rfloor = 3$ und $\lfloor -3.43 \rfloor = -4$.

1.1.2 Vollständige Induktion

Schließlich benötigen wir noch das Prinzip der *vollständigen Induktion*. Es lautet so: Ist eine Aussage, in der eine unbestimmte natürliche Zahl n vorkommt, richtig für $n = 1$ und folgt aus ihrer Richtigkeit für eine natürliche Zahl n die Richtigkeit für $n + 1$, so ist die Aussage für jede natürliche Zahl richtig. Dieses Prinzip entspricht der Intuition und ist eine der fundamentalen Beweismethoden in der Mathematik. In abgewandelter Form wird das Prinzip der vollständigen Induktion auch verwendet, um die Korrektheit von Algorithmen zu beweisen.

Beispiel 1.2 Wir beweisen mit dem Prinzip der vollständigen Induktion, dass für jede reelle Zahl $\alpha > 1$ und jede natürliche Zahl n die folgende Gleichung gilt:

$$\sum_{i=1}^{n} \alpha^{i-1} = \frac{\alpha^n - 1}{\alpha - 1} \tag{1.3}$$

Wie im Prinzip der vollständigen Induktion erläutert, wird die Aussage zunächst für $n = 1$ bewiesen. Dies nennt man *Induktionsverankerung*. Sie ist in unserem Beispiel leicht. Einerseits gilt nämlich

$$\sum_{i=1}^{1} \alpha^{i-1} = 1. \tag{1.4}$$

Andererseits hat man

$$\frac{\alpha^1 - 1}{\alpha - 1} = \frac{\alpha - 1}{\alpha - 1} = 1. \tag{1.5}$$

Jetzt kommt der *Induktionsschritt*. Dabei wird vorausgesetzt, dass die Aussage (1.3) für eine natürliche Zahl n gilt. Daraus wird gefolgert, dass die Aussage auch für $n + 1$ richtig ist. Wir schreiben also unsere Behauptung für $n + 1$ hin und formen sie so um, dass man die als gültig vorausgesetzte Aussage für n verwenden kann. Das geht so:

$$\sum_{i=1}^{n+1} \alpha^{i-1} = \sum_{i=1}^{n} \alpha^{i-1} + \alpha^n \tag{1.6}$$

Auf der rechten Seite von (1.6) steht als Summand die linke Seite von (1.3). Sie wird durch die rechte Seite der Gleichung (1.3) ersetzt, die als wahr vorausgesetzt wurde. Es ergibt sich

$$\sum_{i=1}^{n+1} \alpha^{i-1} = \frac{\alpha^n - 1}{\alpha - 1} + \alpha^n = \frac{\alpha^n - 1 + \alpha^n(\alpha - 1)}{\alpha - 1} = \frac{\alpha^{n+1} - 1}{\alpha - 1}, \tag{1.7}$$

wie behauptet.

Das Prinzip der vollständigen Induktion hat Varianten. So kann die Induktionsverankerung bei jeder ganzen Zahl m beginnen. Die Aussage wird dann für alle ganzen Zahlen $n \geq m$ bewiesen. Im Induktionsschritt kann man auch voraussetzen, dass die zu beweisende Aussage für alle ganzen Zahlen k mit $m \leq k \leq n$ gilt. Das kann den Beweis vereinfachen, weil mehr Information benutzt werden darf.

1.1.3 Konvention

Damit wir nicht immer schreiben müssen „sei n eine ganze Zahl“, bezeichnen im Folgenden kleine lateinische Buchstaben immer ganze Zahlen, ohne dass wir das extra erwähnen.

1.1.4 Teilbarkeit

In diesem Abschnitt diskutieren wir den wichtigen Begriff der Teilbarkeit.

Definition 1.1 Man sagt *a teilt n*, wenn es ein 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 1.3 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 1.1

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 und 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 $a = 0$ ist, dann gilt $b = 0$ und umgekehrt. Wenn $a \neq 0$ und $b \neq 0$ gilt, dann folgt aus 4., dass $|a| \leq |b|$ und $|b| \leq |a|$ ist, also $|a| = |b|$ gilt. □

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

Theorem 1.2 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 1.2 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 1.4 Wenn $a = 133$ und $b = 21$ ist, dann liefert die Division mit Rest $q = 6$ und $r = 7$, d. h. $133 \bmod 21 = 7$. Entsprechend gilt $-50 \bmod 8 = 6$.

1.1.5 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 *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 1.5 Es ist $\log_2 8 = 3$, weil $2^3 = 8$ ist. Ferner ist $\log_8 8 = 1$, weil $8^1 = 8$ ist.

Beispiel 1.6 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 1.3 *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*

$$(a_1, \ldots, a_k) \in \{0, \ldots, g-1\}^k \tag{1.8}$$

mit $a_1 \neq 0$ und

$$a = \sum_{i=1}^{k} a_i g^{k-i}. \tag{1.9}$$

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

$$a_i = \left\lfloor \frac{a - \sum_{j=1}^{i-1} a_j g^{k-j}}{g^{k-i}} \right\rfloor \quad 1 \leq i \leq k. \tag{1.10}$$

Beweis Sei a eine natürliche Zahl. Wenn es eine Darstellung von a wie in (1.9) 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 jetzt die Eindeutigkeit der Folge $(a_1, \ldots, a_k)$ durch Induktion über k.

Für $k = 1$ muss $a_1 = a$ sein. Sei $k > 1$. Wenn es eine Darstellung wie in (1.9) 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 $a_1 = \lfloor a/g^{k-1} \rfloor$, also eindeutig bestimmt. Setze $a' = a - a_1 g^{k-1} = \sum_{i=2}^{k} a_i g^{k-i}$. Dann gilt $0 \leq a' < g^{k-1}$. Entweder ist $a' = 0$. Dann ist $a_i = 0, 2 \leq i \leq n$. Oder wir verwenden die Darstellung $a' = \sum_{i=2}^{k} a_i g^{k-i}$, die nach Induktionsannahme existiert und eindeutig ist.

Schließlich beweisen wir die Existenz durch Induktion über a. Wir setzen $a_1 = \lfloor a/g^{k-1} \rfloor$. Ist $a_1 = a$, sind wir fertig. Andernfalls nehmen wir die anderen Koeffizienten aus der Darstellung von $a' = a - a_1 g^{k-1}$, gegebenenfalls mit führenden Nullen. □

Definition 1.2 Die Folge $(a_1, \ldots, a_k)$ aus Theorem 1.3 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 $a_1 a_2 \ldots a_k$ für ein solche Entwicklung.

Beispiel 1.7 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 1.3 enthält ein Verfahren zur Berechnung der g-adischen Entwicklung einer natürlichen Zahl. Dieses Verfahren wird im nächsten Beispiel angewandt.

Beispiel 1.8 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 1.9 Betrachte die Hexadezimalzahl $n = 6EF$. Die auf Länge 4 normierten Binärentwicklungen der Ziffern sind $6 = 0110$, $E = 1110$, $F = 1111$. Daher ist 11011101111 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 plus 1. Das zusätzliche Bit kodiert das Vorzeichen. Die binäre Länge einer ganzen Zahl a wird auch mit $\text{size}(a)$ oder $\text{size}\, a$ bezeichnet.

1.1.6 Größter gemeinsamer Teiler

Wir führen den größten gemeinsamen Teiler zweier ganzer Zahlen ein.

Definition 1.3 Ein *gemeinsamer Teiler* von a und b ist eine ganze Zahl c, die sowohl a als auch b teilt.

Theorem 1.4 *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 1.1 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 1.10 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 1.11 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 1.5 *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 1.2 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 $a, b \in I$ gilt, 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 1.1, dass $|d| \leq g$ gilt. Damit ist g der größte gemeinsame Teiler von a und b. □

Beispiel 1.12 Das Ergebnis von Beispiel 1.11 hätte man direkt aus Theorem 1.5 folgern können. Es ist nämlich $\gcd(3,4) = 1$ und daher $3\mathbb{Z} + 4\mathbb{Z} = 1\mathbb{Z} = \mathbb{Z}$.

Theorem 1.5 hat einige wichtige Folgerungen.

Korollar 1.1 *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 1.5 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 1.5 gehört n also auch zu $a\mathbb{Z} + b\mathbb{Z}$. Es gibt daher ganze Zahlen x und y mit $n = ax + by$. □

Korollar 1.1 sagt uns, dass die Gleichung

$$3x + 4y = 123$$

eine Lösung hat, weil $\gcd(3,4) = 1$ ist und 123 ein Vielfaches von 1 ist. Aus Korrollar 1.1 folgt, dass insbesondere der größte gemeinsame Teiler zweier ganzer Zahlen als Linearkombination dieser Zahlen dargestellt werden kann.

Korollar 1.2 *Für alle ganzen Zahlen a und b gibt es 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 1.1. □

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 1.3 *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 1.2 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 1.1 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 werden kann und wie ganze Zahlen x und y bestimmt werden können, 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 Abschn. 1.6.2 erläutert wird.

1.1.7 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 1.4 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 1.6 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 1.1 *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 1.2 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 1.1, dass p auch ein Teiler von b ist. □

Korollar 1.4 *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 1.1 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 1.7 *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 1.6 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 1.4 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. Normalerweise schreiben wir die Zerlegung von a so:

$$a = \prod_{p \in \mathbb{P}, p|a} p^{e(p)}. \tag{1.11}$$

Hierbei sind die Exponenten $e(p)$ natürliche Zahlen. Effiziente Algorithmen, die die Primfaktorzerlegung einer natürlichen Zahl berechnen, sind nicht bekannt. Dies ist die Sicherheitsgrundlage des RSA-Verschlüsselungsverfahrens, das in Abschn. 8.3 behandelt wird,

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. Die besten heute bekannten Faktorisierungsalgorithmen werden in Abschn. 9 beschrieben.

Beispiel 1.13 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 berechnet 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.

1.2 Wahrscheinlichkeit

Viele kryptographische Berechnungsverfahren sind probabilistisch, das heißt dass ihre Berechnung vom Zufall abhängt. Das gilt zum Beispiel für die Erzeugung geheimer Schlüssel, für Verschlüsselungsalgorithmen aber auch für Verfahren, die Verschlüsselungsverfahren brechen. Darum erläutern wir in diesem Abschnitt die notwendigen Grundlagen aus der Wahrscheinlichkeitstheorie.

1.2.1 Grundbegriffe

Sei S eine endliche, nicht leere Menge. Wir nennen sie *Ergebnismenge*. Ihre Elemente heißen *Ergebnisse* oder *Elementarereignisse*.

Beispiel 1.14 Wenn man eine Münze wirft, erhält man entweder Zahl oder Kopf. Die entsprechende Ergebnismenge bezeichnen wir mit $S = \{Z, K\}$.

Wenn man würfelt, erhält man eine der Zahlen $1, 2, 3, 4, 5, 6$. Die Ergebnismenge ist dann $S = \{1, 2, 3, 4, 5, 6\}$.

Ein *Ereignis* ist eine Teilmenge der Ergebnismenge S. Das *sichere Ereignis* ist die Menge S selbst. Das *leere Ereignis* ist $\emptyset$. Zwei Ereignisse A und B *schließen sich gegenseitig aus*, wenn ihr Durchschnitt leer ist. Die Menge aller Ereignisse ist also die *Potenzmenge* $P(S)$ von S, also die Menge aller Teilmengen von S.

Beispiel 1.15 Beim Würfeln kann das Ereignis, eine gerade Zahl zu würfeln, auftreten. Formal ist dieses Ereignis die Teilmenge $\{2, 4, 6\}$ der Ergebnismenge $\{1, 2, 3, 4, 5, 6\}$. Es schließt das Ereignis $\{1, 3, 5\}$, eine ungerade Zahl zu würfeln, aus.

Eine *Wahrscheinlichkeitsverteilung* auf S ist eine Abbildung Pr, die jedem Ereignis eine reelle Zahl zuordnet, also

$$\Pr : P(S) \to \mathbb{R},$$

und die folgende Eigenschaften erfüllt:

1. $\Pr(A) \geq 0$ für alle Ereignisse A,
2. $\Pr(S) = 1$,
3. $\Pr(A \cup B) = \Pr(A) + \Pr(B)$ für zwei Ereignisse A und B, die sich gegenseitig ausschließen.

Ist A ein Ereignis, so heißt $\Pr(A)$ *Wahrscheinlichkeit* dieses Ereignisses. Die Wahrscheinlichkeit eines Elementarereignisses $a \in S$ ist als $\Pr(a) = \Pr(\{a\})$ definiert.

Aus den Eigenschaften von Wahrscheinlichkeitsverteilungen folgen einige weitere Aussagen:

1. $\Pr(\emptyset) = 0$;
2. aus $A \subset B$ folgt $\Pr(A) \leq \Pr(B)$;
3. $0 \leq \Pr(A) \leq 1$ für alle $A \in P(S)$;
4. $\Pr(S \setminus A) = 1 - \Pr(A)$;
5. sind $A_1, \ldots, A_n$ Ereignisse, die sich paarweise ausschließen, dann gilt $\Pr(\cup_{i=1}^{n} A_i) = \sum_{i=1}^{n} \Pr(A_i)$.

Weil S eine endliche Menge ist, genügt es, eine Wahrscheinlichkeitsverteilung auf den Elementarereignissen zu definieren, denn es gilt $\Pr(A) = \sum_{a \in A} \Pr(a)$ für jedes Ereignis A.

Beispiel 1.16 Eine Wahrscheinlichkeitsverteilung auf der Menge $\{1, 2, 3, 4, 5, 6\}$, die einem Würfelexperiment entspricht, ordnet jedem Elementarereignis die Wahrscheinlichkeit $1/6$ zu. Die Wahrscheinlichkeit für das Ereignis, eine gerade Zahl zu würfeln, ist dann $\Pr(\{2, 4, 6\}) = \Pr(2) + \Pr(4) + \Pr(6) = 1/6 + 1/6 + 1/6 = 1/2$.

Die Wahrscheinlichkeitsverteilung, die jedem Elementarereignis $a \in S$ die Wahrscheinlichkeit $\Pr(a) = 1/|S|$ zuordnet, heißt *Gleichverteilung*.

1.2.2 Bedingte Wahrscheinlichkeit

Sei S eine Ergebnismenge und sei Pr eine Wahrscheinlichkeitsverteilung auf S. Wir erläutern den Begriff der *bedingten Wahrscheinlichkeit* erst an einem Beispiel.

Beispiel 1.17 Wir modellieren Würfeln mit einem Würfel. Die Ergebnismenge ist $\{1, 2, 3, 4, 5, 6\}$ und Pr ordnet jedem Elementarereignis die Wahrscheinlichkeit $1/6$ zu. Angenommen, wir wissen, dass Klaus eine der Zahlen $4, 5, 6$ gewürfelt hat. Wir wissen also, dass das Ereignis $B = \{4, 5, 6\}$ eingetreten ist. Unter dieser Voraussetzung möchten wir die Wahrscheinlichkeit dafür ermitteln, dass Klaus eine gerade Zahl gewürfelt hat. Jedes Elementarereignis aus B hat die Wahrscheinlichkeit $1/3$. Da zwei Zahlen in B gerade sind, ist die Wahrscheinlichkeit dafür, dass Klaus eine gerade Zahl gewürfelt hat, $2/3$.

Jetzt kann die bedingte Wahrscheinlichkeit formal definiert werden.

Definition 1.5 Seien A und B Ereignisse und $\Pr(B) > 0$. Die Wahrscheinlichkeit für A *unter der Bedingung* B ist definiert als

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}.$$

Diese Definition ist so zu verstehen: Bekannt ist, dass das Ereignis B eingetreten ist. Damit ist B die neue Ergebnismenge. Die relativen Wahrscheinlichkeiten der Ereignisse in dieser Ergebnismenge haben sich nicht geändert. Also werden die Wahrscheinlichkeiten von Ereignissen A in B entsprechend normiert: $\Pr(A|B) = \Pr(A)/\Pr(B)$. Dann ist tatsächlich $\Pr(B|B) = 1$ und für zwei Ereignisse A und A' in B gilt $\Pr(A)/\Pr(A') = \Pr(A|B)/\Pr(A'|B)$. Die bedingte Wahrscheinlichkeit von beliebigen Ereignissen A in S ist dann $\Pr(A|B) = \Pr(A \cap B|B)$, weil nur Elementarereignisse aus B eintreten können.

Zwei Ereignisse A und B heißen *unabhängig*, wenn

$$\Pr(A \cap B) = \Pr(A)\Pr(B)$$

gilt. Diese Bedingung ist äquivalent zu

$$\Pr(A|B) = \Pr(A).$$

Beispiel 1.18 Wenn man zwei Münzen wirft, ist das Ereignis, mit der ersten Münze „Zahl“ zu werfen, unabhängig von dem Ereignis, mit der zweiten Münze „Zahl“ zu werfen. Die Wahrscheinlichkeit dafür, mit beiden Münzen „Zahl“ zu werfen, ist nämlich $1/4$. Die Wahrscheinlichkeit dafür, mit der ersten Münze „Zahl“ zu werfen, ist $1/2$, genauso wie die Wahrscheinlichkeit, mit der zweiten Münze „Zahl“ zu werfen. Wie in der Definition gefordert, ist $1/4 = (1/2)(1/2)$.

Wenn man die Münzen zusammenlötet, so dass sie entweder beide „Zahl“ oder beide „Kopf“ zeigen, sind die Wahrscheinlichkeiten nicht mehr unabhängig. Die Wahrscheinlichkeit dafür, mit beiden Münzen „Zahl“ zu werfen, ist dann nämlich $1/2$, genau wie die Wahrscheinlichkeit dafür, mit der ersten Münze „Zahl“ zu werfen und die Wahrscheinlichkeit dafür, mit der zweiten Münze Zahl zu werfen.

Wir formulieren und beweisen den Satz von Bayes.

Theorem 1.8 *Sind A und B Ereignisse mit $\Pr(A) > 0$ und $\Pr(B) > 0$, so gilt*

$$\Pr(B)\Pr(A|B) = \Pr(A)\Pr(B|A).$$

Beweis Nach Definition gilt

$$\Pr(A|B)\Pr(B) = \Pr(A \cap B) \tag{1.12}$$

und

$$\Pr(B|A)\Pr(A) = \Pr(A \cap B). \tag{1.13}$$

Daraus folgt die Behauptung unmittelbar. □

1.3 Zufallsvariablen

Jetzt führen wir *Zufallsvariablen* ein.

Definition 1.6 Sei S eine endliche Ergebnismenge und sei $\Pr : P(S) \to \mathbb{R}$ eine Wahrscheinlichkeitsverteilung auf S. Eine *Zufallsvariable* auf S ist eine Funktion

$$X : S \to \mathbb{R}.$$

Der *Erwartungswert* von X ist

$$E[X] = \sum_{s \in S} \Pr(s)X(s). \tag{1.14}$$

Beispiel 1.19 Sei $S = \{(i, j) : 0 \leq i, j \leq 6\}$ die Menge aller Ergebnisse eines Wurfs zweier Würfel. Die Funktion

$$X : S \to \mathbb{R}, \quad (i, j) \mapsto i + j$$

ist eine Zufallsvariable auf S. Sie ordnet jedem Münzwurf die Summe der beiden gewürfelten Werte zu. Wir berechnen ihren Erwartungswert. Wir sehen, dass $2 = 1 + 1$,

$3 = 1+2 = 2+1, 4 = 1+3 = 3+1 = 2+2, 5 = 1+4 = 4+1 = 2+3 = 3+2,$
$6 = 1+5 = 5+1 = 2+4 = 4+2 = 3+3, 7 = 1+6 = 6+1 = 2+5 = 5+2 = 3+4 =$
$4+3, 8 = 2+6 = 6+2 = 3+5 = 5+3 = 4+4, 9 = 3+6 = 6+3 = 5+4 = 4+5,$
$10 = 4+6 = 6+4 = 5+5, 11 = 5+6 = 6+5$ und $12 = 6+6$ ist. Damit ist $E[X] =$
$(2+3*2+4*3+5*4+6*5+7*6+8*5+9*4+10*3+11*2+12)/36 = 7.$

Der Erwartungswert einer Zufallsvariablen ist der Mittelwert, den die Zufallsvariable annimmt.

1.3.1 Geburtstagsparadox

Ein gutes Beispiel für wahrscheinlichkeitstheoretische Überlegungen ist das Geburtstagsparadox. Ausgangspunkt ist die folgende Frage: Wieviele Personen müssen in einem Raum sein, damit die Wahrscheinlichkeit mindestens $1/2$ ist, dass wenigstens zwei von ihnen am gleichen Tag Geburtstag haben? Es sind erstaunlich wenige, jedenfalls deutlich weniger als 365, wie wir nun zeigen werden.

Wir machen eine etwas allgemeinere Analyse. Angenommen, es gibt n verschiedene „Geburtstage“. Im Raum sind k Personen. Ein Elementarereignis ist ein Tupel $(g_1, \ldots, g_k) \in \{1, 2, \ldots, n\}^k$. Tritt es ein, so hat die i-te Person den Geburtstag g_i, $1 \leq i \leq k$. Es gibt also n^k Elementarereignisse. Wir nehmen an, dass alle Elementarereignisse gleich wahrscheinlich sind. Jedes Elementarereignis hat also die Wahrscheinlichkeit $1/n^k$.

Wir möchten die Wahrscheinlichkeit dafür berechnen, dass wenigstens zwei Personen am gleichen Tag Geburtstag haben. Bezeichne diese Wahrscheinlichkeit mit p. Dann ist $q = 1 - p$ die Wahrscheinlichkeit dafür, dass alle Personen verschiedene Geburtstage haben. Diese Wahrscheinlichkeit q rechnen wir aus und rekonstruieren daraus $p = 1 - q$. Das Ereignis E, das uns interessiert, ist die Menge aller Vektoren $(g_1, \ldots, g_k) \in \{1, 2, \ldots, n\}^k$, deren sämtliche Koordinaten verschieden sind. Alle Elementarereignisse haben die gleiche Wahrscheinlichkeit $1/n^k$. Die Wahrscheinlichkeit für E ist also die Anzahl der Elemente in E multipliziert mit $1/n^k$. Die Anzahl der Vektoren in $\{1, \ldots, n\}^k$ mit lauter verschiedenen Koordinaten bestimmt sich so: Auf der ersten Position können n Zahlen stehen. Liegt die erste Position fest, so können auf der zweiten Position noch $n-1$ Zahlen stehen usw. Es gilt also

$$|E| = \prod_{i=0}^{k-1} (n-i).$$

Die gesuchte Wahrscheinlichkeit ist

$$q = \frac{1}{n^k} \prod_{i=0}^{k-1} (n-i) = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right). \tag{1.15}$$

Nun gilt aber $1 + x \leq e^x$ für alle reellen Zahlen. Aus (1.15) folgt daher

$$q \leq \prod_{i=1}^{k-1} e^{-i/n} = e^{-\sum_{i=1}^{k-1} i/n} = e^{-k(k-1)/(2n)}. \tag{1.16}$$

Ist

$$k \geq (1 + \sqrt{1 + 8n \log 2})/2 \tag{1.17}$$

so folgt aus (1.16), dass $q \leq 1/2$ ist. Dann ist die Wahrscheinlichkeit $p = 1-q$ dafür, dass zwei Personen im Raum am gleichen Tag Geburtstag haben mindestens $1/2$. Für $n = 365$ genügt $k = 23$, damit $q \leq 1/2$ ist. Sind also 23 Personen im Raum, so haben mit Wahrscheinlichkeit $\geq 1/2$ wenigstens zwei am gleichen Tag Geburtstag. Allgemein genügen etwas mehr als $\sqrt{n}$ viele Personen, damit zwei am gleichen Tag Geburtstag haben.

1.4 Algorithmen

Von Berechnungsverfahren war schon häufiger die Rede. *Algorithmen* sind formalisierte Beschreibungen solcher Berechnungsverfahren. Dieser Abschnitt erläutert den Begriff *Algorithmus* und weitere damit zusammenhängende Begriffe.

1.4.1 Grundbegriffe

Wir beginnen mit einem Beispiel: einem Algorithmus, der das größte Element einer Zahlenfolge findet.

Beispiel 1.20 Die Eingabe von Algorithmus 1.1 findMax ist eine endliche Folge $a = (a_1, a_2, \ldots, a_n)$ ganzer Zahlen. Die Ausgabe ist das größte Element $m = \max_{1 \leq i \leq n} a_i$ der Folge a. Der Algorithmus funktioniert so: Er setzt zuerst m auf a_1. Dann prüft er für $1 < i \leq n$, ob a_i größer als m ist. Wenn ja, ersetzt der Algorithmus m durch a_i. Die *Ausgabe* des Algorithmus ist m.

Algorithmus 1.1 (findMax($a[\,], n$))

$m \leftarrow a[1]$
for $i = 2, \ldots, n$ **do**
  if $a[i] > m$ **then**
    $m \leftarrow a[i]$
  end if
end for
return m

Ein Algorithmus ist also eine Beschreibung, die zeigt, wie aus einer Eingabe eine Ausgabe berechnet wird. Zur Beschreibung des Algorithmus FindMax wurde als Beschreibung Pseudcode verwendet, der aber nicht weiter formalisiert wurde, obwohl das möglich ist, zum Beispiel mit Hilfe von *Turing-Maschinen* (siehe [46]). Die in diesem Abschnitt behandelten Algorithmen werden im Gegensatz zu den in Abschn. 1.4.3 beschriebenen *probabilistischen Algorithmen* auch als *deterministisch* bezeichnet. Dies besagt, dass die Ausgabe des Algorithmus eindeutig von seiner Eingabe bestimmt ist, während dies bei probabilistischen Algorithmen nicht der Fall ist. Zu jedem deterministischen Algorithmus gehört also eine Funktion, die einer Eingabe die entsprechende Ausgabe zuordnet.

1.4.2 Zustandsbehaftete Algorithmen

Ein Algorithmus heißt *zustandsbehaftet*, wenn seine Berechnung von einem *Zustand* abhängt. Nach Abschluss der Berechnung wird der Zustand aktualisiert.

Beispiel 1.21 Ein Algorithmus legt für neue Kunden einer Autoversicherung ein Profil an. Die Eingabe des Algorithmus sind die persönlichen Daten des Kunden wie zum Beispiel Name, Alter, Wohnort und Informationen über das zu versichernde Fahrzeug. Die Ausgabe ist das Profil, das neben den relevanten persönlichen Daten die Kundennummer und den Tarif des Kunden enthält. Der Zustand dieses Algorithmus ist die nächste freie Kundennummer. Nachdem der Algorithmus das Kundenprofil berechnet hat, erhöht der Algorithmus den Zustand um 1.

1.4.3 Probabilitistische Algorithmen

Probabilistische Algorithmen dürfen zu jedem Zeitpunkt mehrfach eine *ideale Münze* werfen und die weitere Berechnung vom Ergebnis des Münzwurfs abhängig machen. Wird eine ideale Münze geworfen, zeigt sie mit exakt gleicher Wahrscheinlichkeit $1/2$ entweder Kopf oder Zahl. Wir stellen den Münzwürf als Prozedur coinToss dar. Wenn coinToss aufgerufen wird, gibt die Prozedur mit gleicher Wahrscheinlichkeit $1/2$ die Werte 1 (entspricht „Kopf“) und 0 (entspricht „Zahl“) aus.

Der probabilitische Algorithmus 1.2 random gibt eine gleichverteilt zufällige Zahl im Intervall $[0, d-1]$, $d \in \mathbb{N}$ zurück. Der Algorithmus verwendet die Prozedur coinToss, um die Ziffern in der Binärentwicklung der Zufallszahl zu wählen.

Der random-Algorithmus ist vom Typ *Monte-Carlo*. Er liefert nur mit einer gewissen Wahrscheinlichkeit das richtige Ergebnis. Probabilistische Algorithmen vom Typ *Las-Vegas* liefern immer das richtige Ergebnis, müssen aber nicht terminieren.

Algorithmus 1.2 (random(d))

$n \leftarrow \lfloor \log_2 d \rfloor + 1$
$r \leftarrow 0$
for $i = 0, \ldots, n-1$ **do**
  $r \leftarrow 2r + \text{coinToss}$
end for
if $r < d$ **then**
  return r
else
  return "failed"
end if

Probabilitische Algorithmen benötigen nicht notwendig eine Eingabe. Wenn wir zum Beispiel im Algorithmus random die Eingabe weglassen und d auf einen festen Wert setzen, so gibt der Algorithmus zufällig und gleichverteilt eine Zahl zwischen 0 und $d-1$ aus.

Die *Erfolgswahrscheinlichkeit* eines Monte-Carlo-Algorithmus ist die Wahrscheinlichkeit dafür, dass der Algorithmus die richtige Antwort gibt. Diese Wahrscheinlichkeit ist tatsächlich wohldefiniert. Da Monte-Carlo-Algorithmen terminieren, sind für jede feste Eingabe nur endlich viele verschiedene Durchläufe des Algorithmus möglich. Bei jedem Durchlauf wird endlich oft eine Münze geworfen. Die Wahrscheinlichkeit für den betreffenden Durchlauf ist $(1/2)^k$, wobei k die Anzahl der Münzwürfe in diesem Durchlauf ist. Die Erfolgswahrscheinlichkeit ist dann die Summe über die Wahrscheinlichkeiten aller erfolgreichen Durchläufe.

Übung 1.20 beweist, dass die Erfolgswahrscheinlichkeit von Algorithmus 1.2 mindestens $1/2$ ist.

1.4.4 Asymptotische 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 *asymptotische Notation* zu verwenden, die Funktionen qualitativ vergleicht. In diesem Abschnitt seien k eine natürliche Zahl, $X, Y \subset \mathbb{N}^k$ und $f : X \to \mathbb{R}$, $g : Y \to \mathbb{R}$ Funktionen.

Definition 1.7

1. Wir schreiben $f = \mathrm{O}(g)$, falls es positive reelle Zahlen B und C gibt, so dass für alle $(n_1, \ldots, n_k) \in X$ mit $n_i > B$, $1 \leq i \leq k$, folgendes gilt:
   (a) $(n_1, \ldots, n_k) \in Y$, d. h. $g(n_1, \ldots, n_k)$ ist definiert,
   (b) $f(n_1, \ldots, n_k) \leq C g(n_1, \ldots, n_k)$.
2. Wir schreiben $f = \Omega(g)$, wenn $g = \mathrm{O}(f)$ ist.
3. Wir schreiben $f = \Theta(g)$, wenn $f = \mathrm{O}(g)$ und $g = \mathrm{O}(f)$ ist.
4. Ist $k = 1$ und $g(n) > 0$ für alle $n \in \mathbb{N}$, dann schreiben wir $f(x) = o(g)$, wenn $\lim_{n\to\infty} f(n)/g(n) = 0$.

Beispiel 1.22 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 1.23 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 $\log n / \log g + 1 < (1/\log g + 1) \log n$.

1.4.5 Laufzeit von deterministischen Algorithmen

In der Kryptographie spielt die Effizienz von Algorithmen eine wichtige Rolle. Um die Effizienz von Algorithmen vergleichen zu können, muss ihre *Laufzeit* quantifiziert werden. Die Laufzeit ist eine Funktion der *Eingabelänge*. Darum definieren wir zuerst die Eingabelänge. Wir nehmen an, dass die Eingabe eines Algorithmus aus Bitsrings besteht. Die Eingabelänge des Algorithmus ist die Summe der Längen dieser Bistrings. Wenn in der Eingabe zum Beispiel ganze Zahlen vorkommen, werden sie durch ihre Binärentwicklung dargestellt. Die Länge der Binärentwicklung einer ganzen Zahl a ist $\mathrm{size}(a) = \lfloor \log |a| \rfloor + 2$ (siehe Abschn. 1.1.5).

Beispiel 1.24 Die Eingabe des findMax-Algorithmus ist eine Folge $a = (a_1, \ldots, a_n)$ ganzer Zahlen und die Länge n dieser Folge. Also ist die Eingabelänge dieses Algorithmus $\sum_{i=1}^{n} \mathrm{size}(a_i) + \mathrm{size}(n)$.

Die Laufzeit eines deterministischen Algorithmus ist eine Funktion, die jeder Eingabelänge die Anzahl der Operationen zuordnet, die der Algorithmus bei Eingaben dieser Länge maximal, also im *schlechtesten Fall*, benötigt. Man nennt dies auch die *Worst-Case-Laufzeit* des Algorithmus. Welche Operationen dabei gezählt werden, ist je nach Laufzeitmodell unterschiedlich. Häufig werden die *Bit-Operationen* gezählt. Die formale Definition der *Bit-Komplexität* ist aufwändig. Dazu kann zum Beispiel der Formalismus der Turing-Maschinen verwendet werden.

Wir geben eine intuitive Beschreibung des Modells einer Turing-Maschine. Detaillierte Beschreibungen sind in [46] zu finden. Angenommen, die Laufzeit eines Algorithmus A soll beschrieben werden. Die Eingabe von A ist eine Bit-Folge. Der Computer, der A ausführt, hat ein *Ein/Ausgabeband*, das zu Anfang die Eingabe und nach Ende der Berechnung die Ausgabe enthält. Außerdem gibt es eine feste Anzahl von Bit-Registern. Der Computer hat außerdem einen *Lese/Schreibkopf*, der an jeder Position des Eingabebandes stehen und dort ein Eingabe-Bit lesen kann. Zu jedem Zeitpunkt ist der Computer in einem *Zustand*: die Register haben einen bestimmten Inhalt, der Lese/Schreibkopf steht an einer bestimmten Position und liest ein bestimmtes Bit. Dann führt der Algorithmus eine Bit-Operation aus. Sie ändert in Abhängigkeit vom Inhalt der Register und des gelesenen Bits auf dem Ein/Ausgabeband, den Inhalt der Register, das gelesene Bit, und die Position des Lesekopfes. Die neue Position des Lese/Schreibkopfes kann die aktuelle Position oder die Position links oder rechts davon sein. Der neue Zustand ist durch den alten Zustand eindeutig bestimmt.

Reale Computer werden von Turing-Maschinen nur unvollkommen modelliert. So können reale Computer Adressen im Speicher in einem Schritt erreichen und lesen. Sie müssen den Lese/Schreibkopf nicht schrittweise zu der gewünschten Position bewegen. Computer, die das können, werden *Random Access Machine (RAM)* genannt. Entsprechende Komplexitätsmodelle findet man in [3] und [4].

Um die Laufzeit von Algorithmen vergleichen zu können, werden Laufzeitklassen verwendet. Wir erläutern einige dieser Klassen, die im Kontext dieses Buches relevant sind. Sei A ein Algorithmus und sei $T : \mathbb{N} \to \mathbb{N}$ seine Laufzeit. Der Algorithmus A wird *polynomiell* genannt, wenn $T(n) = \mathrm{O}(n^v)$ ist für ein $v \in \mathbb{N}$. Die Laufzeit des Algorithmus ist also durch ein Polynom in der Eingabelänge n beschränkt. Wir sagen, dass der Algorithmus *lineare Laufzeit* hat, wenn $T(n) = \mathrm{O}(n)$ ist. Wir gehen davon aus, dass Algorithmen ihre Eingabe lesen müssen. Darum haben alle Algorithmen mindestens lineare Laufzeit. Ist $T(n) = \Theta(n \log n^v)$ für ein $v \in \mathbb{N}$ so heißt der Algorithmus *quasi-linear*. Entsprechend bedeutet *quadratische* Laufzeit, dass $T(n) = \Theta(n^2)$ ist. Weitere Klassifizierungen erlauben die Funktionen

$$L_{u,v} : \mathbb{N} \to \mathbb{N}, n \mapsto e^{v n^u (\log n)^{1-u}} \tag{1.18}$$

wobei $0 \leq u \leq 1$ und $v > 0$ ist. Es gilt

$$L_{0,v}(n) = n^v. \tag{1.19}$$

Ist also $T(n) \leq L_{0,v}(n))$ für ein $v \geq 1$, so ist A polynomiell. Außerdem ist

$$L_{1,v}(n) = (e^n)^v. \tag{1.20}$$

Algorithmus A wird daher *exponentiell* genannt, wenn $T(n) = L_{1,v}(n)$ für ein $v > 0$ ist. Schließlich heißt A *subexponentiell*, wenn $T(n) = L_{u,v}(n)$, wobei $0 < u < 1$ und $v > 0$ ist.

Beispiele für Algorithmen mit solchen Laufzeiten werden in Kap. 9 und Kap. 10 behandelt.

1.4.6 Laufzeit von probabilistischen Algorithmen

Wir behandeln nun die Laufzeit probabilitischer Monte-Carlo-Algorithmen. Sie terminieren, liefern aber nicht notwendig das richtige Ergebnis. Sei A ein solcher Algorithmus. Er kann während des Ablaufs zusätzlich zu den anderen möglichen Operationen an jeder Stelle einmal oder mehrfach eine Münze werfen. Jeder Münzwurf wird als eine Bitoperation gezählt. Wegen der Möglichkeit von Münzwürfen kann die Laufzeit von A bei ein und derselben Eingabe variieren. Darum wird die Laufzeit als Erwartungswert einer Zufallsvariablen definiert. Wir fixieren eine Eingabe. Als Ereignismenge wählen wir die Menge aller Folgen von Münzwürfen, die bei den unterschiedlichen Durchläufen von A mit der gewählten Eingabe auftreten können. Die Zufallsvariable ordnet einer solchen Folge von Münzwürfen die Laufzeit zu, die A bei dieser Auswahl benötigt. Für eine Eingabelänge n ist dann $T(n)$ die maximale erwartete Laufzeit bei Eingaben der Länge n. Die Begriffe der linearen, quadratischen, polynomiellen, subexponentiellen und exponentiellen Laufzeit übertragen sich entsprechend.

1.4.7 Durchschnittliche Laufzeit

Bis jetzt haben wir nur die Laufzeit von Algorithmen im schlechtesten Fall beschrieben. In vielen Situationen ist aber auch die *durchschnittliche Laufzeit* interessant. Sie wird auch *Average-Case-Laufzeit* genannt und durch folgendes Modell beschrieben. Eingaben einer festen Länge n werden gemäß einer festgelegten Wahrscheinlichkeitsverteilung ausgewählt. Betrachtet wird die Zufallsvariable, die diesen Eingaben die entsprechende erwartete Laufzeit zuordnet. Die durchschnittliche Laufzeit ist der Erwartungswert dieser Zufallsvariablen. Auch hier übertragen sich die Begriffe der linearen, quadratischen, polynomiellen, subexponentiellen und exponentiellen Laufzeit.

1.5 Berechnungsprobleme

Schwere Berechnungsprobleme sind die Grundlage für die Sicherheit von kryptographischen Verfahren. *Berechnungsprobleme* bestehen darin, einen möglichst effizienten Algorithmen zu finden, der aus einer zulässigen Eingabe eine gewünschte Ausgabe macht.

Beispiel 1.25 Ein Berechnungsproblem besteht zum Beispiel darin, aus einem Paar (a, b) natürlicher Zahlen den größten gemeinsamen Teiler von a und b zu berechnen. Die zulässigen Eingaben sind bei diesem Berechnungsproblem alle Paare (a, b) natürlicher Zahlen. Die gewünschte Ausgabe ist der ggT von a und b.

Tab. 1.1 Sicherheitsparameter

Schutz bis zum Jahr a	Sicherheitsparameter $k(a)$
2020	96
2030	112
2040	128
für absehbare Zukunft	256

Ein Berechnungsproblem besteht also aus einer Beschreibung der zulässigen Eingaben und der gewünschten Ausgaben eines Algorithmus. Die Ausgabe ist jeweils die Lösung der durch die Eingabe festgelegten *Instanz* des Problems. Die Lösung muss nicht eindeutig sein.

Die nächste Definition erlaubt es uns, schwere Berechnungsprobleme zu charakterisieren.

Definition 1.8 Seien t und ε positive reelle Zahlen, $0 \leq \varepsilon \leq 1$. Ein Berechnungsproblem P heißt (t, ε)-*schwer*, wenn die Erfolgswahrscheinlichkeit aller probablistischen Algorithmen A, die P in Worst-Case-Laufzeit $\leq t$ lösen, höchstens ε ist.

Um schwere Berechnungsprobleme zu definieren, verwenden wir die Funktion k, die einer Jahreszahl eine natürliche Zahl zuordnet. Sie hat folgende Bedeutung: Bis zum Jahr a ist es unmöglich, $2^{k(a)}$ Operationen durchzuführen. Eine Tabelle von Werten dieser Funktion zeigt Tab. 1.1. Sie stammt aus Forschungsergebnissen des EU-Projekts ECRYPT II (siehe [73]). Sie berücksichtigt die gegenwärtige und für die Zukunft prognostizierte Fähigkeit von Computern.

Definition 1.9 Ein Berechnungsproblem P heißt *unlösbar* bis zum Jahr a, wenn die Erfolgswahrscheinlichkeit aller probablilistischen Algorithmen A, die P in Worst-Case-Laufzeit $\leq 2^{k(a)}$ lösen, höchstens $1/2^{k(a)}$ ist.

Es ist im Allgemeinen nicht möglich, zu beweisen, dass ein Berechnungsproblem bis zu einem bestimmten Jahr schwer ist. Statt dessen wird Definition 1.9 verwendet, um Sicherheitsvoraussetzungen für kryptographische Verfahren exakt beschreiben zu können. Dazu werden Aussagen von folgender Art bewiesen: Das kryptographische Verfahren K ist sicher bis zum Jahr a solange das Berechnungsproblem P bis zum Jahr a unlösbar ist.

Schließlich definieren wir asymptotisch schwere Berechnungsprobleme. Dazu benötigen wir noch einen Begriff.

Definition 1.10 Sei P ein Berechnungsproblem und sei A ein probabilistischer Algorithmus, der P löst. Sei $\Pr(k)$ die maximale Erfolgswahrscheinlichkeit von A bei einer Eingabe der Länge k. Die Erfolgswahrscheinlichkeit von A heißt *vernachlässigbar*, wenn $\Pr(k) = \mathrm{O}(1/k^c)$ gilt für alle $c > 0$.

Beispiel 1.26 Betrachte das Berechnungsproblem P aus Beispiel 1.25. Die zulässigen Eingaben sind Paare (a, b) natürlicher Zahlen. Die gewünschte Ausgabe ist $\gcd(a, b)$. Wir wenden folgenden Algorithmus an: Er bestimmt die binären Längen von a und b und setzt l auf das Minimum dieser beiden Zahlen. Dann wählt der Algorithmus zufällig und gleichverteilt eine natürliche Zahl g mit Bitlänge $\leq l$, indem er l-mal eine Münze wirft und so die Bits von g bestimmt. Diese Zahl gibt der Algorithmus zurück. Die Erfolgswahrscheinlichkeit dieses Algorithmus ist $1/2^l$, weil es 2^l natürliche Zahlen der Bitlänge $\leq l$ gibt, von denen jede mit Wahrscheinlichkeit $1/2^l$ zurückgegeben wird. Aber nur eine ist der gesuchte ggT.

Wir zeigen, dass diese Erfolgswahrscheinlichkeit vernachlässigbar ist. Die Länge k der Eingabe $I = (a, b)$ ist mindestens $2l$. Bezeichnet also $\Pr(k)$ die Erfolgswahrscheinlichkeit des Algorithmus bei einer Eingabe der Länge k, so gilt

$$\Pr(k) = 1/2^l \leq 1/2^{k/2}. \tag{1.21}$$

Dies impliziert, dass $\Pr(k) = \mathrm{O}(1/k^c)$ ist für alle $c > 0$.

Jetzt folgt die Definition asymptotisch schwerer Berechnungsprobleme.

Definition 1.11 Ein Berechnungsproblem P heißt *asymptotisch schwer*, wenn die Erfolgswahrscheinlichkeit jedes polynomiell beschränkten probabilistischen Algorithmus, der P löst, vernachlässigbar ist.

Beispiel 1.27 Obwohl der ggT-Algorithmus aus Beispiel 1.26 vernachlässigbare Erfolgswahrscheinlichkeit hat, ist ggT-Berechnen nicht asymptotisch schwer. In Abschn. 1.6 wird nämlich der euklidische Algorithmus vorgestellt, der größte gemeinsame Teiler deterministisch, also mit Erfolgswahrscheinlichkeit 1, in Polynomzeit berechnet.

Es ist im Allgemeinen auch nicht möglich, zu beweisen, dass ein Berechnungsproblem asymptotisch schwer ist. Statt dessen wird der Begriff *asymptotisch schwer* ebenfalls dazu benutzt, um Sicherheitsvoraussetzungen für kryptographische Verfahren anzugeben. Dazu werden Aussagen von folgender Art bewiesen: Solange Berechnungsproblem P asymptotisch schwer ist, ist das kryptographische Verfahren K sicher. Dies wird in Kap. 4 ausführlicher diskutiert.

1.6 Algorithmen für ganze Zahlen

1.6.1 Addition, Multiplikation und Division mit Rest

In vielen kryptographischen Verfahren werden lange ganze Zahlen addiert, multipliziert und mit Rest dividiert. In diesem Abschnitt schätzen wir die Laufzeit von Algorithmen für diese Aufgaben ab.

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 die Zahlen Bit für Bit mit Übertrag.

Beispiel 1.28 Sei $a = 10101, b = 111$. Wir berechnen $a + b$.

$$\begin{array}{rr} & 1\,0\,1\,0\,1 \\ + & \quad 1\,1\,1 \\ \text{Übertrag} & 1\,1\,1 \\ \hline & 1\,1\,1\,0\,0 \end{array}$$

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

$$\begin{array}{rr} & 1\,0\,1\,0\,1 * 1\,0\,1 \\ \hline & \quad 1\,0\,1\,0\,1 \\ + & 1\,0\,1\,0\,1 \\ \text{Übertrag} & \quad 1 \quad 1 \\ \hline & 1\,1\,0\,1\,0\,0\,1 \end{array}$$

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 entsprechenden 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 1.30 Sei $a = 10101, b = 101$. Wir dividieren a mit Rest durch b.

$$\begin{array}{l} 1\,0\,1\,0\,1 = 1\,0\,1 * 1\,0\,0 + 1 \\ 1\,0\,1 \\ \quad 0\,0\,0 \\ \quad 0\,0\,0 \\ \quad\quad 0\,0\,1 \\ \quad\quad 0\,0\,0 \\ \quad\quad\quad 1 \end{array}$$

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

Der benötigte Platz aller dieser Operation ist $O(\text{size}\, a + \text{size}\, b)$.

Tatsächlich gibt es schnellere Algorithmen für Multiplikation und Division mit Rest ganzer Zahlen. Die schnellsten von ihnen sind quasilinear. Für mehr Details verweisen wir auf [3] und [39].

1.6.2 Euklidischer Algorithmus

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

Theorem 1.9
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 1.2 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 1.1. □

Wir erläutern den euklidischen Algorithmus erst an einem Beispiel.

Beispiel 1.31 Wir möchten $\gcd(100, 35)$ berechnen. Nach Theorem 1.9 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. Algorithmus 1.3 zeigt den euklidischen Algorithmus im Pseudocode.

Algorithmus 1.3 (euclid(a, b))

$a \leftarrow |a|$
$b \leftarrow |b|$
while $b \neq 0$ **do**
  $r \leftarrow a \bmod b$
  $a \leftarrow b$
  $b \leftarrow r$
end while
return a

Theorem 1.10 *Der euklidische Algorithmus 1.3 berechnet den größten gemeinsamen Teiler von a und b.*

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

$$r_0 = |a|, r_1 = |b| \tag{1.22}$$

und für $k \geq 1$ und $r_k \neq 0$

$$r_{k+1} = r_{k-1} \bmod r_k. \tag{1.23}$$

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 1.9 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 (1.23) 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 entweder $\gcd(a, b)$ 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

$$q_k = \lfloor r_{k-1}/r_k \rfloor, \quad 1 \le k \le n. \tag{1.24}$$

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

$$r_{k-1} = q_k r_k + r_{k+1}. \tag{1.25}$$

Beispiel 1.32 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 1.2 *Es gilt $q_k \ge 1$ für $1 \le k \le n-1$ und $q_n \ge 2$.*

Beweis Da $r_{k-1} > r_k > r_{k+1}$ gilt, folgt aus (1.25), dass $q_k \ge 1$ ist für $1 \le k \le n$. Angenommen, $q_n = 1$. Dann folgt $r_{n-1} = r_n$ und das ist nicht möglich, weil die Restefolge streng monoton fällt. Daher ist $q_n \ge 2$. □

Theorem 1.11 *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 1.28 können wir annehmen, dass $\gcd(a, b) = r_n = 1$ ist. Durch Induktion wird bewiesen, dass

$$r_k \ge \Theta^{n-k}, \quad 0 \le k \le n \tag{1.26}$$

gilt. Dann ist insbesondere

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

Durch Logarithmieren erhält man daraus

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

wie behauptet.

Wir beweisen nun (1.26). Zunächst gilt

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

und nach Lemma 1.2

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

Sei $n-2 \geq k \geq 0$ und gelte die Behauptung für $k' > k$. Dann folgt aus Lemma 1.2

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

Damit sind (1.26) und das Theorem bewiesen. □

1.6.3 Erweiterter euklidischer Algorithmus

Im vorigen Abschnitt haben wir gesehen, wie man den größten gemeinsamen Teiler zweier ganzer Zahlen berechnen kann. In Korollar 1.2 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 Abschn. 1.6.2 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

$$x_{k+1} = q_k x_k + x_{k-1}, \quad y_{k+1} = q_k y_k + y_{k-1}, \quad 1 \leq k \leq n. \tag{1.27}$$

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

Theorem 1.12 *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 1.33 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 Pseudocode findet man in Algorithmus 1.4.

Algorithmus 1.4 (xEuclid(a, b))

$xs[0] \leftarrow 1, xs[1] \leftarrow 0$
$ys[0] \leftarrow 0, ys[1] \leftarrow 1$
sign $\leftarrow 1$
while $b \neq 0$ **do**
  $q \leftarrow \lfloor a/b \rfloor$
  $r \leftarrow a - qb$
  $a \leftarrow b$
  $b \leftarrow r$
  $xx \leftarrow xs[1]$
  $yy \leftarrow ys[1]$
  $xs[1] \leftarrow q \cdot xs[1] + xs[0]$
  $ys[1] \leftarrow q \cdot ys[1] + ys[0]$
  $xs[0] \leftarrow xx$

$ys[0] \leftarrow yy$
sign $\leftarrow -$sign
end while
$x \leftarrow \text{sign} \cdot xs[0]$
$y \leftarrow -\text{sign} \cdot ys[0]$
return (a, x, y)

Die Korrektheit des erweiterten euklidischen Algorithmus 1.4 folgt aus Theorem 1.12.

1.6.4 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

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

so gelten wegen

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

die Rekursionen

$$u_{k-1} = q_k u_k + u_{k+1}, \quad v_{k-1} = q_k v_k + v_{k+1}, \quad 1 \le k \le n. \tag{1.28}$$

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 1.3 *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 1.2 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 1.3 können wir Abschätzungen für die Koeffizienten x_k und y_k ableiten.

Korollar 1.5 *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 1.3 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 1.6 *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 1.4 *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 1.13 *Sind a und b ganze Zahlen, dann braucht die Anwendung des erweiterten euklidischen Algorithmus auf a und b Zeit $O((\text{size}\, a)(\text{size}\, b))$.*

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 Abschn. 1.6.1 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

$$T_1(a,b) = O\left(\text{size}(b)\left(n + \sum_{k=1}^{n} \log q_k\right)\right). \tag{1.29}$$

Nach Theorem 1.11 ist

$$n = O(\text{size}\, b). \tag{1.30}$$

Ferner ist

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

Daraus folgt

$$\sum_{k=1}^{n} \log q_k = O(\text{size}\, a). \tag{1.31}$$

Setzt man (1.30) und (1.31) in (1.29) 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 1.5 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

$$T_2(a,b) = O\left(\text{size}(a)\left(1 + \sum_{k=2}^{n} \text{size}(q_k)\right)\right) = O\left(\text{size}(a)\left(n + \sum_{k=2}^{n} \log q_k\right)\right). \tag{1.32}$$

Wie oben beweist man leicht

$$\prod_{k=2}^{n} q_k \leq b. \tag{1.33}$$

Setzt man dies in (1.32) ein, folgt die Behauptung. Damit ist das Theorem bewiesen. □

1.7 Übungen

Übung 1.1 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 1.2 Bestimmen Sie die Anzahl der Teiler von 2^n, $n \in \mathbb{Z}_{\geq 0}$.

Übung 1.3 Bestimmen Sie alle Teiler von 195.

Übung 1.4 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 1.5 Berechnen Sie 1243 mod 45 und -1243 mod 45.

Übung 1.6 Finden Sie eine ganze Zahl a mit $a \bmod 2 = 1$, $a \bmod 3 = 1$, und $a \bmod 5 = 1$.

Übung 1.7 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 1.8 Berechnen Sie die Binärdarstellung und die Hexadezimaldarstellung von 225.

Übung 1.9 Bestimmen Sie die binäre Länge der n-ten Fermat-Zahl $2^{2^n} + 1$, $n \in \mathbb{Z}_{\geq 0}$.

Übung 1.10 Schreiben Sie ein Programm, das für gegebenes $g \geq 2$ die g-adische Darstellung einer natürlichen Zahl n berechnet.

Übung 1.11 Sei S eine endliche Menge und Pr eine Wahrscheinlichkeitsverteilung auf S. Zeigen Sie:

1. $\Pr(\emptyset) = 0$.
2. Aus $A \subset B \subset S$ folgt $\Pr(A) \leq \Pr(B)$.

Übung 1.12 In einem Experiment wird m zufällig und gleichverteilt aus der Menge $\{1, 2, \ldots, 1000\}$ gewählt. Bestimmen Sie die Wahrscheinlichkeit dafür,

1. ein Quadrat zu erhalten;
2. eine Zahl mit i Primfaktoren zu erhalten, $i \geq 1$.

Übung 1.13 Geben Sie die Ergebnismenge und die Wahrscheinlichkeitsverteilung an, die einem Wurf von zwei Münzen entspricht. Geben Sie das Ereignis „wenigstens eine Münze zeigt Kopf“ an und berechnen Sie seine Wahrscheinlichkeit.

Übung 1.14 Es wird mit zwei Würfeln gewürfelt. Wie groß ist die Wahrscheinlichkeit dafür, dass beide Würfel ein verschiedenes Ergebnis zeigen unter der Bedingung, dass die Summe der Ergebnisse gerade ist?

Übung 1.15 Bestimmen Sie n so, dass die Wahrscheinlichkeit dafür, dass zwei von n Personen am gleichen Tag Geburtstag haben, wenigstens $9/10$ ist.

Übung 1.16 Angenommen, vierstellige Geheimnummern von EC-Karten werden zufällig verteilt. Wieviele Leute muss man versammeln, damit die Wahrscheinlichkeit dafür, dass zwei dieselbe Geheimnummer haben, wenigstens $1/2$ ist?

Übung 1.17 Berechnen Sie den Erwartungswert für das Produkt der Ergebnisse zweier unabhängiger Würfelexperimente.

Übung 1.18 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 1.19 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 1.20 Berechnen Sie die Erfolgswahrscheinlichkeit von Algorithmus 1.2.

Übung 1.21 Schätzen Sie die erwartete Laufzeit von Algorithmus 1.2 ab.

Übung 1.22 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 1.23 Beweisen Sie, dass der eulidische Algorithmus auch funktioniert, wenn die Division mit Rest so modifiziert ist wie in Übung 1.4.

Übung 1.24 Berechnen Sie $\gcd(235, 124)$ samt seiner Darstellung mit dem erweiterten euklidischen Algorithmus.

Übung 1.25 Benutzen Sie den modifizierten euklidischen Algorithmus aus Übung 1.23, um $\gcd(235, 124)$ einschließlich Darstellung zu berechnen. Vergleichen Sie diese Berechnung mit der Berechnung aus Beispiel 1.24.

Übung 1.26 Beweisen Sie Lemma 1.4.

Übung 1.27 Sei $a > b > 0$. Beweisen Sie, dass der modifizierte euklidische Algorithmus aus Beispiel 1.23 $O(\log b)$ Iterationen braucht, um $\gcd(a, b)$ zu berechnen.

Übung 1.28 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 1.29 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 1.30 Zeigen Sie, dass aus $\gcd(a, m) = 1$ und $\gcd(b, m) = 1$ folgt, dass $\gcd(ab, m) = 1$ ist.

Übung 1.31 Berechnen Sie die Primfaktorzerlegung von 37.800.

Übung 1.32 Zeigen Sie, dass jede zusammengesetzte Zahl $n > 1$ einen Primteiler $p \leq \sqrt{n}$ hat.

Übung 1.33 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, das diese Idee implementiert.