5 Randomisierte Algorithmen für Probleme aus der Zahlentheorie

Zu Aussagen, die mit $(*)$ markiert sind, gibt es Beweise oder Anmerkungen im Anhang A.

In diesem Kapitel betrachten wir randomisierte Algorithmen für Probleme, die mit dem Rechnen mit ganzen Zahlen zu tun haben. Dies sind Primzahltests, ein darauf aufbauendes Verfahren zur Erzeugung von zufälligen Primzahlen großer Bitlänge sowie ein Verfahren zum Ziehen von Quadratwurzeln modulo p, wo p eine Primzahl ist. Insbesondere wegen ihrer Anwendung in der Kryptographie haben diese Verfahren sehr große praktische Bedeutung. Zunächst diskutieren wir die nötigen Grundkonzepte aus der Zahlentheorie und grundlegende effiziente Algorithmen für natürliche und ganze Zahlen.

5.1 Fakten aus der Zahlentheorie und grundlegende Algorithmen

5.1.1 Teiler, Reste, Euklidischer Algorithmus

Die in diesem Kapitel relevanten Zahlenbereiche sind:

- $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$, die natürlichen Zahlen, und
- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$, die ganzen Zahlen.

In den Diskussionen in diesem Kapitel sehen wir ganze Zahlen unter zwei Blickwinkeln an: Erstens ist es die unendliche Menge $\mathbb{Z}$, die mit ihren abstrakten Operationen $+$, $-$, $\cdot$ und $/$ und den neutralen Elementen 0 und 1 einen *Ring mit* 1 bildet. Über diese Struktur und Eigenschaften von ganzen Zahlen können wir *mathematische Aussagen* machen und beweisen. Andererseits interessieren uns *Algorithmen* für Zahlen. Für Betrachtungen aus dieser Perspektive stellen wir uns die Zahlen immer als zu

einer passenden Basis b dargestellt vor: Binärdarstellung, Dezimaldarstellung, Hexadezimaldarstellung, Darstellung zur Basis 256 (eine Ziffer ist ein Byte) oder 2^{32} (eine Ziffer ist ein 32-Bit-Wort, passend für die Darstellung in einem Rechner). Eine Zahl ist dann als ein String über einem endlichen Alphabet gegeben. Die Anzahl der Ziffern in der Darstellung von $a \in \mathbb{N}$ zur Basis b ist $\lceil \log_b(a+1) \rceil$. Für Rechenzeitbetrachtungen ist es nicht wichtig, ob negative Zahlen als Paar aus Vorzeichen und Absolutbetrag oder in Zweierkomplementdarstellung dargestellt werden.

Wir nehmen an, dass Algorithmen für die folgenden Grundoperationen auf ganzen Zahlen gegeben sind: Addition, Subtraktion, Multiplikation, Division (mit Rest). Zwei einziffrige Zahlen können in Zeit $O(1)$ addiert und subtrahiert werden („von der Hardware“). Addition und Subtraktion zweier n-ziffriger Zahlen kosten Zeit $O(n)$, Multiplikation einer n-ziffrigen und einer ℓ-ziffrigen Zahl (mit der Schulmethode) kostet Zeit $O(n\ell)$, ebenso Division einer n-ziffrigen durch eine ℓ-ziffrige Zahl. Dabei entspricht „Rechenzeit“ einfach der Anzahl der ausgeführten Operationen auf einzelnen Ziffern.[1]

Eine Zahl $x \in \mathbb{Z}$ heißt **Teiler** von $y \in \mathbb{Z}$, wenn es ein $z \in \mathbb{Z}$ mit $y = x \cdot z$ gibt. Notation: $x \mid y$. Wir sagen dann auch, dass y von x geteilt wird oder dass y ein Vielfaches von x ist. Wenn x nicht Teiler von y ist, schreibt man $x \nmid y$.

Beachte: Die Teiler von 1 sind 1 und -1. Bezüglich der Teilbarkeitsrelation $x \mid y$ gibt es keinen Unterschied zwischen x und $-x$ und zwischen y und $-y$. Die Teilbarkeitsrelation ist reflexiv und transitiv: $x \mid x$ und $(x \mid y \wedge y \mid z) \Rightarrow x \mid z$. Weiter gilt: Wenn x Teiler von $y \neq 0$ ist, dann gilt $|x| \leq |y|$. Die Teilbarkeitsrelation wird in Abb. 5.1.1 anschaulich dargestellt.

Eine immer wieder benutzte Eigenschaft ist, dass die Menge $V_x = \{y \in \mathbb{Z} \mid x \text{ teilt } y\}$ der Vielfachen von x unter Addition und Multiplikation mit beliebigen Zahlen abgeschlossen ist, d. h.:[2]

$$x \mid y \ \wedge \ x \mid z \quad \Rightarrow \quad x \mid (sy + tz), \text{ für beliebige } s, t \in \mathbb{Z}. \tag{5.1.1}$$

[1] Es gibt asymptotisch schnellere Verfahren: Der Karatsuba-Algorithmus (s. Vorlesung „Algorithmen und Datenstrukturen“) mit Rechenzeit $O(n^{1,59})$ für zwei n-ziffrige Zahlen, der Algorithmus von Schönhage und Strassen sogar mit Rechenzeit $O(n \log n \log\log n)$. Im Jahr 2019 erschien eine Arbeit [David Harvey and Joris van der Hoeven, Integer multiplication in time $O(n \log n)$, Proceedings of the Thirteenth Algorithmic Number Theory Symposium, S. 293–310, 2019. `http://dx.doi.org/10.2140/obs.2019.2.293`], die zeigt, dass man zwei n-Bit-Zahlen in Zeit $O(n \log n)$ multiplizieren kann. Nach aktuellem Stand ergeben sich Vorteile aber erst für unrealistisch lange Zahlen. Wir werden diese „schnellen“ Algorithmen nicht berücksichtigen.

[2] Man sagt: V_x ist ein *Ideal* in $\mathbb{Z}$.

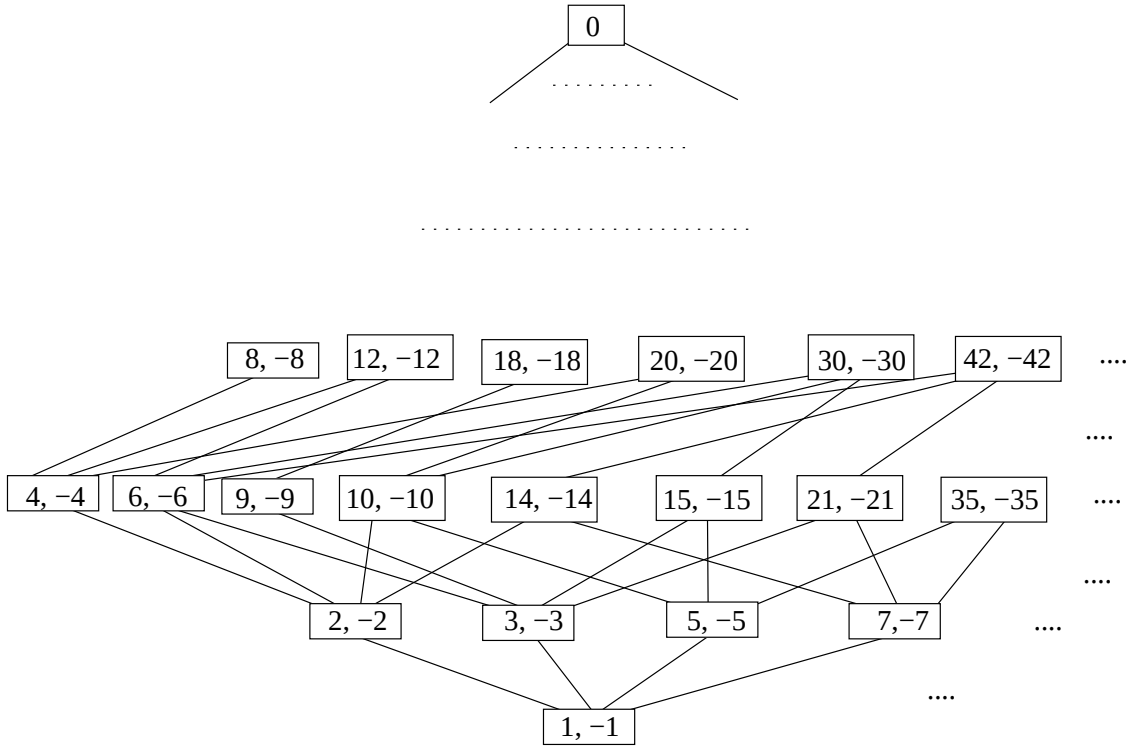

Abbildung 5.1.1: Teilbarkeitsrelation, Ausschnitt. Eine Linie verbindet x (unten) mit y (weiter oben), wenn $x \mid y$ und $|x| \neq |y|$ gilt und wenn es kein z gibt, das $|z| < |y|$ und $x \mid z$ und $z \mid y$ erfüllt.

Fakt 5.1.1 ***Division mit Rest***
Zu $x \in \mathbb{Z}$ und $m \geq 2$ gibt es ein r mit $0 \leq r < m$ und ein q mit $x = qm + r$. Die Zahlen q und r sind eindeutig bestimmt.

Beispiele für $m = 4$: $23 = 5 \cdot 4 + 3$, $-12 = (-3) \cdot 4 + 0$, $-23 = (-6) \cdot 4 + 1$.

Die Zahl $r = x - qm$ („**Rest**“) bezeichnen wir mit $\boldsymbol{x}$ **mod** $\boldsymbol{m}$. Beachte, dass $x - r$ ein Vielfaches von m ist. Der „**Quotient**“ $q = \lfloor x/m \rfloor$ wird auch mit $\boldsymbol{x}$ **div** $\boldsymbol{m}$ bezeichnet. Der Divisionsalgorithmus für Zahlen in Dezimaldarstellung, wie er in der Schule gelehrt wird, berechnet aus Eingabe x und m Quotient q und Rest r, in Rechenzeit $O((\log x)(\log m))$.

Wir betrachten nun die Menge $\mathbb{Z}_m = [m] = \{0, 1, \ldots, m-1\}$ und Operationen

- Addition modulo m: $+_m \colon \mathbb{Z}_m^2 \ni (x, y) \mapsto x +_m y := (x + y) \bmod m \in \mathbb{Z}_m$, und
- Multiplikation modulo m: $\cdot_m \colon \mathbb{Z}_m^2 \ni (x, y) \mapsto x \cdot_m y := xy \bmod m \in \mathbb{Z}_m$.

Fakt 5.1.2 $(*)$
Für jedes $m \geq 2$ bildet $\mathbb{Z}_m$ mit $+_m$ und $\cdot_m$ einen *kommutativen Ring mit* 1. Die neutralen Elemente für $+_m$ und $\cdot_m$ sind 0 und 1.

Das heißt im Detail: Die Operationen $+_m$ und $\cdot_m$ führen nicht aus dem Bereich $\mathbb{Z}_m$ heraus. Die Addition erfüllt alle Rechenregeln für abelsche Gruppen, mit neutralem Element 0. Insbesondere hat jedes Element $x \in \mathbb{Z}_m$ ein additives Inverses $-x$ (beachte $-x = m - x$ für $0 < x < m$ und $-0 = 0$). Die Multiplikation ist assoziativ und kommutativ. Die 1 ist neutrales Element. Für Addition und Multiplikation gelten die Distributivgesetze.

Lemma 5.1.3 $(*)$
Für jedes $m \geq 2$ ist die Abbildung $\mathbb{Z} \to \mathbb{Z}_m$, $x \mapsto x \bmod m$, ein *Homomorphismus*; d. h. für $x, y \in \mathbb{Z}$ gilt:

(i) $(x + y) \bmod m = (x \bmod m) +_m (y \bmod m)$,

(ii) $xy \bmod m = (x \bmod m) \cdot_m (y \bmod m)$.

Dieses Lemma wird hauptsächlich benutzt, um Rechnungen zu vereinfachen. Um einen arithmetischen Ausdruck modulo m auszuwerten (inklusive Potenzen), kann man auf beliebige Zwischenergebnisse die (mod m)-Operation anwenden.

Beispiel: Um $13^7 \bmod 11$ zu berechnen, rechnet man

$$\begin{aligned}(13 \bmod 11)^7 \bmod 11 = 2^7 \bmod 11 &= ((2^5 \bmod 11) \cdot 4) \bmod 11 \\ &= 10 \cdot 4 \bmod 11 = 40 \bmod 11 = 7.\end{aligned}$$

Fakt 5.1.4 $(*)$
Für jedes $m \geq 2$ und alle $x, y \in \mathbb{Z}$ gilt:
$x \bmod m = y \bmod m$ genau dann wenn $m \mid x - y$.

Beispiel: $29 \bmod 12 = 53 \bmod 12 = 5$, und $53 - 29 = 24$ ist durch 12 teilbar.

Schreibweise: $x \equiv y \pmod m$ (oder „$x \equiv_m y$“ oder „$x \equiv y\,(m)$“) bedeutet, dass $x - y$ durch m teilbar ist. Man sagt: „x ist *kongruent* y modulo m.“ Aus Fakt 5.1.4 folgt sofort, dass $\equiv_m$ eine Äquivalenzrelation ist. (Sie ist reflexiv, symmetrisch und transitiv.)

Fakt 5.1.5
Für jedes $m \geq 2$ und alle ganzen Zahlen x_1, x_2, y_1, y_2 gilt:

$$x_1 \equiv_m y_1 \text{ und } x_2 \equiv_m y_2 \qquad \Rightarrow \qquad x_1 + x_2 \equiv_m y_1 + y_2 \text{ und } x_1 \cdot x_2 \equiv_m y_1 \cdot y_2.$$

Dies hat den Effekt, dass man bei mit „$\equiv$“ geschriebenen Transformationen von $\{+, -, \cdot\}$-Ausdrücken stets Zahlen oder Unterausdrücke durch kongruente Zahlen bzw. Unterausdrücke ersetzen kann. Dies führt zu größerer Flexibilität und daher u. U. einfacheren Rechnungen als nur die Rechnung mit Resten in $[m]$, wie von Lemma 5.1.3 erlaubt.

Beispiel: $13^7 \equiv 2^7 \equiv 2^5 \cdot 2^2 = 32 \cdot 4 \equiv (-1) \cdot 4 = -4 \equiv 7 \pmod{11}$.

Definition 5.1.6 ***Größter gemeinsamer Teiler***
Für $x, y \in \mathbb{Z}$ ist $\mathrm{ggT}(x, y)$, der *größte gemeinsame Teiler* von x und y, die (eindeutig bestimmte) *nichtnegative* Zahl d mit:

$$\text{(i) } d \mid x \text{ und } d \mid y; \quad \text{(ii) wenn } c \mid x \text{ und } c \mid y \text{ gilt, dann folgt } c \mid d.$$

Zahlen d, die nur (i) erfüllen, heißen *gemeinsame Teiler* von x und y. Wenn x und y nicht beide 0 sind, beschreibt die Definition den größten gemeinsamen Teiler im Standardsinn, d. h. die größte ganze Zahl, die sowohl x als auch y teilt. Weiter folgt aus der Definition[3]: $\mathrm{ggT}(0, 0) = 0$. Zwei Zahlen x und y mit $\mathrm{ggT}(x, y) = 1$ heißen ***teilerfremd***. Der größte gemeinsame Teiler zweier Zahlen ist eindeutig bestimmt.

Beispiele: Die gemeinsamen Teiler von 0 und -3 sind $-3, -1, 1, 3$, und es gilt $\mathrm{ggT}(0, -3) = 3$.

[3]Die Teilbarkeitsrelation $\mid$ ist reflexiv und transitiv; man nennt so etwas (partielle) *Quasiordnung* oder *Präordnung*. Bezüglich dieser partiellen Quasiordnung sind 1 und -1 kleinste Elemente (sie teilen jede ganze Zahl) und 0 ist größtes Element (jede ganze Zahl teilt die 0), vgl. Abb. 5.1.1. Im Sinn dieser partiellen Quasiordnung ist 0 also das größte Element, das $x = 0$ und $y = 0$ teilt.

Die gemeinsamen Teiler von -30 und 24 sind $-6, -3, -2, -1, 1, 2, 3, 6$, und es gilt $\mathrm{ggT}(-30, 24) = 6$.
Die gemeinsamen Teiler von -12 und 35 sind $-1, 1$, also gilt $\mathrm{ggT}(-12, 35) = 1$; die beiden Zahlen sind teilerfremd.

Es gibt einen wohlbekannten effizienten Algorithmus zur Ermittlung des größten gemeinsamen Teilers von zwei Zahlen. (Dieser beweist auch die Existenz.) Weil Teilbarkeit nicht vom Vorzeichen der Argumente abhängt, gilt $\mathrm{ggT}(x, y) = \mathrm{ggT}(|x|, |y|)$; daher kann man sich darauf beschränken, $\mathrm{ggT}(a, b)$ für $a, b \geq 0$ zu berechnen. Weiter gelten die Gleichungen

$$\begin{aligned}
&\mathrm{ggT}(a, 0) = a \text{ , für beliebige } a \geq 0,\\
&\mathrm{ggT}(a, b) = \mathrm{ggT}(b, a) \text{ und}\\
&\mathrm{ggT}(a, b) = \mathrm{ggT}(b, a \bmod b) \text{ , für } b > 0. \qquad (\text{Und: } a \bmod b < b.)
\end{aligned}$$

(Die beiden ersten Gleichungen sind offensichtlich. Für die dritte überlegt man sich Folgendes. Wir haben $a \bmod b = a - qb$ für $q = \lfloor a/b \rfloor$. Daraus folgt mit (5.1.1), dass jeder gemeinsame Teiler von a und b auch $a \bmod b = a - qb$ teilt und dass jeder gemeinsame Teiler von b und $a \bmod b$ auch $a = qb + (a \bmod b)$ teilt. Also haben die Zahlenpaare (a, b) und $(b, a \bmod b)$ dieselbe Menge gemeinsamer Teiler, und damit auch denselben größten gemeinsamen Teiler.)

Diese Gleichungen liefern sofort einen rekursiven Algorithmus für den größten gemeinsamen Teiler, aber sie führen auch leicht zu einem iterativen Verfahren.

Algorithmus 5.1.7 ***Euklidischer Algorithmus***
Input: Zwei ganze Zahlen x und y.
Methode:

```
1    a, b: integer; a ← |x|; b ← |y|;
2    while b > 0 repeat
3        (a, b) ← (b, a mod b);      // simultane Zuweisung
4    return a.
```

Fakt 5.1.8 $(*)$
Algorithmus 5.1.7 gibt $\mathrm{ggT}(x, y)$ aus. Die Gesamtzahl von ausgeführten Ziffernoperationen ist $O((\log x)(\log y))$.

i	a_i	b_i
0	10534	12742
1	12742	10534
2	10534	2208
3	2208	1702
4	1702	506
5	506	184
6	184	138
7	138	46
8	46	0

Tabelle 1: Ablauf des Euklidischen Algorithmus auf Eingabe $x = 10534$, $y = 12742$.

Man beachte, dass $\lceil \log(x+1) \rceil \approx \log x$ die Anzahl der Bits ist, die man braucht, um x aufzuschreiben. Damit hat der Euklidische Algorithmus bis auf einen konstanten Faktor denselben Aufwand wie die Multiplikation von x und y, wenn man die Schulmethode benutzt.

Beispiel: Auf Eingabe $x = 10534$, $y = 12742$ ergibt sich der in Tab. 1 angegebene Ablauf. Die Zahlen a_i und b_i bezeichnen den Inhalt der Variablen `a` und `b`, nachdem die Schleife in Zeilen 2–3 i-mal ausgeführt worden ist. Die Ausgabe ist $46 = \mathrm{ggT}(10534, 12742)$.

Beispiel: (a) 21 und 25 sind teilerfremd. Es gilt $31 \cdot 21 + (-26) \cdot 25 = 651 - 650 = 1$.
(b) Es gilt $\mathrm{ggT}(21, 35) = 7$, und $2 \cdot 35 - 3 \cdot 21 = 7$.

Die folgende sehr nützliche Aussage verallgemeinert diese Beobachtung:

Lemma 5.1.9 ***... von Bezout*** $(*)$
(a) Wenn $x, y \in \mathbb{Z}$ teilerfremd sind, gibt es $s, t \in \mathbb{Z}$ mit $sx + ty = 1$.
(b) Für $x, y \in \mathbb{Z}$ gibt es s, t mit $sx + ty = \mathrm{ggT}(x, y)$.

Beweis: (a) folgt direkt aus (b). Also müssen wir nur (b) beweisen. Wenn $x = y = 0$, gilt $sx + ty = 0 = \mathrm{ggT}(x, y)$ für alle $s, t \in \mathbb{Z}$. Seien also ab hier nicht x und y beide 0. Wir definieren $I_{x,y} := \{sx + ty \mid s, t \in \mathbb{Z}\}$. Diese Menge enthält die Zahlen $|x| \geq 0$

(setze $s = \text{sign}(x)$ und $t = 0$) und $|y| \geq 0$ (setze $s = 0$ und $t = \text{sign}(y)$), und sie hat folgende „Abschlusseigenschaft“:

$$\text{Wenn } a \geq b > 0 \text{ und } a, b \in I_{x,y}, \text{ dann gilt auch } (a \bmod b) \in I_{x,y}.$$

(Wir haben $a = s_a x + t_a y$ und $b = s_b x + t_b y$ für passende Koeffizienten und $a \bmod b = a - qb$ für den Quotienten $q = \lfloor a/b \rfloor$. Dann gilt $a \bmod b = (s_a - q s_b)x + (t_a - q t_b)y$, also $(a \bmod b) \in I_{x,y}$.) Man betrachtet nun den Ablauf des Euklidischen Algorithmus 5.1.7, gestartet mit $a = |x| \in I_{x,y}$ und $b = |y| \in I_{x,y}$. Nach der Abschlusseigenschaft sind sämtliche dabei entstehenden Zwischenwerte Elemente von $I_{x,y}$, also auch der schließlich entstehende Wert $\text{ggT}(x, y)$. □

Die Überlegungen in diesem Beweis führen direkt zu einem sehr effizienten Algorithmus, dem Erweiterten Euklidischen Algorithmus, der zu x und y die Koeffizienten s und t aus dem Lemma von Bezout berechnet. Die Rechenzeiten des folgenden Algorithmus sind (in O-Notation) ebenso groß wie die des gewöhnlichen Euklidischen Algorithmus.

Algorithmus 5.1.10 ***Erweiterter Euklidischer Algorithmus***
EINGABE: Ganze Zahlen x, y.
METHODE:

```
0   a, b, s_a, t_a, s_b, t_b, q: integer;
1   (a, b) ← (|x|, |y|);
2   (s_a, t_a, s_b, t_b) ← (sign(x), 0, 0, sign(y));
3   while b > 0 repeat
4       q ← a div b;
5       (a, b) ← (b, a − q · b);
6       (s_a, t_a, s_b, t_b) ← (s_b, t_b, s_a − q · s_b, t_a − q · t_b);
7   return (a, s_a, t_a);
```

Genau wie im (einfachen) Euklidischen Algorithmus findet die eigentliche Arbeit in der **while**-Schleife statt (4 Zeilen). Wie dort werden in den Variablen `a` und `b` Zahlen $a, b \geq 0$ mitgeführt, die stets $\text{ggT}(a, b) = d = \text{ggT}(x, y)$ erfüllen. Die Variablen $\mathtt{s_a}$, $\mathtt{t_a}$, $\mathtt{s_b}$, $\mathtt{t_b}$ enthalten stets Zahlen s_a, t_a, s_b, t_b, die folgende Gleichungen erfüllen:

$$a = s_a \cdot x + t_a \cdot y \quad \text{und} \quad b = s_b \cdot x + t_b \cdot y.$$

Diese Gleichung wird durch die Initialisierung hergestellt. In einem Schleifendurchlauf wird a durch b ersetzt und (s_a, t_a) durch (s_b, t_b), und es wird b durch $a - q \cdot b$ ersetzt sowie (s_b, t_b) durch $(s_a - q \cdot s_b, t_a - q \cdot t_b)$. Dadurch bleiben die Gleichungen gültig. Wenn schließlich $b = 0$ geworden ist, gilt $d = \mathrm{ggT}(x, y) = a = s_a \cdot x + t_a \cdot y$. Das bedeutet, dass die Ausgabe das gewünschte Ergebnis darstellt. Als Beispiel betrachten wir den Ablauf des Algorithmus auf der Eingabe $(x, y) = (10534, 12742)$. Die Zahlen a_i ,b_i, $s_{a,i}$, $t_{a,i}$, $s_{b,i}$, $t_{b,i}$ bezeichnen den Inhalt von `a`, `b`, $\mathtt{s_a}$, $\mathtt{t_a}$, $\mathtt{s_b}$, $\mathtt{t_b}$ nach dem i-ten Schleifendurchlauf. Die Zahl q_i ist der Quotient im i-ten Durchlauf.

i	a_i	b_i	$s_{a,i}$	$t_{a,i}$	$s_{b,i}$	$t_{b,i}$	q_i
0	10534	12742	1	0	0	1	–
1	12742	10534	0	1	1	0	0
2	10534	2208	1	0	−1	1	1
3	2208	1702	−1	1	5	−4	4
4	1702	506	5	−4	−6	5	1
5	506	184	−6	5	23	−19	3
6	184	138	23	−19	−52	43	2
7	138	46	−52	43	75	−62	1
8	46	0	75	−62	−277	229	3

Die Ausgabe ist $(46, 75, -62)$. Man überprüft leicht, dass $46 = \mathrm{ggT}(10534, 12742) = 75 \cdot 10534 - 62 \cdot 12742$ gilt.

Fakt 5.1.11

(a) Wenn Algorithmus 5.1.10 auf Eingabe (x, y) die Ausgabe (d, s, t) liefert, dann gilt $d = \mathrm{ggT}(x, y) = sx + ty$.

(b) Die Anzahl der Schleifendurchläufe ist dieselbe wie beim gewöhnlichen Euklidischen Algorithmus.

(c) Die Anzahl von Ziffernoperationen für Algorithmus 5.1.10 ist $O((\log |x|)(\log |y|))$.

Wir notieren noch eine wichtige Folgerung aus dem Lemma von Bezout.[4]

[4] In der Schule begründet man diese Tatsache mit Hilfe der Primzahlzerlegung. Dieser Umweg ist aber gar nicht nötig.

Fakt 5.1.12
Wenn x und y teilerfremd sind und a sowohl durch x als auch durch y teilbar ist, dann ist a auch durch xy teilbar.

Beweis: Weil x und y Teiler von a sind, kann man $a = ux$ und $a = vy$ schreiben, für ganze Zahlen u, v. Weil x und y teilerfremd sind, folgt aus Lemma 5.1.9, dass man $1 = \text{ggT}(x, y) = sx + ty$ schreiben kann, für ganze Zahlen s, t. Dann ist $a = asx + aty = vysx + uxty = (vs + ut)xy$, also ist xy Teiler von a. □

5.1.2 Eigenschaften der Multiplikation in $\mathbb{Z}_m$

In $\mathbb{Z}_m$ spielen die Elemente, die bezüglich $\cdot_m$ ein *multiplikatives Inverses* haben, eine spezielle Rolle.

Lemma 5.1.13
Für jedes $m \geq 2$ und jedes $a \in \mathbb{Z}$ gilt:
Es gibt ein b mit $ab \bmod m = 1$ genau dann wenn $\text{ggT}(a, m) = 1$.

Beweis: „⇒“: Weil $ab \bmod m = 1$ gilt, können wir $ab = qm + 1$ schreiben, für ein $q \in \mathbb{Z}$. Wenn nun eine Zahl d Teiler von a und von m ist, dann teilt d auch $ab - qm = 1$, also ist $d \in \{-1, 1\}$. Daraus folgt $\text{ggT}(a, m) = 1$.
„⇐“: Weil $\text{ggT}(a, m) = 1$ gilt, können wir nach dem Lemma von Bezout $sa + tm = 1$ schreiben, für passende ganze Zahlen s und t. Setze $b = s \bmod m$. Dann gilt $ab = a(s - \lfloor s/m \rfloor m) = 1 - (t + a\lfloor s/m \rfloor)m$, also gilt $ab \bmod m = 1$. □

Wegen der Homomorphieeigenschaft (Lemma 5.1.3) kommt es für die Frage, ob $ab \bmod m = 1$ gilt, nur auf $a \bmod m$ und $b \bmod m$ an. Wir können unsere Überlegungen also auf Elemente von $\mathbb{Z}_m$ beschränken. Wenn $a, b \in \mathbb{Z}_m$ und $ab \bmod m = 1$, nennen wir b ein *multiplikatives Inverses* zu a modulo m.

Beispiel: Aus $6 \cdot 21 + (-5) \cdot 25 = 1$ folgt, dass 6 ein multiplikatives Inverses zu 21 modulo 25 ist.

Definition 5.1.14
Für $m \geq 2$ sei $\mathbb{Z}_m^* := \{a \in \mathbb{Z}_m \mid \text{ggT}(a, m) = 1\}$.

Fakt 5.1.15 $(*)$
Für jedes $m \geq 2$ gilt:
$\mathbb{Z}_m^*$ mit der Multiplikation modulo m als Operation ist eine abelsche Gruppe.

Beispiel: $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$ und $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$.

Am schönsten ist die Situation, wenn alle Zahlen aus $\mathbb{Z}_m$ außer der 0 in $\mathbb{Z}_m^*$ liegen. Dies ist genau dann der Fall, wenn m Primzahl ist.

Definition 5.1.16

(a) Eine Zahl $p \geq 1$ heißt **Primzahl**, wenn p genau zwei positive Teiler hat.[a] (Diese Teiler sind dann 1 und p.)

(b) Eine Zahl $x \geq 1$ heißt **zusammengesetzt**, wenn sie einen Teiler y mit $1 < y < x$ besitzt.[b]

[a] Die Zahl 1 hat nur einen positiven Teiler, nämlich 1. Also ist 1 keine Primzahl.
[b] Die Zahl 1 besitzt keinen Teiler y mit $1 < y < 1$. Also ist 1 auch nicht zusammengesetzt.

Proposition 5.1.17 $(*)$
Für jedes $m \geq 2$ gilt:
m ist Primzahl $\Leftrightarrow$ $\mathbb{Z}_m^* = \{1, \ldots, m-1\}$ $\Leftrightarrow$ $\mathbb{Z}_m$ ist ein Körper.

Die zweite Äquivalenz ist dabei klar: Der Ring $\mathbb{Z}_m$ ist nach Definition ein Körper genau dann wenn jedes Element von $\mathbb{Z}_m - \{0\}$ ein multiplikatives Inverses besitzt. Für die erste Äquivalenz (die auch einfach einzusehen ist) siehe Anhang A.

Beispiel: Wir rechnen modulo 13 und geben für jedes $a \in \mathbb{Z}_{13}^*$ das Inverse b sowie das Produkt $a \cdot b$ an (das natürlich bei der Division durch 13 Rest 1 lassen muss).

a	1	2	3	4	5	6	7	8	9	10	11	12
b	1	7	9	10	8	11	2	5	3	4	6	12
$a \cdot b$	1	14	27	40	40	66	14	40	27	40	66	144

14 ist keine Primzahl, und es gibt keine Zahl b mit $2 \cdot b \bmod 14 = 1$; das heißt, dass $2 \notin \mathbb{Z}_{14}^*$ und daher, dass $\mathbb{Z}_{14}$ kein Körper ist.

Basis für randomisierte Primzahltests ist das folgende grundlegende Ergebnis. Wir beginnen mit einem Beispiel, für die Primzahlen $p = 7$ und $p = 257$.

$$\begin{aligned} 5^6 \equiv (-2)^6 \equiv 64 \equiv 1 \qquad & (\text{mod } 7); \\ 3^6 \equiv 9^3 \equiv 2^3 \equiv 8 \equiv 1 \qquad & (\text{mod } 7); \\ 2^{256} \equiv (2^8)^{32} \equiv 256^{32} \equiv (-1)^{32} \equiv 1 \qquad & (\text{mod } 257). \end{aligned}$$

Dass hier stets das Ergebnis 1 entsteht, ist kein Zufall.

Fakt 5.1.18 ***Kleiner Satz von Fermat***
Wenn p eine Primzahl ist, dann gilt:

$$a^{p-1} \bmod p = 1, \text{ für jedes } a \in \mathbb{Z} \text{ mit } \mathrm{ggT}(a,p) = 1 \text{ (d. h. } p \not| \, a).$$

Beweis: Da sich a und $a \bmod p$ bezüglich Rechnungen modulo p gleich verhalten, können wir $a \in \{1, \ldots, p-1\}$ annehmen. (Es gilt $a \bmod p \neq 0$, weil a nicht durch p teilbar ist.) Betrachte die Abbildung

$$g_a \colon \mathbb{Z}_p^* \ni x \mapsto ax \bmod p \in \mathbb{Z}_p^*.$$

(Beispiel: Wenn man mit $a = 4$ modulo $p = 7$ rechnet, ist $(g_4(1), \ldots, g_4(6)) = (4, 1, 5, 2, 6, 3)$. Man sieht, dass alle Zahlen in $\{1, \ldots, 6\}$ im Bild von g_4 vorkommen.) Die Abbildung g_a ist injektiv, da gilt: $a^{-1} \cdot g_a(x) \bmod p = a^{-1}ax \bmod p = x$. Also gilt

$$\{1, \ldots, p-1\} = \{a \cdot_p 1, \ldots, a \cdot_p (p-1)\}.$$

Wir multiplizieren, in $\mathbb{Z}_p$, alle Elemente der Menge $\{1, \ldots, p-1\}$, in zwei verschiedenen Reihenfolgen, und gruppieren dann um:

$$1 \cdot_p \ldots \cdot_p (p-1) = (a \cdot_p 1) \cdot_p \ldots \cdot_p (a \cdot_p (p-1)) = a^{p-1} \cdot_p (1 \cdot_p \ldots \cdot_p (p-1)).$$

Weil p eine Primzahl ist, ist nach Fakt 5.1.15 die Zahl $P := 1 \cdot_p \ldots \cdot_p (p-1)$ ein Element von $\mathbb{Z}_p^*$. Daher existiert P^{-1}, das Inverse in dieser Gruppe. Wenn wir beide Seiten der letzten Gleichung mit P^{-1} multiplizieren, erhalten wir $1 = a^{p-1} \bmod p$. □

Wir bemerken, dass auch die Umkehrung des kleinen Satzes von Fermat gilt. Wenn m keine Primzahl ist, dann gibt es ein $a \in \{1, \ldots, m-1\}$ mit $\mathrm{ggT}(a,m) \neq 1$. Für jede solche Zahl gilt $a^{m-1} \bmod m \neq 1$. (Wäre $a^{m-1} \bmod m = 1$, hätten wir $a \cdot a^{m-2} - qm = 1$ für ein q, also $\mathrm{ggT}(a,m) = 1$.)

Ab hier soll p immer eine Primzahl bezeichnen. Normalerweise ist dabei $p > 2$, also p ungerade.[5]

Oft kommt es darauf an, Potenzen $x^y \bmod m$ zu berechnen, für ein $y \geq 0$. Dabei kann y eine sehr große Zahl sein. *Beispiel* für eine solche Aufgabe: Berechne

$$3^{1384788374932954500363985403554603584759089} \bmod 28374610730064817362847320073310723412 34.$$

Der Exponent hat 43 Dezimalziffern und seine Binärdarstellung hat 140 Bits. Ein Computeralgebraprogramm berechnet das Ergebnis[6] in Sekundenbruchteilen. Wie kann das gehen? Sicherlich nicht durch y-faches Multiplizieren! Es gibt eine einleuchtende rekursive Formel, die formal auf Lemma 5.1.3 beruht und die den Weg zu einer effizienten Berechnung weist:[7]

$$x^y \bmod m = \begin{cases} 1, & \text{wenn } y = 0, \\ x \bmod m, & \text{wenn } y = 1, \\ ((x^2 \bmod m)^{y/2}) \bmod m, & \text{wenn } y \geq 2 \text{ gerade ist}, \\ ((x^2 \bmod m)^{(y-1)/2} \bmod m) \cdot x \bmod m, & \text{wenn } y \geq 2 \text{ ungerade ist}. \end{cases}$$

Diese Formel führt unmittelbar zu folgender rekursiver Prozedur.

Algorithmus 5.1.19 ***Schnelle modulare Exponentiation***
function modexp(x, y, m) // berechnet $z = x^y \bmod m$
EINGABE: Ganze Zahlen x, $y \geq 0$, und $m \geq 2$, mit $0 \leq x < m$.
METHODE:
0 **if** $y = 0$ **then return** 1;
1 **if** $y = 1$ **then return** x;
2 z $\leftarrow$ modexp$(x \cdot x \bmod m, \lfloor y/2 \rfloor, m)$; // rekursiver Aufruf
3 **if** y ist ungerade **then** z $\leftarrow$ z $\cdot\, x \bmod m$
4 **return** z.

Um $x^y \bmod m$ für beliebige x zu berechnen, ruft man modexp$(x \bmod m, y, m)$ auf.

Was macht dieser Algorithmus anschaulich? Wir betrachten ein Beispiel. Sei $x = 2$, $y = 25$, $m = 7$. Es entstehen rekursive Aufrufe mit $y = 25, 12, 6, 3, 1$, wobei das erste

[5] „All primes are odd, except 2, which is the oddest of all.“ (D. E. Knuth) – Ein Wortspiel damit, dass „odd“ sowohl „ungerade“ als auch „merkwürdig“ bedeuten kann. Das zahlentheoretische Verhalten der Potenzen von 2 ist ganz anders als das der Potenzen einer Primzahl $p > 2$.

[6] 74590552468911142187725098562763415853 5.

[7] Den Fall $y = 1$ könnte man im Programmtext einsparen.

Argument die Werte $x = 2$, $x^2 = 2^2 = 4$, $x^4 = 4^2 \bmod 7 = 2$, $x^8 = 2^2 = 4$, $x^{16} = 4^2 \bmod 7 = 2$ annimmt. Beim Zurückgehen in der Rekursion werden die Argumente aufmultipliziert, bei denen y ungerade ist, dies sind 2, ~~4~~, ~~2~~, 4, 2, mit Produkt 16 mod $7 = 2$. Diese Auswahl entspricht gerade der Binärdarstellung 11001 von 25, von rechts nach links gelesen. Allgemein berechnet die Rekursion iterativ die Potenzen $x^{2^i} \bmod m$, für $i = 0, 1, \ldots, \lfloor \log y \rfloor$, und die Prüfung, ob y gerade oder ungerade ist, wählt die Faktoren $x^{2^i} \bmod m$ aus, bei denen Bit b_i in der Binärdarstellung $b_k \ldots b_1 b_0$ von y Wert 1 hat.

Man erkennt sofort, dass bei jedem rekursiven Aufruf die Bitanzahl des Exponenten y um 1 sinkt, dass also die Anzahl der Rekursionsebenen etwa $\log y$ beträgt. In jeder Rekursionsebene ist eine oder sind zwei Multiplikationen modulo m auszuführen, was $O((\log m)^2)$ Ziffernoperationen erfordert (Schulmethode für Multiplikation und Division). Damit kommt man bei der schnellen Exponentiation selbst bei der einfachsten Implementierung der Multiplikation mit $O((\log y)(\log m)^2)$ Ziffernoperationen zum Ergebnis.

Lemma 5.1.20
Die Berechnung von $x^y \bmod m$ benötigt $O(\log y)$ Multiplikationen und Divisionen modulo m von Zahlen aus $\{0, \ldots, m^2 - 1\}$ sowie $O((\log y)(\log m)^2)$ Bitoperationen.

Bemerkung 1: Man kann denselben Algorithmus in einem beliebigen *Monoid* (Bereiche mit einer assoziativen Operation und einem neutralen Element) benutzen. Einen Monoid bilden zum Beispiel die natürlichen Zahlen mit der Multiplikation oder die Menge Σ^* über einem Alphabet Σ mit der Konkatenation. In Abschnitt 5.4 benutzen wir die schnelle Exponentiation für einen etwas ungewöhnlichen Monoid. Wenn man Zeile 1 weglässt und $y \geq 1$ fordert, genügt auch eine assoziative Operation ohne neutrales Element, um das Verfahren anwendbar zu machen.

Bemerkung 2: Der Algorithmus kann auch iterativ formuliert werden, was zu einer beschleunigten Ausführung führt. Die Idee ist, schon beim Aufbau der Potenzen $x^{2^i} \bmod m$ in der Ergebnisvariablen `z` diejenigen Werte aufzumultiplizieren, die zu 1-Bits in der Binärdarstellung von y gehören. Die Anordnung der Multiplikationen ist also umgekehrt.

Algorithmus 5.1.21 ***Schnelle modulare Exponentiation, iterativ***
function modexp-iter(x, y, m) // berechnet $z = x^y \bmod m$
EINGABE: Ganze Zahlen x, $y \geq 0$, und $m \geq 2$, mit $0 \leq x < m$.
METHODE:

```
0    z ← 1;
1    y ← y;    // verbleibender Exponent
2    xpt ← x;    // x^{2^i}, i = 0, 1, 2, ...
3    while y > 0 do
4        if y is odd then z ← z · xpt mod m;
5        xpt ← xpt · xpt mod m;
6        y ← ⌊y/2⌋;
7    return z.
```

Es ist eine gute Übung in der Verwendung von Schleifeninvarianten, die Korrektheit dieser Implementierung direkt zu beweisen. Die Schleifeninvariante, gültig jeweils vor der Ausführung des Tests in Zeile 3, lautet: Wenn $\hat{x}$, $\hat{y}$, $\hat{z}$ die Inhalte der Variablen `xpt`, `y`, `z` bezeichnen, gilt:

$$\hat{z} \cdot (\hat{x})^{\hat{y}} \bmod m = x^y \bmod m.$$

Nach Verlassen der **while**-Schleife gilt $\hat{y} = 0$, also $\hat{z} = x^y \bmod m$.

5.1.3 Der Chinesische Restsatz und die Eulersche φ-Funktion

Der „Chinesische Restsatz“ besagt im Wesentlichen, dass für teilerfremde Zahlen m und n die Strukturen $\mathbb{Z}_m \times \mathbb{Z}_n$ (mit komponentenweisen Operationen) und $\mathbb{Z}_{mn}$ isomorph sind.

Wir beginnen mit einem Beispiel, nämlich $m = 3$, $n = 8$, also $mn = 24$. Die folgende Tabelle gibt die Reste der Zahlen $x \in \{0, 1, \ldots, 23\}$ modulo 3 und modulo 8 an. Die Restepaare wiederholen sich zyklisch für die anderen Elemente $x \in \mathbb{Z}$.

x	0	1	2	3	4	5	6	7	8	9	10	11
$x \bmod 3$	0	1	2	0	1	2	0	1	2	0	1	2
$x \bmod 8$	0	1	2	3	4	5	6	7	0	1	2	3

x	12	13	14	15	16	17	18	19	20	21	22	23
$x \bmod 3$	0	1	2	0	1	2	0	1	2	0	1	2
$x \bmod 8$	4	5	6	7	0	1	2	3	4	5	6	7

Wenn wir die Einträge in Zeilen 2 und 3 als 24 Paare in $\mathbb{Z}_3 \times \mathbb{Z}_8$ ansehen, erkennen wir, dass sie alle verschieden sind, also auch alle Möglichkeiten in $\{0,1,2\} \times \{0,1,\ldots,7\}$ abdecken. D. h.: Die Abbildung $x \mapsto (x \bmod 3, x \bmod 8)$ ist eine Bijektion zwischen $\mathbb{Z}_{24}$ und $\mathbb{Z}_3 \times \mathbb{Z}_8$. Zudem spiegeln sich arithmetische Operationen auf den Elementen von $\mathbb{Z}_{24}$ in den Resten modulo 3 und 8 wider. Beispielsweise liefert die komponentenweise Addition von $(2,7)$ und $(2,1)$ das Resultat $(1,0)$, was der Addition von 23 und 17 (modulo 24) mit dem Resultat $40 \bmod 24 = 16$ entspricht. Genauso ist $(2^5 \bmod 3, 3^5 \bmod 8) = (2,3)$, was der Gleichung $11^5 \bmod 24 = 11$ entspricht.

Der Chinesische Restsatz besagt im Wesentlichen, dass eine solche strukturelle Entsprechung zwischen den Resten modulo mn und Paaren von Resten modulo m bzw. n immer gilt, wenn m und n teilerfremd sind.

Fakt 5.1.22 ***Chinesischer Restsatz***
m und n seien teilerfremd. Dann ist die Abbildung[a]

$$\Phi\colon \mathbb{Z}_{mn} \ni x \mapsto (x \bmod m, x \bmod n) \in \mathbb{Z}_m \times \mathbb{Z}_n$$

bijektiv. Zudem: Wenn $\Phi(x) = (x_1, x_2)$ und $\Phi(y) = (y_1, y_2)$, dann gilt:

(a) $\Phi(x +_{mn} y) = (x_1 +_m y_1, x_2 +_n y_2)$;

(b) $\Phi(x \cdot_{mn} y) = (x_1 \cdot_m y_1, x_2 \cdot_n y_2)$;

(c) $\Phi(0) = (0,0)$, $\Phi(1) = (1,1)$, $\Phi(nm-1) = (n-1, m-1)$.

[a] Φ ist der griechische Großbuchstabe *Phi*, gesprochen „(groß-)Fi“.

Für mathematisch-strukturell orientierte Leser/innen: Die Gleichungen (a) und (b) kann man etwas abstrakter auch so fassen: Die Abbildung Φ ist ein Ring-mit-1-Isomorphismus zwischen $\mathbb{Z}_{mn}$ und $\mathbb{Z}_m \times \mathbb{Z}_n$.

Wir bemerken noch Folgendes, obgleich es für die in dieser Vorlesung betrachteten Situationen nicht direkt wichtig ist. Die Funktion Φ lässt sich selbstverständlich durch

zwei Divisionen sehr effizient berechnen. Wie sieht es aber mit der Umkehrung aus? Kann man aus $y \in \mathbb{Z}_m$ und $z \in \mathbb{Z}_n$ *effizient* ein $x \in \mathbb{Z}_{mn}$ mit $\Phi(x) = (y, z)$ berechnen? Dies ist tatsächlich der Fall. Man berechnet mit dem erweiterten Euklidischen Algorithmus das multiplikative Inverse $u = n^{-1} \bmod m$ und das multiplikative Inverse $v = m^{-1} \bmod n$. (Dies ist möglich, weil $\mathrm{ggT}(n, m) = 1$, also $n \in \mathbb{Z}_m^*$ und $m \in \mathbb{Z}_n^*$.) Dann setzt man $x := (yun + zvm) \bmod mn$. Dann ist $x \bmod m = yun \bmod m = y$ (weil $un \bmod m = 1$ gilt) und analog $x \bmod n = zvm \bmod n = z$. Also gilt $\Phi(x) = (y, z)$.

Wir halten noch fest, wie sich Zahlen, die zu m und n teilerfremd sind, in der Sichtweise des Chinesischen Restsatzes verhalten.

Proposition 5.1.23
Wenn man die Abbildung Φ aus dem Chinesischen Restsatz auf $\mathbb{Z}_{mn}^*$ einschränkt, ergibt sich eine Bijektion zwischen $\mathbb{Z}_{mn}^*$ und $\mathbb{Z}_m^* \times \mathbb{Z}_n^*$.

In der Beispieltabelle für $m = 3$, $n = 8$, $mn = 24$ sieht das so aus, für $\mathbb{Z}_3^* = \{1, 2\}$, $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$, $\mathbb{Z}_{24}^* = \{1, 5, 7, 11, 13, 17, 19, 23\}$:

$x \in \mathbb{Z}_{24}^*$	1	5	7	11	13	17	19	23
$x \bmod 3$	1	2	1	2	1	2	1	2
$x \bmod 8$	1	3	5	7	1	3	5	7

Der Chinesische Restsatz und die nachfolgenden Bemerkungen und Behauptungen lassen sich leicht auf $r > 2$ paarweise teilerfremde Faktoren $n_1, \ldots, n_r$ verallgemeinern. Die Aussagen lassen sich durch vollständige Induktion über r beweisen.

Prop. 5.1.23 liefert auch eine Formel für die Kardinalitäten der Mengen $\mathbb{Z}_m^*$ für beliebige $m \geq 2$.

Definition 5.1.24 ***Eulersche φ-Funktion***
Für $m \geq 2$ sei[a]

$$\varphi(m) := |\mathbb{Z}_m^*| = |\{x \mid 0 \leq x < m, \mathrm{ggT}(x, m) = 1\}|.$$

[a] φ ist der griechische Kleinbuchstabe *phi*, gesprochen „(klein-)fi“.

m	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\varphi(m)$	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8

Tabelle 2: Eulersche φ-Funktion für kleine m

Einige Beispielwerte sind in Tab. 2 angegeben.

Folgendes ist eine unmittelbare Konsequenz aus Proposition 5.1.23:

Lemma 5.1.25
Für teilerfremde Zahlen n und m gilt $\varphi(mn) = \varphi(m) \cdot \varphi(n)$.

Uns interessiert (später) besonders der Fall $\varphi(p) = p - 1$ für Primzahlen p und $\varphi(pq) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1)$ für verschiedene Primzahlen p und q. Man kann aber auch eine allgemeine Formel für $\varphi(n)$ gewinnen.

Lemma 5.1.26
Für $m \geq 2$ gilt

$$\varphi(m) = m \cdot \prod_{\substack{p \text{ prim} \\ p \mid m}} \left(1 - \frac{1}{p}\right).$$

Beweis: Wenn m eine Primzahlpotenz p^t ist, dann besteht $\mathbb{Z}_m^*$ aus den Zahlen in $\mathbb{Z}_m = \{0, 1, \ldots, p^t - 1\}$, die nicht durch p teilbar sind. Da es in $\mathbb{Z}_m$ insgesamt p^t Zahlen gibt und p^{t-1} Vielfache von p, gilt $\varphi(m) = p^t - p^{t-1} = m - m/p = m(1 - 1/p)$. Nun nehmen wir an, dass

$$m = p_1^{t_1} \cdots p_s^{t_s}$$

gilt, für verschiedene Primzahlen $p_1, \ldots, p_s$ und $t_1, \ldots, t_s \geq 1$. Die Faktoren $p_1^{t_1}, \ldots, p_s^{t_s}$ sind teilerfremd. Mit Lemma 5.1.25, $(s-1)$-mal angewendet, erhalten wir

$$\varphi(m) = \prod_{i=1}^{s} \varphi(p_i^{t_i}) = \prod_{i=1}^{s} p_i^{t_i}(1 - 1/p_i) = m \cdot \prod_{i=1}^{s} \left(1 - \frac{1}{p_i}\right). \qquad \square$$

Mit dieser Formel lassen sich die Werte in Tabelle 2 schnell verifizieren. (*Beispiel*: $\varphi(12) = 12(1 - 1/2)(1 - 1/3) = 12 \cdot (1/2) \cdot (2/3) = 4$.)

Bemerkung: Die simple Formel in Lemma 5.1.26 könnte zu dem Schluss verleiten, dass sich $\varphi(m)$ zu gegebenem m immer leicht berechnen lässt. Aber Achtung: Man muss

dazu die Menge der Primfaktoren von m kennen. Im allgemeinen Fall läuft das darauf hinaus, dass man das Faktorisierungsproblem für m lösen muss, und hierfür kennt man keine effizienten Algorithmen (also Algorithmen mit Rechenzeit $O((\log m)^c)$ für eine Konstante c). Tatsächlich ist kein effizienter Algorithmus bekannt, der es erlaubt, $\varphi(m)$ aus m zu berechnen.

5.1.4 Primzahlen

Wir erinnern an Definition 5.1.16, die Definition der Konzepte „Primzahl“ und „zusammengesetzte Zahl“.

Fakt 5.1.27
Wenn p eine Primzahl ist und $p \mid xy$ gilt, dann gilt $p \mid x$ oder $p \mid y$.

Beweis: Wenn $p \mid x$, sind wir fertig. Also können wir $p \nmid x$ annehmen, das heißt $\mathrm{ggT}(p, x) = 1$. Nach dem Lemma von Bezout (Lemma 5.1.9) können wir $1 = sp + tx$ schreiben, für ganze Zahlen s, t. Daraus folgt: $y = spy + txy$. Weil xy durch p teilbar ist, folgt $p \mid y$. □

Satz 5.1.28 ***Fundamentalsatz der Arithmetik***
Jede Zahl $N \geq 1$ kann als Produkt von Primzahlen geschrieben werden. Die Faktoren sind dabei bis auf die Reihenfolge eindeutig bestimmt.

Fakt 5.1.29
Jede zusammengesetzte Zahl $N \geq 2$ besitzt einen Primfaktor p mit $p \leq \sqrt{N}$.

Bemerkung: Wir betrachten das resultierende naive Faktorisierungsverfahren: Teste die Zahlen in $\{2, \ldots, \lfloor\sqrt{N}\rfloor\}$ nacheinander darauf, ob sie N teilen; wenn ein Faktor p gefunden wurde (dieses p muss eine Primzahl sein), suche mit demselben Verfahren (startend bei p) Teiler von $N' = N/p$. Dieses Verfahren hat im schlechtesten Fall Rechenzeit mindestens $\Theta(\sqrt{N}) = \Theta(2^{(\log N)/2})$, also exponentiell in der Bitlänge von N. Wie wir später genauer diskutieren werden, sind für das Auffinden der Primzahlzerlegung einer gegebenen Zahl N überhaupt keine effizienten Algorithmen bekannt (also Algorithmen mit Laufzeiten $O((\log N)^c)$ für konstantes c). Aber es gibt effiziente Algorithmen, mit denen man feststellen kann, ob eine Zahl N eine Primzahl

ist oder nicht. Dieser Unterschied in der Schwierigkeit des Faktorisierungsproblems und des Primzahlproblems liegt einer ganzen Reihe von kryptographischen Verfahren zugrunde.

Satz 5.1.30 ***Satz von Euklid***
Es gibt unendlich viele Primzahlen.

Über die Verteilung der Primzahlen (ihre „Dichte“) in $\mathbb{N}$ gibt der berühmte Primzahlsatz[8] Auskunft. Mit $\pi(x)$ bezeichnen wir die Anzahl der Primzahlen, die nicht größer als x sind.

Satz 5.1.31 ***Primzahlsatz***

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1.$$

Das heißt, dass für große x in $(x, 2x]$ etwa $\frac{2x}{\ln(2x)} - \frac{x}{\ln x} \approx \frac{x}{\ln x}$ Primzahlen zu erwarten sind. Die μ-ziffrigen Zahlen bilden das Intervall $[2^{\mu-1}, 2^{\mu})$. Der Anteil der Primzahlen in diesem Intervall ist näherungsweise

$$\frac{2^{\mu-1}/\ln(2^{\mu-1})}{2^{\mu-1}} \approx \frac{1}{(\ln 2)(\mu-1)} \approx 1{,}44/\mu.$$

Für $\mu \approx 2000$ ist der relative Anteil von Primzahlen im interessanten Zahlenbereich also etwa $\approx 1{,}44/2000 \approx 1/1400$. Er sinkt umgekehrt proportional zur Ziffernzahl.

Eine schärfere Form des Primzahlsatzes ist folgende Aussage[9] (wobei der Beweis Nichtspezialisten nicht zugänglich ist):

$$\frac{x}{\ln x}\left(1+\frac{1}{\ln x}\right) \overset{(x\geq 599)}{\leq} \pi(x) \overset{(x\geq 2)}{\leq} \frac{x}{\ln x}\left(1+\frac{1.2762}{\ln x}\right).$$

Für $x \geq 500\,000$ folgt daraus: $\frac{x}{\ln x} < \pi(x) < 1{,}1\frac{x}{\ln x}$.

[8] 1793 von Gauß und unabhängig 1798 von Legendre vermutet; 1896 unabhängig von Hadamard und de La Vallée Poussin bewiesen.

[9] Corollary 5.2 in: Dusart, Pierre. Explicit estimates of some functions over primes. Ramanujan J. 45, 227–251 (2018). https://doi.org/10.1007/s11139-016-9839-4.

Daraus folgt, dass für $n \geq 20$ der Anteil der Primzahlen unter den n-Bit-Zahlen folgende Ungleichung erfüllt:

$$\begin{aligned}\frac{|\{p \in [2^{n-1}, 2^n) \mid p \text{ is prime}\}|}{2^{n-1}} &= \frac{\pi(2^n) - \pi(2^{n-1})}{2^{n-1}} \\ &\geq \frac{2^n/(n \ln 2) - 1{,}1 \cdot 2^{n-1}/((n-1)\ln 2)}{2^{n-1}} \\ &\geq \frac{2}{n \ln 2} - \frac{1{,}1}{n \ln 2} \cdot \frac{n}{n-1} \\ &\geq \frac{6}{5n}.\end{aligned}$$

Mit Hilfe eines Computeralgebraprogramms findet man heraus, dass die Ungleichung $|\{p \in [2^{n-1}, 2^n) \mid p \text{ is prime}\}|/2^{n-1} \geq 6/(5n)$ auch für $9 \leq n \leq 20$ gilt. (Für $n = 8$ ist sie falsch.)Man kann sich also merken:

Für $n \geq 9$ ist der Anteil der Primzahlen an den n-Bit-Zahlen mindestens $\frac{6}{5n}$.

Ziffernzahl n	Dusart-Schranke für $(\pi(2^n) - \pi(2^{n-1}))/2^{n-1}$	numerische untere Schranke
256	$\frac{6}{5 \cdot 256}$	$\geq \frac{1}{214}$
512	$\frac{6}{5 \cdot 512}$	$\geq \frac{1}{427}$
1024	$\frac{6}{5 \cdot 1024}$	$\geq \frac{1}{854}$
2048	$\frac{6}{5 \cdot 2048}$	$\geq \frac{1}{1707}$

Eine leichter zu beweisende Aussage der Art $\pi(2m) - \pi(m) = \Theta(m/\log m)$ ist die folgende:

Satz 5.1.32 ***Ungleichung von Finsler***
Für jede ganze Zahl $m \geq 2$ liegen im Intervall $(m, 2m]$ mindestens $m/(3\ln(2m))$ Primzahlen*:*

$$\pi(2m) - \pi(m) \geq \frac{m}{3\ln(2m)}.$$

Ein vollständiger, vergleichsweise einfacher *Beweis* für Satz 5.1.32 findet sich zum Beispiel in dem Lehrbuch „Elemente der Diskreten Mathematik: Zahlen und Zählen, Graphen und Verbände“ von Diekert, Kufleitner, Rosenberger (De Gruyter 2013).

5.2 Der Miller-Rabin-Primzahltest

In diesem Abschnitt lernen wir einen randomisierten Algorithmus kennen, der es erlaubt, zu einer gegebenen Zahl N zu entscheiden, ob N eine Primzahl ist oder nicht.

Ein idealer Primzahltest sieht so aus:

Eingabe: Eine natürliche Zahl $N \geq 3$.

Ausgabe: 0, falls N eine Primzahl ist; 1, falls N zusammengesetzt ist.

Wozu braucht man Primzahltests? Zunächst ist die Frage „Ist N eine Primzahl?“ eine grundlegende mathematisch interessante Fragestellung. Spätestens mit dem Siegeszug des RSA-Kryptosystems ab den späten 1970er Jahren hat sich die Situation jedoch dahin entwickelt, dass man Algorithmen benötigt, die immer wieder neue vielziffrige Primzahlen (etwa mit 1000 oder 1500 Bits[10] bzw. 301 oder 452 Dezimalziffern) bereitstellen können. Den Kern dieser Primzahlerzeugungs-Verfahren (siehe Abschnitt 5.3) bildet ein Verfahren, das eine gegebene Zahl N darauf testet, ob sie prim ist.

Der naive Primzahltest („trial division“), der dem *brute-force*-Paradigma folgt und oben schon beschrieben wurde, findet durch direkte Division der Zahl N durch $2, 3, 4, \ldots, \lfloor\sqrt{N}\rfloor$ heraus, ob N einen nichttrivialen Teiler hat. Man kann dieses Verfahren durch einige Tricks beschleunigen, aber die Rechenzeit wächst dennoch mit $\Theta(\sqrt{N})$. Dies macht es für Zahlen N mit mehr als 40 Dezimalstellen praktisch undurchführbar, von Zahlen mit mehr als 100 Dezimalstellen ganz zu schweigen. (Achtung: Damit wird nichts über die Qualität anderer Faktorisierungsalgorithmen gesagt. Es gibt andere, sehr fortgeschrittene Faktorisierungsalgorithmen, die bei entsprechendem Zeitaufwand und mit sehr leistungsstarken Rechnern auch noch mit 200-stelligen Zahlen zurechtkommen. Für Information zu früheren und aktuelleren Faktorisierungserfolgen siehe z. B. `https://en.wikipedia.org/wiki/RSA_numbers`.)

[10] `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=12` Stand 24.03.2020. BSI TR-02102-1 Kryptographische Verfahren: Empfehlungen und Schlüssellängen, S. 28: „Für einen Einsatzzeitraum über das Jahr 2022 hinaus wird empfohlen, RSA/DLIES-Schlüssel von 3000 Bits Länge zuverwenden, um ein gleichmäßiges Sicherheitsniveau in allen empfohlenen asymmetrischen Verschlüsselungsverfahren zu erzielen. Die Schlüssellänge von 2000 Bit bleibt für DLIES-Schlüssel bis 2022 zur vorliegenden Richtlinie konform, außerdem übergangsweise für RSA-Schlüssel bis Ende 2023.“ Dabei bedeutet *Schlüssellänge* die Bitanzahl eines Produkts $n = p \cdot q$ für Primzahlen p und q mit jeweils der halben Länge.

In diesem Abschnitt beschreiben wir den randomisierten Primzahltest von Miller und Rabin. Dabei handelt es sich um einen Monte-Carlo-Algorithmus mit einseitigem Fehler. Das heißt: Auf Eingaben N, die Primzahlen sind, wird immer 0 ausgegeben; auf Eingaben N, die zusammengesetzt sind, gibt es eine gewisse (von N abhängige) Wahrscheinlichkeit, dass die Ausgabe 0, also falsch ist. Für kein zusammengesetztes N ist diese Wahrscheinlichkeit größer als die „Fehlerschranke“ $\frac{1}{4}$. Wir beweisen nur die Fehlerschranke $\frac{1}{2}$. Im Beweis benutzen wir einfache zahlentheoretische Überlegungen. Eine herausragende Eigenschaft des Miller-Rabin-Tests ist seine Effizienz. Wir werden sehen, dass selbst bei Verwendung der Schulmethoden für Multiplikation und Division die Bitkomplexität des Primzahltests nur $O((\log N)^3)$ ist, also $O(n^3)$, wenn $n = \log N$ die Bitlänge der Eingabezahl ist.

Bemerkung: Der Miller-Rabin-Algorithmus stammt aus dem Jahr 1977; er folgte einem kurz vorher vorgestellten anderen randomisierten Primzahltest (Solovay-Strassen-Test). Für diesen und andere randomisierte Primzahltests (z. B. den „Strong Lucas Probable Prime Test“ oder den „Quadratic Frobenius Test“ von Grantham) sei auf die Literatur verwiesen. Im Jahr 2002 stellten Agrawal, Kayal und Saxena einen deterministischen Primzahltest mit polynomieller Rechenzeit vor. Die Rechenzeit ist z. B. durch $O((\log N)^{7,5})$ beschränkt. Dieser Algorithmus stellte insofern einen gewaltigen Durchbruch dar, als er ein Jahrhunderte altes offenes Problem löste, nämlich die Frage nach einem effizienten[11] *deterministischen* Verfahren für das Entscheidungsproblem „ist N Primzahl oder zusammengesetzte Zahl“? Andererseits ist seine Laufzeit im Vergleich etwa zu dem hier diskutierten randomisierten Verfahren so hoch, dass nach wie vor die randomisierten Algorithmen benutzt werden, um für kryptographische Anwendungen Primzahlen zu erzeugen.

Da gerade Zahlen leicht zu erkennen sind, beschränken wir im Folgenden unsere Überlegungen auf ungerade Zahlen $N \geq 3$.

5.2.1 Der Fermat-Test

Wir erinnern uns an den kleinen Satz von Fermat (Fakt 5.1.18):

$$p \text{ ist Primzahl und } 1 \leq a < p \quad \Rightarrow \quad a^{p-1} \bmod p = 1.$$

Wir können diese Aussage dazu benutzen, um „Belege“ oder „Zertifikate“ oder „Zeu-

[11] d. h. mit Rechenzeit polynomiell in der Bitlänge $\log N$.

gen“ dafür anzugeben, dass eine Zahl N zusammengesetzt ist: Wenn wir eine Zahl a mit $1 \leq a < N$ finden, für die $a^{N-1} \bmod N \neq 1$ gilt, dann ist N definitiv keine Primzahl.

Definition 5.2.1
Sei $N \geq 3$ ungerade und zusammengesetzt.
Eine Zahl $a \in \{1, \ldots, N-1\}$ heißt **F-Zeuge** für N, wenn $a^{N-1} \bmod N \neq 1$ gilt.
Eine Zahl $a \in \{1, \ldots, N-1\}$ heißt **F-Lügner** für N, wenn $a^{N-1} \bmod N = 1$ gilt.
Die Menge der F-Lügner nennen wir L_N^{F}.

Wir bemerken, dass ein F-Zeuge zwar belegt, dass es Faktoren $k, \ell > 1$ mit $N = k \cdot \ell$ *gibt*, dass aber ein F-Zeuge nicht auf solche Faktoren hinweist oder sie beinhaltet. Das *Finden* von Faktoren wird von Primzahltests auch nicht verlangt und normalerweise auch nicht geleistet.

Man sieht sofort, dass 1 und $N-1$ immer F-Lügner sind: Es gilt $1^{N-1} \bmod N = 1$ und $(N-1)^{N-1} \equiv (-1)^{N-1} \equiv 1 \pmod{N}$, weil $N-1$ gerade ist.

Für jede zusammengesetzte Zahl N gibt es mindestens einen F-Zeugen. Wir erinnern uns, dass nach Proposition 5.1.17 die Zahl N genau dann zusammengesetzt ist, wenn $\{1, \ldots, N-1\} - \mathbb{Z}_N^* \neq \emptyset$ gilt.

Lemma 5.2.2
Wenn N keine Primzahl ist, ist jedes $a \in \{1, \ldots, N-1\} - \mathbb{Z}_N^*$ ein F-Zeuge.

Beweis: Sei $d = \mathrm{ggT}(a, N) > 1$. Dann ist auch a^{N-1} durch d teilbar, also auch $a^{N-1} \bmod N = a^{N-1} - \lfloor a^{N-1}/N \rfloor \cdot N$. Daher ist $a^{N-1} \bmod N \neq 1$. □

Leider ist für manche zusammengesetzten Zahlen N die Menge $\{1, \ldots, N-1\} - \mathbb{Z}_N^*$ äußerst dünn. Wenn zum Beispiel $N = pq$ für zwei Primzahlen p und q ist, dann gilt $\mathrm{ggT}(a, N) > 1$ genau dann wenn p oder q ein Teiler von a ist. Es gibt genau $p + q - 2$ solche Zahlen a in $\{1, \ldots, N-1\}$, was gegenüber N sehr klein ist, wenn p und q annähernd gleich groß sind. Um eine gute Chance zu haben, F-Zeugen zu finden, sollte es also mehr als nur die in $\{1, \ldots, N-1\} - \mathbb{Z}_N^*$ geben.

Beispiel: $N = 91 = 7 \cdot 13$. Tabelle 3 zeigt, dass es 18 Vielfache von 7 und 13 gibt (für größere p und q wird der Anteil dieser offensichtlichen F-Zeugen noch kleiner sein), und daneben weitere 36 F-Zeugen und 36 F-Lügner in $\{1, 2, \ldots, 90\}$. In diesem

F-Zeugen in $\{1,\ldots,90\} - \mathbb{Z}_{91}^*$:	
7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84; 13, 26, 39, 52, 65, 78	
F-Lügner:	F-Zeugen in $\mathbb{Z}_{91}^*$:
1, 3, 4, 9, 10, 12, 16, 17, 22,	2, 5, 6, 8, 11, 15, 18, 19, 20,
23, 25, 27, 29, 30, 36, 38, 40, 43,	24, 31, 32, 33, 34, 37, 41, 44, 45,
48, 51, 53, 55, 61, 62, 64, 66, 68,	46, 47, 50, 54, 57, 58, 59, 60, 67,
69, 74, 75, 79, 81, 82, 87, 88, 90	71, 72, 73, 76, 80, 83, 85, 86, 89

Tabelle 3: F-Zeugen und F-Lügner für $N = 91 = 7 \cdot 13$. Es gibt 36 F-Lügner und 36 F-Zeugen in $\mathbb{Z}_{91}^*$. Wir wissen nach Lemma 5.2.2, dass alle 18 Vielfachen von 7 und 13 F-Zeugen sind.

Beispiel gibt es um einiges mehr F-Zeugen als F-Lügner. Wenn dies für alle zusammengesetzten Zahlen N der Fall wäre, wäre es eine elegante randomisierte Strategie, einfach zufällig nach F-Zeugen zu suchen.

Dies führt zu unserem ersten Versuch für einen randomisierten Primzahltest.

Algorithmus 5.2.3 ***Fermat-Test***
EINGABE: Ungerade Zahl $N \geq 3$.
METHODE:
1 Wähle a zufällig aus $\{1,\ldots,N-1\}$;
2 **if** $a^{N-1} \bmod N \neq 1$ **then return** 1 **else return** 0.

Die Laufzeitanalyse liegt auf der Hand: Der teuerste Teil ist die Berechnung der Potenz $a^{N-1} \bmod N$ durch schnelle Exponentiation, die nach den Ergebnissen von Lemma 5.1.20 $O(\log N)$ arithmetische Operationen und $O((\log N)^3)$ Ziffernoperationen benötigt. Weiter ist es klar, dass der Algorithmus einen F-Zeugen gefunden hat, wenn er „1“ ausgibt, dass in diesem Fall also N zusammengesetzt sein muss. Umgekehrt ausgedrückt: Wenn N eine Primzahl ist, gibt der Fermat-Test garantiert „0“ aus.

Für $N = 91$ wird das falsche Ergebnis 0 ausgegeben, wenn als a einer der 36 F-Lügner gewählt wird. Die Wahrscheinlichkeit hierfür ist $\frac{36}{90} = \frac{2}{5} = 0{,}4$.

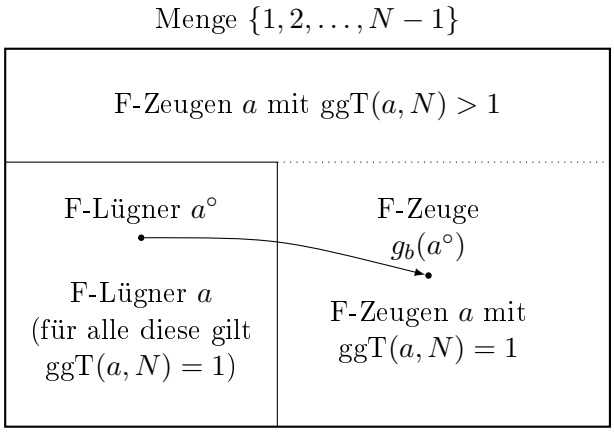

Abbildung 5.2.2: Möglichkeiten für a im Fermat-Test. N ist zusammengesetzt; L_N^{F} ist die Menge der F-Lügner; es gibt einen F-Zeugen $b \in \mathbb{Z}_N^*$. Die Abbildung g_b bildet L_N^{F} injektiv in die Menge der F-Zeugen in $\mathbb{Z}_N^*$ ab. Also gibt es mehr F-Zeugen als F-Lügner.

Für viele zusammengesetzte Zahlen N gibt es reichlich F-Zeugen, so dass der Fermat-Test für diese N mit konstanter Wahrscheinlichkeit das korrekte Ergebnis liefert. Wir analysieren das Verhalten des Fermat-Tests für solche „gutmütigen“ Eingabezahlen N (für die $N = 91$ ein typisches Beispiel ist).

Satz 5.2.4
Sei $N \geq 3$ eine ungerade zusammengesetzte Zahl. (Dann ist $N \geq 9$.) Wenn es mindestens einen F-Zeugen $b \in \mathbb{Z}_N^*$ gibt, dann liefert der Fermat-Test auf Eingabe N mit Wahrscheinlichkeit größer als $\frac{1}{2}$ die korrekte Antwort „1“.

Beweis: Sei $b \in \mathbb{Z}_N^*$ ein F-Zeuge. Betrachte die Funktion $g_b \colon L_N^{\mathrm{F}} \to \mathbb{Z}_N^*$, die den F-Lügner a auf $g_b(a) = ba \bmod N$ abbildet. Wie im Beweis von Fakt 5.1.18 sieht man, dass g_b injektiv ist. Weiter ist $g_b(a)$ für jedes $a \in L_N^{\mathrm{F}}$ ein F-Zeuge:

$$(ba \bmod N)^{N-1} \bmod N = (b^{N-1} \bmod N)(\underbrace{(a^{N-1}) \bmod N}_{=1}) = b^{N-1} \bmod N \neq 1.$$

Wir können also jedem F-Lügner a einen „Zwilling“ $g_b(a)$ in der Menge der F-Zeugen zuordnen. Daraus folgt, dass es in $\mathbb{Z}_N^*$ mindestens so viele F-Zeugen wie F-Lügner gibt. Mit Lemma 5.2.2 ergibt sich, dass $\{1, \ldots, N-1\}$ mehr F-Zeugen als F-Lügner

enthält. Daher ist die Wahrscheinlichkeit, dass die im Fermat-Test gewählte Zahl a ein F-Lügner ist, kleiner als $\frac{1}{2}$. □

Die Fehlerwahrscheinlichkeit $\frac{1}{2}$ ist natürlich viel zu groß. Wir verringern sie durch wiederholte Ausführung des Fermat-Tests.

Algorithmus 5.2.5 ***Iterierter Fermat-Test***
EINGABE: Ungerade Zahl $N \geq 3$, eine Zahl $\ell \geq 1$.
METHODE:
1 **repeat** ℓ **times**
2 $\quad$ $\mathsf{a} \leftarrow$ ein zufälliges Element von $\{1, \ldots, N-1\}$;
3 $\quad$ **if** $\mathsf{a}^{N-1} \bmod N \neq 1$ **then return** 1;
4 **return** 0.

Wenn die Ausgabe 1 ist, hat der Algorithmus einen F-Zeugen für N gefunden, also ist N zusammengesetzt. D. h.: Wenn N eine Primzahl ist, ist die Ausgabe 0. Andererseits: Wenn N zusammengesetzt ist, und es mindestens einen F-Zeugen $a \in \mathbb{Z}_N^*$ gibt, dann ist nach Satz 5.2.4 die Wahrscheinlichkeit für die falsche Ausgabe „0“ höchstens $\left(\frac{1}{2}\right)^\ell = 2^{-\ell}$. Indem wir ℓ genügend groß wählen, können wir die Fehlerwahrscheinlichkeit so klein wie gewünscht einstellen.

Wenn es darum geht, aus einem genügend großen Bereich zufällig gewählte Zahlen darauf zu testen, ob es sich um eine Primzahl handelt, dann ist der Fermat-Test eine sehr effiziente und zuverlässige Methode. Wir kommen im folgenden Abschnitt über Primzahlerzeugung nochmals darauf zurück.

Wenn man allerdings über die Herkunft der zu testenden Zahl N keine Information hat und eventuell damit rechnen muss, dass jemand (ein „Gegenspieler“) absichtlich eine besonders schwierige Eingabe vorlegt, dann stößt der Fermat-Test an eine Grenze. Es gibt nämlich „widerspenstige“ zusammengesetzte Zahlen, denen man mit diesem Test nicht beikommen kann, weil alle Elemente von $\mathbb{Z}_N^*$ F-Lügner sind. Mit diesen befasst sich der folgende Abschnitt.

5.2.2 Carmichael-Zahlen

Definition 5.2.6
Eine ungerade zusammengesetzte Zahl N heißt eine **Carmichael-Zahl**, wenn für alle $a \in \mathbb{Z}_N^*$ die Gleichung $a^{N-1} \bmod N = 1$ gilt.

Die kleinste Carmichael-Zahl ist $561 = 3 \cdot 11 \cdot 17$. Weitere kleine Carmichael-Zahlen sind $1105 = 5 \cdot 13 \cdot 17$ und $1729 = 7 \cdot 13 \cdot 19$. Erst im Jahr 1994 wurde bewiesen, dass es unendlich viele Carmichael-Zahlen gibt, genauer: Wenn x genügend groß ist, dann gibt es in $\{N \mid N \leq x\}$ mehr als $x^{2/7}$ Carmichael-Zahlen. Die aktuell beste bekannte untere Schranke für $|\{N \leq x \mid N \text{ Carmichael-Zahl}\}|$ ist $x^{1/3}$. Von Erdős (1956) stammt die obere Schranke $x \cdot \exp(\frac{-c \ln x \ln\ln\ln x}{\ln\ln x})$ für die Größe dieser Menge, für eine Konstante $c > 0$, die zeigt, dass Carmichael-Zahlen *viel* seltener als Primzahlen sind. (Mehr Details: `https://en.wikipedia.org/wiki/Carmichael_number`.)

Wenn wir dem Fermat-Test eine Carmichael-Zahl N als Eingabe geben, ist die Wahrscheinlichkeit für die falsche Antwort 0 nach Lemma 5.1.26 genau

$$\frac{\varphi(N)}{N-1} > \frac{\varphi(N)}{N} = \prod_{\substack{p \text{ prim} \\ p \text{ teilt } N}} \left(1 - \frac{1}{p}\right) > 1 - \sum_{\substack{p \text{ prim} \\ p \text{ teilt } N}} \frac{1}{p}.$$

Diese Wahrscheinlichkeit liegt nahe an 1, wenn N nur wenige und relativ große Primfaktoren hat. An solchen Carmichael-Zahlen besteht etwa im Bereich der Zahlen im Bereich $[10^{15}, 10^{16}]$ kein Mangel, wie ein Blick in entsprechende Tabellen zeigt. Zum Beispiel ist $N = 651693055693681 = 72931 \cdot 87517 \cdot 102103$ eine Carmichael-Zahl mit $\varphi(N)/N > 0.99996$.

Der Wiederholungstrick zur Wahrscheinlichkeitsverbesserung hilft hier leider auch nicht, denn wenn etwa p_0 der kleinste Primfaktor von N ist, und N nur 3 oder 4 Faktoren hat, dann sind $\Omega(p_0)$ Wiederholungen nötig, um die Fehlerwahrscheinlichkeit auf $\frac{1}{2}$ zu drücken. Sobald p_0 mehr als 30 Dezimalstellen hat, ist dies undurchführbar.

Für einen zuverlässigen, effizienten Primzahltest, der für *alle* Eingabezahlen N funktioniert, müssen wir über den Fermat-Test hinausgehen. Interessanterweise ist dies praktisch ohne Effizienzverlust möglich.

Für spätere Benutzung stellen wir noch eine Hilfsaussage über Carmichael-Zahlen bereit.

Lemma 5.2.7
Wenn N eine Carmichael-Zahl ist, dann ist N keine Primzahlpotenz.

Beweis: Wir beweisen die Kontraposition: Wenn $N = p^\ell$ für eine ungerade Primzahl p und einen Exponenten $\ell \geq 2$ ist, dann ist N keine Carmichael-Zahl.

Dazu genügt es, eine Zahl $a \in \mathbb{Z}_N^*$ anzugeben, so dass $a^{N-1} \bmod N \neq 1$ ist. Wir definieren:

$$a := p^{\ell-1} + 1.$$

(Wenn z. B. $p = 7$ und $\ell = 3$ ist, ist $N = 343$ und $a = 49 + 1 = 50$.) Man sieht sofort, dass $a < p^\ell = N$ ist, und dass a nicht von p geteilt wird, also a und N teilerfremd sind; also ist $a \in \mathbb{Z}_N^*$. Nun rechnen wir modulo N, mit der binomischen Formel:

$$\begin{aligned} a^{N-1} &\equiv (p^{\ell-1} + 1)^{N-1} \\ &\equiv \sum_{0 \leq j \leq N-1} \binom{N-1}{j} (p^{\ell-1})^j \\ &\equiv 1 + (p^\ell - 1) \cdot p^{\ell-1} \pmod{N}. \end{aligned} \tag{5.2.2}$$

(Die letzte Äquivalenz ergibt sich daraus, dass für $j \geq 2$ gilt, dass $(\ell-1)j \geq \ell$ ist, also der Faktor $(p^{\ell-1})^j = p^{(\ell-1)j}$ durch $N = p^\ell$ teilbar ist, der entsprechende Summand also modulo N wegfällt.) Nun ist $p^\ell - 1$ nicht durch p teilbar, also ist $(p^\ell - 1) \cdot p^{\ell-1}$ nicht durch $N = p^\ell$ teilbar. Damit folgt aus (5.2.2), dass $a^{N-1} \not\equiv 1 \pmod{N}$ ist, also $a^{N-1} \bmod N \neq 1$. □

Folgerung: Jede Carmichael-Zahl N lässt sich als $N = N_1 \cdot N_2$ schreiben, wo N_1 und N_2 teilerfremde ungerade Zahlen ≥ 3 sind.

Bemerkung. Ganz ähnlich kann man sogar zeigen, dass die Primfaktoren einer Carmichael-Zahl N alle verschieden sein müssen. Weiter haben Carmichael-Zahlen mindestens drei Primfaktoren.[12] Schon aus diesen Tatsachen kann man entnehmen, dass Carmichael-Zahlen wohl eher selten sind.

5.2.3 Nichttriviale Quadratwurzeln der 1

Lemma 5.2.8
Wenn p eine ungerade Primzahl ist, dann gilt $b^2 \bmod p = 1$, $b \in \mathbb{Z}_p$, genau für $b \in \{1, p-1\}$.

[12] Für an solchen Details interessierte Leser/innen finden sich in Anhang B nicht prüfungsrelevante nähere Informationen zu Carmichael-Zahlen, u. a. ein mathematisches Kriterium und ein randomisierter Algorithmus, um sie zu erkennen.

Beweis: Offensichtlich gilt für jedes beliebige $m \geq 2$, dass $1^2 \bmod m = 1$ und $(m-1)^2 \bmod m = (m(m-2)+1) \bmod m = 1$ ist. Nun sei $b \in \{0, \ldots, p-1\}$ beliebig mit $b^2 \equiv 1 \pmod{p}$. Dann gilt $b^2 - 1 \equiv 0 \pmod{p}$, also ist p ein Teiler von $b^2 - 1 = (b+1)(b-1)$. Nach Fakt 5.1.27 ist p Teiler von $b+1$ oder von $b-1$. Im ersten Fall ist $b \equiv -1 \pmod{p}$, im zweiten Fall ist $b \equiv 1 \pmod{p}$. □

Die im Lemma angegebene Eigenschaft lässt sich in ein weiteres Zertifikat dafür ummünzen, dass eine Zahl N zusammengesetzt ist:

> Wenn es ein $b \in \{2, \ldots, N-2\}$ gibt, das $b^2 \bmod N = 1$ erfüllt,
> dann ist N zusammengesetzt.

Zahlen $b \in \{2, \ldots, N-2\}$ mit $b^2 \bmod N = 1$ heißen ***nichttriviale Quadratwurzeln der* 1 modulo *N***. Solche Zahlen gibt es nur, wenn N zusammengesetzt ist. Beispielsweise sind genau die Zahlen 1, 27, 64 und 90 die Quadratwurzeln der 1 modulo 91; davon sind 27 und $64 = 91 - 27$ nichttrivial. Wir beobachten: $27^2 \bmod 91 = 729 \bmod 91 = 1$. Beachte, dass $27 \equiv -1 \pmod{7}$ und $27 \equiv 1 \pmod{13}$. Allgemeiner sieht man mit der Verallgemeinerung von Fakt 5.1.22 (Chinesischer Restsatz) auf r Faktoren leicht ein, dass es für ein Produkt $N = p_1 \cdots p_r$ aus verschiedenen ungeraden Primzahlen $p_1, \ldots, p_r$ genau 2^r Quadratwurzeln der 1 modulo N gibt, nämlich die Zahlen b, $0 \leq b < N$, die $b \bmod p_j \in \{1, p_j - 1\}$, $1 \leq j \leq r$, erfüllen. Wenn N nicht sehr viele verschiedene Primfaktoren hat, ist es also aussichtslos, einfach zufällig gewählte b's darauf zu testen, ob sie vielleicht nichttriviale Quadratwurzeln der 1 sind. Dennoch wird uns dieser Begriff bei der Formulierung eines effizienten Primzahltests helfen.

5.2.4 Der Miller-Rabin-Test

Wir kehren nochmals zum Fermat-Test zurück und sehen uns die dort durchgeführte Exponentiation $a^{N-1} \bmod N$ etwas genauer an. Die Zahl $N-1$ ist gerade, daher kann man sie als $N - 1 = u \cdot 2^k$ schreiben, für eine ungerade Zahl u und ein $k \geq 1$. (In der Binärdarstellung von $N-1$ ist k die Anzahl der Nullen am Ende, u ist die Zahl, die durch Weglassen dieser Nullen entsteht.) Dann gilt $a^{N-1} \equiv (a^u \bmod N)^{2^k} \bmod N$,

a	$b_0 = a^{81}$	$b_1 = a^{162}$	$b_2 = a^{324}$	F-Z.?	MR-Z.?
2	252	129	66	×	×
7	307	324	1		
32	57	324	1		
49	324	1	1		
65	0	0	0	×	×
126	1	1	1		
201	226	51	1		×
224	274	1	1		×

Tabelle 4: Potenzen $a^{N-1} \bmod N$ mit Zwischenschritten, $N = 325$

und wir können $a^{N-1} \bmod N$ mit $k+1$ Zwischenschritten berechnen: Mit

$$\begin{aligned}
b_0 &= a^u \bmod N \\
b_1 &= b_0^2 \bmod N = a^{u\cdot 2} \bmod N, \\
b_2 &= b_1^2 \bmod N = a^{u\cdot 2^2} \bmod N, \\
&\vdots \\
b_i &= b_{i-1}^2 \bmod N = a^{u\cdot 2^i} \bmod N, \\
&\vdots \\
b_k &= b_{k-1}^2 \bmod N = a^{u\cdot 2^k} \bmod N
\end{aligned}$$

ist $b_k = a^{N-1} \bmod N$. Beispielsweise erhalten wir für $N = 325 = 5^2 \cdot 13$ den Wert $N - 1 = 324 = 81 \cdot 2^2$. (Man bemerkt, dass 325 die Binärdarstellung 101000100 und 81 die Binärdarstellung 1010001 hat.) In Tabelle 4 berechnen wir a^{81}, a^{162} und a^{324}, alle modulo 325, für verschiedene a.

Die Grundidee des Miller-Rabin-Tests ist nun, diese verlangsamte Berechnung der Potenz $a^{N-1} \bmod N$ auszuführen und dabei nach nichttrivialen Quadratwurzeln der 1 Ausschau zu halten.

b_0	b_1	$\cdots$				$\cdots$	b_{k-1}	b_k	Fall	F-Z.?	MR-Z.?
1	1	$\cdots$	1	1	1	$\cdots$	1	1	1		
$N-1$	1	$\cdots$	1	1	1	$\cdots$	1	1	2a		
$*$	$*$	$\cdots$	$*$	$N-1$	1	$\cdots$	1	1	2b		
$*$	$*$	$\cdots$	$*$	$*$	$*$	$\cdots$	$*$	$\neq 1$	3	$\times$	$\times$
$*$	$*$	$\cdots$	$*$	1	1	$\cdots$	1	1	4a		$\times$
$*$	$*$	$\cdots$	$*$	$*$	$*$	$\cdots$	$*$	1	4b		$\times$

Tabelle 5: Potenzen a^{N-1} mod N berechnet mit Zwischenschritten, mögliche Fälle.

Im Beispiel in Tabelle 4 sehen wir, dass 2 ein F-Zeuge für 325 ist, der in $\mathbb{Z}_{325}^*$ liegt, und dass 65 ein F-Zeuge ist, der nicht in $\mathbb{Z}_{325}^*$ liegt. Dagegen sind 7, 32, 49, 126, 201 und 224 alles F-Lügner für 325. Wenn wir aber 201^{324} mod 325 mit zwei Zwischenschritten berechnen, dann entdecken wir, dass 51 eine nichttriviale Quadratwurzel der 1 ist, was beweist, dass 325 keine Primzahl ist. Ähnlich liefert die Berechnung mit Basis 224, dass 274 eine nichttriviale Quadratwurzel der 1 ist. Die entsprechenden Berechnungen mit 7, 32 und 49 dagegen liefern keine weiteren Informationen, weil $7^{162} \equiv 32^{162} \equiv -1 \pmod{325}$ und $49^{81} \equiv -1 \pmod{325}$ gilt. Auch die Berechnung der Potenzen von 126 liefert keine nichttriviale Quadratwurzel der 1, weil $126^{81} \bmod 325 = 1$ gilt.

Wie kann die Folge $b_0, \ldots, b_k$ überhaupt aussehen? Wenn $b_i = 1$ oder $b_i = N-1$ gilt, dann sind $b_{i+1}, \ldots, b_k$ alle gleich 1. Daher beginnt die Folge $b_0, \ldots, b_k$ mit einer (eventuell leeren) Folge von Elementen $\notin \{1, N-1\}$ und endet mit einer (eventuell leeren) Folge von Einsen. Diese beiden Teile können von einem Eintrag $N-1$ getrennt sein; dies muss aber nicht so sein. Die möglichen Muster sind in Tabelle 5 angegeben. Dabei steht „$*$“ für ein Element $\notin \{1, N-1\}$. Wir unterscheiden vier Fälle, mit einigen Unterfällen:

Fall 1: $b_0 = 1$.

Fall 2: In $b_0, \ldots, b_{k-1}$ kommt der Wert $N-1$ vor.
(Fall 2a: $b_0 = N-1$; Fall 2b: $b_i = N-1$ für ein $i \in \{1, \ldots, k-1\}$.)

Fall 3: $b_k \neq 1$. – Dann ist N zusammengesetzt, weil a ein F-Zeuge für N ist.

(Damit dies passieren kann, müssen alle Werte $b_0, \ldots, b_{k-1}$ von 1 und $N-1$ verschieden sein.)

Fall 4: Es gibt ein i mit $0 < i \leq k$ mit $b_{i-1} \notin \{1, N-1\}$ und $b_i = 1$. – Dann ist N zusammengesetzt, weil b_{i-1} eine nichttriviale Quadratwurzel der 1 ist. (Wenn dies passiert, sind alle Werte $b_0, \ldots, b_{i-1}$ von 1 und $N-1$ verschieden.)

Wir beobachten: Wenn N eine Primzahl ist, dann gilt nach Fakt 5.1.18 (Kleiner Satz von Fermat) für jedes a die Gleichung $b_k = a^{N-1} \bmod N = 1$. Weiter kann es nicht passieren, dass $b_{i-1} \notin \{1, N-1\}$ und $b_i = 1$ ist. Daraus folgt, dass für eine Primzahl N *nur* die Fälle 1 und 2 vorkommen können. Wenn wir bei Eingaben N, die Primzahlen sind, keinen Fehler machen wollen, müssen wir also in Fällen 1 und 2 die Ausgabe 0 erzeugen. (Bei einer zusammengesetzten Zahl N als Eingabe tritt Fall 1 etwa für $a = 1$ auf und Fall 2a für $a = N-1$, weil u ungerade ist. Aber auch Fall 2b kann vorkommen; in Tab. 4 ist etwa $a = 32$ eine solche Zahl.)

In den Fällen 3 und 4 stellt die Zahl a (mit ihren Potenzen $b_0, \ldots, b_k$) hingegen einen Beleg dafür dar, dass N keine Primzahl ist: Im Fall 3 ist a ein F-Zeuge, im Fall 4 ist b_{i-1} eine nichttriviale Quadratwurzel der 1, und das kann nur passieren, wenn N zusammengesetzt ist. Hier sollten wir also 1 ausgeben.

Das ist auch schon der ganze Algorithmus von Miller-Rabin, abstrakt formuliert: Wähle a aus $\{1, \ldots, N-1\}$ zufällig. Finde heraus, ob Fall 1 oder 2 eintritt (dann ist die Ausgabe 0) oder Fall 3 oder 4 eintritt (dann ist die Ausgabe 1).

Um einen besonders effizienten Algorithmus zu bekommen, wollen wir die Fallunterscheidung noch etwas stromlinienförmiger gestalten. Betrachten von Tab. 5 zeigt, dass Fall 1 oder 2 genau dann eintritt, wenn Folgendes gilt:

$$b_0 = 1 \quad \text{oder} \quad \text{in der Folge } b_0, \ldots, b_{k-1} \text{ zu } a \text{ erscheint der Wert } N-1. \tag{5.2.3}$$

(Interessanterweise spielt das letzte Folgenglied b_k gar keine Rolle für die Unterscheidung!) Dies führt zu folgender Definition in Analogie zu der der F-Zeugen und F-Lügner.

Definition 5.2.9
Sei $N \geq 3$ ungerade und zusammengesetzt.
Schreibe $N-1 = u \cdot 2^k$, für u ungerade, $k \geq 1$.
Eine Zahl a, $1 \leq a < N$, heißt ein **MR-Lügner für N**, wenn (5.2.3) gilt.
Die Menge der MR-Lügner für N nennen wir L_N^{MR}.
Eine Zahl a, $1 \leq a < N$, heißt ein **MR-Zeuge für N**, wenn (5.2.3) nicht gilt.
(D. h.: $a^u \notin \{1, N-1\}$ und in der Folge $b_1, \ldots, b_{k-1}$ kommt $N-1$ nicht vor.)

Aus der obigen Diskussion der möglichen Fälle folgt sofort:

Lemma 5.2.10
Sei $N \geq 3$ ungerade.

(a) a ist F-Zeuge für $N \quad \Rightarrow \quad a$ ist MR-Zeuge für N (Fall 3).

(b) Wenn N eine Primzahl ist, dann gilt (5.2.3) für alle $a < N$.

Der Test selbst ist leicht und sehr effizient durchführbar: Man wählt a zufällig, berechnet zunächst $b_0 = a^u \bmod N$ und testet, ob $b_0 \in \{1, N-1\}$ ist. Falls ja, trifft Fall 1 bzw. Fall 2a zu, die Ausgabe ist 0. Falls nein, berechnet man durch iteriertes Quadrieren nacheinander die Werte $b_1, b_2, \ldots$, bis

- entweder der Wert $N-1$ auftaucht (Fall 2b, Ausgabe 0)
- oder der Wert 1 auftaucht (Fall 4a, a ist MR-Zeuge, Ausgabe 1)
- oder $b_1, \ldots, b_{k-1}$ berechnet worden sind, ohne dass Werte $N-1$ oder 1 vorgekommen sind (Fall 3 oder 4b, a ist MR-Zeuge, Ausgabe 1).

In Pseudocode sieht das Verfahren dann folgendermaßen aus. Im Unterschied zur bisherigen Beschreibung wird nur eine Variable `b` benutzt, die nacheinander die Werte $b_0, b_1, \ldots$ aufnimmt.

Algorithmus 5.2.11 ***Miller-Rabin-Primzahltest***
EINGABE: Ungerade Zahl $N \geq 3$.
METHODE:

```
1    Bestimme u ungerade und k ≥ 1 mit N − 1 = u · 2^k;
2    wähle zufällig ein a aus {1, ..., N − 1};
3    b ← a^u mod N;                          // mit schneller Exponentiation
4    if b ∈ {1, N − 1} then return 0;        // Fall 1 oder 2a
5    for j from 1 to k − 1 do                // „wiederhole (k − 1)-mal“
6        b ← b^2 mod N;
7        if b = N − 1 then return 0;         // Fall 2b
8        if b = 1 then return 1;             // Fall 4a
9    return 1.                               // Fall 3 oder 4b
```

Wie steht es mit der Laufzeit des Algorithmus? Man benötigt höchstens $\log N$ Divisionen durch 2, um u und k zu finden (Zeile 1). (Wenn man benutzt, dass N normalerweise in Binärdarstellung gegeben sein wird, ist die Sache noch einfacher: k ist die Zahl der Nullen, mit der die Binärdarstellung von $N-1$ aufhört, u ist die Zahl, die durch den Binärstring ohne diese letzten Nullen dargestellt wird.) Die Berechnung von $a^u \bmod N$ mit schneller Exponentiation in Zeile 3 benötigt $O(\log N)$ arithmetische Operationen und $O((\log N)^3)$ Ziffernoperationen (mit der Schulmethode für Multiplikation und Division). Die Schleife in Zeilen 5–8 wird weniger als k-mal durchlaufen; offenbar ist $k \leq \log N$. In jedem Durchlauf ist die Multiplikation modulo N die teuerste Operation. Insgesamt benutzt der Algorithmus $O(\log N)$ arithmetische Operationen auf Zahlen, die kleiner als N^2 sind, und $O((\log N)^3)$ Ziffernoperationen. Dies ist auch die Rechenzeit des Fermat-Tests. Tatsächlich verursacht der Miller-Rabin-Test im Vergleich zum Fermat-Test kaum Mehraufwand.

Nun wenden wir uns dem Ein-/Ausgabeverhalten des Miller-Rabin-Tests zu.

Lemma 5.2.12
Wenn die Eingabe N für den Miller-Rabin-Test eine Primzahl ist, dann wird 0 ausgegeben.

Beweis: Wir haben oben schon festgestellt, dass Ausgabe 1 genau dann erfolgt, wenn die zufällig gewählte Zahl a ein MR-Zeuge ist. Nach Lemma 5.2.10(b) gibt es nur dann MR-Zeugen, wenn N zusammengesetzt ist. □

Historische Notiz. Im Jahr 1976 stellte Gary L. Miller[13] einen deterministischen Algorithmus vor, der auf der Idee beruhte, die Folge $b_0, \ldots, b_k$ zu betrachten. Dieser testete die kleinsten $O((\log N)^2)$ Elemente $a \in \{2, 3, \ldots, N-1\}$ auf die Eigenschaft, MR-Zeugen zu sein. Der Algorithmus hat polynomielle Laufzeit und liefert das korrekte Ergebnis, wenn die „erweiterte Riemannsche Vermutung (ERH)“ stimmt, eine berühmte Vermutung aus der Zahlentheorie, die bis heute unbewiesen ist. Um 1976 schlug Michael O. Rabin[14] vor, Millers Algorithmus so zu modifizieren, dass die zu testende Zahl a zufällig gewählt wird. Er und (unabhängig) Louis Monier[15] bewiesen dann, dass dieser Algorithmus konstante Fehlerwahrscheinlichkeit kleiner als $\frac{1}{4}$ hat. Der Algorithmus heißt heute allgemein der ***Miller-Rabin-Test***. Es sollte noch erwähnt werden, dass der ***Solovay-Strassen-Test***[16], der ein ähnliches Verhalten wie der Miller-Rabin-Test hat, aber nicht ganz so effizient ist, (1974 eingereicht und) 1977 publiziert wurde.

5.2.5 Fehlerschranke für den Miller-Rabin-Test

Wir müssen nun noch die Fehlerwahrscheinlichkeit des Miller-Rabin-Tests für den Fall analysieren, dass N eine zusammengesetzte (ungerade) Zahl ist. Dies wird also für diesen Abschnitt angenommen.

Wir beweisen, dass die Wahrscheinlichkeit, dass der Miller-Rabin-Test die (unerwünschte) Antwort 0 gibt, kleiner als $\frac{1}{2}$ ist. Dazu wollen wir ähnlich vorgehen wie bei der Analyse des Fermat-Tests für Nicht-Carmichael-Zahlen (Beweis von Satz 5.2.4). Dort haben wir gezeigt, dass die F-Lügner für solche N höchstens die Hälfte von $\mathbb{Z}_N^*$ ausmachen.

Wenn N keine Carmichael-Zahl ist, ist dies leicht: Nach Lemma 5.2.10(a) gilt $L_N^{\mathrm{MR}} \subseteq L_N^{\mathrm{F}}$, und wir können den Beweis von Satz 5.2.4 verwenden.

Es bleibt der Fall, dass N eine Carmichael-Zahl ist. Unser Plan ist, eine Menge B_N mit $L_N^{\mathrm{MR}} \subseteq B_N \subseteq \mathbb{Z}_N^*$ zu definieren, die höchstens die Hälfte der Elemente von $\mathbb{Z}_N^*$ enthält.

[13] Gary L. Miller, amer. Informatiker, Paris-Kanellakis-Preis 2003.

[14] Michael O. Rabin (*1931), israelischer Mathematiker und Informatiker, Turing Award 1976.

[15] Louis Monier, frz.-amer. Informatiker, Mitbegründer der AltaVista-Suchmaschine

[16] Robert M. Solovay (*1938), amer. Mathematiker; und Volker Strassen (*1936), deutscher Mathematiker, Paris-Kanellakis-Preis 2003, Knuth Prize 2008.

a	b_0	b_1	$\cdots$		b_{i_0}		$\cdots$	b_{k-1}	b_k
$a_1 = 1$	1	1	$\cdots$	1	1	1	$\cdots$	1	1
$a_2 = N-1$	$N-1$	1	$\cdots$	1	1	1	$\cdots$	1	1
a_3	$*$	$*$	$\cdots$	$*$	$N-1$	1	$\cdots$	1	1
a_4	$*$	$*$	$\cdots$	$N-1$	1	1	$\cdots$	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	1	1
a	a^u	$a^{u\cdot 2^1}$	$\cdots$	$a^{u\cdot 2^{i_0-1}}$	$a^{u\cdot 2^{i_0}}$	$a^{u\cdot 2^{i_0+1}}$	$\cdots$	$a^{u\cdot 2^{k-1}}$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$a_\#$	$*$	$*$	$\cdots$	$*$	$N-1$	1	$\cdots$	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$a_?$	$*?$	$*?$	$\cdots$	$*?$	$*?$	$\cdots$	$\cdots$	$\cdots$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$a_{\varphi(N)}$	?	?	$\cdots$	?	?	?	$\cdots$	?	1

Tabelle 6: Sicht auf B_N als Teilmenge von $\mathbb{Z}_N^*$

Weil u ungerade ist, gilt

$$(N-1)^{u\cdot 2^0} \equiv (N-1)^u \equiv (-1)^u \equiv -1 \equiv N-1 \pmod{N}.$$

Sei i_0 das größte $i \in \{0, 1, \ldots, k\}$ mit der Eigenschaft, dass es einen MR-Lügner $a_\#$ mit $a_\#^{u\cdot 2^i} \bmod N = N-1$ gibt. (Wir werden dieses $a_\#$ weiter unten nochmals benutzen.) Weil N eine Carmichael-Zahl ist, gilt $a^{u\cdot 2^k} \bmod N = a^{N-1} \bmod N = 1$ für alle $a \in \mathbb{Z}_N^*$, und daher $0 \le i_0 < k$. Wir definieren:

$$B_N := \{a \in \mathbb{Z}_N^* \mid a^{u\cdot 2^{i_0}} \bmod N \in \{1, N-1\}\}. \tag{5.2.4}$$

Zur Veranschaulichung betrachte man Tabelle 6. In der dort angegebenen Matrix entspricht jede Zeile einem der $\varphi(N)$ Elemente von $\mathbb{Z}_N^*$, beginnend mit $a_1 = 1$ und $a_2 = -1 \equiv N-1$. In der Zeile für a ist die von a erzeugte Folge $b_0, \ldots, b_k$ eingetragen. Dass N eine Carmichael-Zahl ist, drückt sich dadurch aus, dass alle Einträge in der

b_k-Spalte gleich 1 sind. Der Eintrag für $a_2 = -1$ in der b_0-Spalte ist $N-1$. Spalte i_0 ist die am weitesten rechts stehende Spalte, in der es einen Eintrag $-1 \equiv N-1$ gibt, und $a_\#$ ist ein Element von $\mathbb{Z}_N^*$, das zu diesem Eintrag führt. Die Menge B_N besteht aus den Elementen von $\mathbb{Z}_N^*$, die in Spalte i_0 Eintrag 1 oder $N-1$ haben. – Wir werden nun zeigen, dass diese Menge B_N die gewünschten Eigenschaften hat.

Lemma 5.2.13

(a) $L_N^{\mathrm{MR}} \subseteq B_N$.

(b) $B_N \subsetneq \mathbb{Z}_N^*$.

(c) $|B_N| \leq \frac{1}{2}|\mathbb{Z}_N^*|$, also $\mathbf{Pr}_{a\in\{1,\ldots,N-1\}}(a \text{ ist MR-Lügner}) < \frac{1}{2}$.

Beweis: (Teil (a) ist nur die Überprüfung, dass die Definition von B_N technisch sinnvoll ist. Der eigentlich interessante Teil ist (b). Teil (c) ist dann Routine.)

(a) Sei a ein beliebiger MR-Lügner.
Fall 1: $a^u \bmod N = 1$. – Dann ist auch $a^{u\cdot 2^{i_0}} \bmod N = 1$, und daher gilt $a \in B_N$. (Die Zeile zu a in Tab. 6 sieht aus wie die von $a_1 = 1$.)
Fall 2: $a^{u\cdot 2^i} \bmod N = N-1$ für ein $i < k$. – Dann ist $0 \leq i \leq i_0$ nach der Definition von i_0. Wenn $i = i_0$ ist, haben wir direkt $a \in B_N$ (Beispiel in Tab. 6: a_3 oder $a_\#$); wenn $i < i_0$ ist, dann gilt

$$a^{u\cdot 2^{i_0}} \bmod N = (a^{u\cdot 2^i} \bmod N)^{2^{i_0-i}} \bmod N = (-1)^{2^{i_0-i}} \bmod N = 1,$$

also ebenfalls $a \in B_N$ (Beispiel in Tab. 6: a_2 oder a_4).

(b) Wir benutzen die in Lemma 5.2.7 beobachtete Eigenschaft von Carmichael-Zahlen, keine Primzahlpotenz zu sein. Nach diesem Lemma können wir $N = N_1 \cdot N_2$ schreiben, für teilerfremde ungerade Zahlen N_1, N_2. Im Folgenden werden wir diese Zerlegung und die Eigenschaften aus dem Chinesischen Restsatz (Fakt 5.1.22) benutzen.

Die grobe Idee der nun folgenden Konstruktion ist folgende: Wir haben das Element 1 mit $1^{u\cdot 2^{i_0}} \bmod N = 1$ und das Element $a_\#$ mit $a_\#^{u\cdot 2^{i_0}} \equiv -1 \pmod N$. Aus diesen beiden Elementen „basteln“ wir mit Hilfe des Chinesischen Restsatzes ein $b \in \mathbb{Z}_N^*$, das sich modulo N_1 wie 1 und modulo N_2 wie $a_\#$ verhält. Es wird sich zeigen, dass $b^{u\cdot 2^{i_0}} \bmod N \notin \{1, N-1\}$ gilt, also $b \notin B_N$ ist.

Wir führen diese Idee jetzt formal durch. Sei $x_2 = a_\# \bmod N_2$. Nach dem Chinesischen Restsatz (Fakt 5.1.22) gibt es eine eindeutig bestimmte Zahl $b \in \{0, 1, \ldots, N-1\}$,

die

$$b \equiv 1 \pmod{N_1} \quad \text{und} \quad b \equiv x_2 \pmod{N_2}$$

erfüllt. (Es folgt $b \equiv a_\# \pmod{N_2}$.) Wir zeigen, dass b in $\mathbb{Z}_N^* - B_N$ liegt.

Wir notieren, dass für beliebige $x, x' \in \mathbb{Z}$ gilt:

$$x \equiv x' \pmod{N} \qquad \Rightarrow \qquad x \equiv x' \pmod{N_i}, \quad \text{für } i = 1, 2. \tag{5.2.5}$$

(Wenn $x - x'$ durch N teilbar ist, dann auch durch N_1 und N_2.)

Beh. 1: $b \in \mathbb{Z}_N^*$.
Bew.: Offensichtlich gilt $b^{N-1} \equiv 1^{N-1} \equiv 1 \pmod{N_1}$. Weiter haben wir, modulo N_2 gerechnet:

$$b^{N-1} \equiv a_\#^{N-1} \overset{(5.2.5)}{\equiv} (a_\#^{N-1} \bmod N) \equiv 1 \pmod{N_2}.$$

Wegen der Eindeutigkeitsaussage im Chinesischen Restsatz folgt daraus $b^{N-1} \equiv 1 \pmod{N}$. Nach Lemma 5.2.2 folgt $b \in \mathbb{Z}_N^*$.

Beh. 2: $b \notin B_N$.
Bew.: Indirekt. Annahme: $b \in B_N$, d. h. $b^{u \cdot 2^{i_0}} \equiv 1 \pmod{N}$ oder $b^{u \cdot 2^{i_0}} \equiv -1 \pmod{N}$.

1. *Fall*: $b^{u \cdot 2^{i_0}} \equiv 1 \pmod{N}$. – Mit (5.2.5) folgt $b^{u \cdot 2^{i_0}} \equiv 1 \pmod{N_2}$. Andererseits gilt

$$b^{u \cdot 2^{i_0}} \equiv x_2^{u \cdot 2^{i_0}} \equiv a_\#^{u \cdot 2^{i_0}} \overset{(5.2.5)}{\equiv} (a_\#^{u \cdot 2^{i_0}} \bmod N) \equiv N - 1 \equiv -1 \pmod{N_2},$$

ein Widerspruch, weil N_2 ungerade ist.

2. *Fall*: $b^{u \cdot 2^{i_0}} \equiv -1 \pmod{N}$. – Mit (5.2.5) folgt $b^{u \cdot 2^{i_0}} \equiv -1 \pmod{N_1}$. Andererseits gilt

$$b^{u \cdot 2^{i_0}} \equiv 1^{u \cdot 2^{i_0}} \equiv 1 \pmod{N_1},$$

ebenfalls ein Widerspruch, weil N_1 ungerade ist.

(c) Wir verwenden die in (b) konstruierte Zahl $b \in \mathbb{Z}_N^* - B_N$. Wie im Beweis von Satz 5.2.4 betrachtet man die injektive Funktion $g_b \colon B_N \ni a \mapsto ba \bmod N \in \mathbb{Z}_N^*$. Es gilt $g_b(a) \notin B_N$ für jedes $a \in B_N$, denn

$$g_b(a)^{u \cdot 2^{i_0}} \bmod N = (b^{u \cdot 2^{i_0}} \bmod N)(a^{u \cdot 2^{i_0}}) \bmod N \in \{b^{u \cdot 2^{i_0}} \bmod N, (N - b^{u \cdot 2^{i_0}}) \bmod N\}.$$

Daraus ergibt sich sofort $|B_N| \leq \frac{1}{2}|\mathbb{Z}_N^*|$. □

Wir haben also eine Schranke von $\frac{1}{2}$ für die Irrtumswahrscheinlichkeit im Miller-Rabin-Algorithmus bewiesen. Eine etwas kompliziertere Analyse zeigt, dass sogar die Fehlerschranke $\frac{1}{4}$ gilt; man kann auch zeigen, dass es zusammengesetzte Zahlen N gibt (zum Beispiel $703 = 19 \cdot 37$), bei denen eine Fehlerwahrscheinlichkeit von fast $\frac{1}{4}$ tatsächlich auftritt. Durch ℓ-fache Wiederholung, ebenso wie in Algorithmus 5.2.5, kann man die Fehlerschranke auf $4^{-\ell}$ reduzieren. Wir kürzen den Miller-Rabin-Test mit „MiRa-Test“ ab. Man beachte, dass bei jedem neuen Aufruf eine neue Zufallszahl a gewählt wird.

Algorithmus 5.2.14 ***Iterierter Miller-Rabin-Test***
EINGABE: Ungerade Zahl $N \geq 3$, eine Zahl $\ell \geq 1$.
METHODE:

```
1     repeat ℓ times
2         if MiRa-Test(N) = 1 then return 1;
3     return 0;
```

Diesen Test wollen wir kurz IterMiRa(N, ℓ) nennen. – Wir erhalten zusammenfassend:

Proposition 5.2.15
Algorithmus 5.2.14 benötigt $O(\ell \log N)$ arithmetische Operationen auf Zahlen, die kleiner als N^2 sind, und $O(\ell(\log N)^3)$ Bitoperationen. Wenn N eine Primzahl ist, ist die Ausgabe 0, wenn N zusammengesetzt ist, ist die Wahrscheinlichkeit, dass 0 ausgegeben wird, kleiner als $4^{-\ell}$.

5.3 Die Erzeugung von Primzahlen

Für kryptographische Anwendungen (zum Beispiel für die Erzeugung von Schlüsselpaaren für das RSA-Public-Key-Kryptosystem) werden vielziffrige Primzahlen benötigt. Eine typische Aufgabe in diesem Zusammenhang könnte lauten:

Finde eine zufällige Primzahl mit n Bits!

Dabei hängt n von der Anwendung ab; wir können uns z. B. $n = 512$ oder $n = 2048$ vorstellen.

Man erinnere sich an Abschnitt 5.1.4 mit dem Primzahlsatz 5.1.31 und den Abschätzungen von Dusart, die angeben, wie viele Primzahlen es in $[2^{n-1}, 2^n)$ gibt.

Ein naheliegender Ansatz zur Erzeugung einer zufälligen Primzahl in $[2^{n-1}, 2^n)$ ist folgender: Man wählt wiederholt eine (ungerade) Zahl N mit genau n Bits (erstes und letztes Bit ist 1, die anderen $n-2$ Bits sind zufällig) und wendet auf N den Miller-Rabin-Test an. Dies wiederholt man, bis eine Zahl gefunden wurde, für die der Test Ausgabe 0 liefert.

Algorithmus 5.3.1 ***GetPrime – Randomisierte Primzahlerzeugung***
EINGABE: Zahl $n \geq 9$, Zahl $\ell \geq 1$ // Ziffernzahl und Zuverlässigkeitsparameter
METHODE:
1 **repeat**
2 $\mathtt{N} \leftarrow$ zufällige ungerade Zahl in $[2^{n-1}, 2^n)$;
3 **until** IterMiRa$(\mathtt{N}, \ell) = 0$; // Algorithmus 5.2.14
4 **return** N.

Die Ausgabe ist korrekt, wenn er eine Primzahl zurückgibt, ein Fehler tritt auf, wenn eine zusammengesetzte Zahl zurückgegeben wird. Auf den ersten Blick könnte man meinen (aufgrund von Proposition 5.2.15), dass die Fehlerwahrscheinlichkeit höchstens $1/4^\ell$ ist – eben die Fehlerwahrscheinlichkeit des iterierten MiRa-Tests. Wir werden sehen, dass wir auf der Basis der Analyse des Miller-Rabin-Tests nur ein etwas schwächeres Ergebnis bekommen.[17]

Wir definieren: $\Pi_n := \{p \in [2^{n-1}, 2^n) \mid p \text{ ist Primzahl}\}$.

[17] Tatsächlich ist die Fehlerwahrscheinlichkeit durch $1/4^\ell$ beschränkt, für „genügend große“ n. Dies kann man aber nur durch fortgeschrittene zahlentheoretische Untersuchungen über das Verhalten einer zufälligen ungeraden Zahl beim Miller-Rabin-Test beweisen [P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, C. Pomerance: The generation of random numbers that are probably prime. *J. Cryptology* **1**(1): 53-64 (1988)].

Satz 5.3.2
Bei der Anwendung von Algorithmus 5.3.1 auf n und ℓ gilt:

(a) Für jedes $p \in \Pi_n$ gilt:

$$\mathbf{Pr}(\text{GetPrime}(n,\ell) = p \mid \text{GetPrime}(n,\ell) \in \Pi_n) = \frac{1}{|\Pi_n|}.$$

In Worten: Jede Primzahl in $[2^{n-1}, 2^n)$ hat dieselbe Wahrscheinlichkeit, als Ergebnis zu erscheinen.

(b) Für die Irrtumswahrscheinlichkeit gilt:[a]

$$\mathbf{Pr}(\text{GetPrime}(n,\ell) \notin \Pi_n) \leq \frac{5}{12} \cdot \frac{n}{4^\ell} = O\left(\frac{n}{4^\ell}\right).$$

(c) Die erwartete Rundenzahl ist $\leq \frac{5}{12}n$, der erwartete Aufwand ist $O(\ell n^2)$ arithmetische Operationen und $O(\ell n^4)$ Bitoperationen.

[a] Achtung: Nicht $\leq 1/4^\ell$, wie eventuell naiv vermutet. Um den Faktor n im Zähler zu neutralisieren, benötigt man eigentlich $\ell \geq \log_4 n$.

Beweis: (a) Eine Primzahl wird als Resultat genau dann geliefert, wenn keine zusammengesetzte Zahl fälschlicherweise vom Miller-Rabin-Test akzeptiert wird, bevor (in Zeile 2) die erste echte Primzahl gewählt wird. Jede der Primzahlen in $[2^{n-1}, 2^n)$ hat dieselbe Wahrscheinlichkeit, diese erste gewählte Primzahl zu sein.

(b) Mit der Abschätzung $|\Pi_n| \geq \frac{6}{5} \cdot 2^{n-1}/n$ von Seite (5.1.4), die aus den Dusart-

Schranken folgt, erhalten wir:

$$\begin{aligned}
&\mathbf{Pr}(\text{GetPrime}(n,\ell) \notin \Pi_n) \\
&= \mathbf{Pr}\Big(\exists i \geq 1\text{: in Runden } j = 1,\ldots,i-1 \text{ wird als } N \text{ eine} \\
&\qquad\qquad \text{zusammengesetzte Zahl gewählt } \textit{und erkannt } \wedge \\
&\qquad\qquad \text{in Runde } i \text{ wird eine zusammengesetzte Zahl } N \text{ gewählt} \\
&\qquad\qquad \text{und der iterierte MiRa-Test auf } N \text{ liefert } 0\Big) \\
&\leq \sum_{i\geq 1} \left(1 - \frac{|\Pi_n|}{2^{n-2}}\right)^{i-1} \cdot \frac{1}{4^\ell} \\
&= \frac{2^{n-2}}{|\Pi_n|} \cdot \frac{1}{4^\ell} \\
&\leq \frac{2^{n-2} \cdot n}{\frac{6}{5} \cdot 2^{n-1} 4^\ell} = \frac{5n}{12 \cdot 4^\ell} = O\left(\frac{n}{4^\ell}\right).
\end{aligned}$$

(c) Da man in jeder Runde mit Wahrscheinlichkeit mindestens $\frac{|\Pi_n|}{2^{n-2}}$ eine Primzahl wählt, ist die erwartete Rundenzahl nicht größer als

$$\frac{2^{n-2}}{|\Pi_n|} \leq \frac{2^{n-2}}{\frac{6}{5} \cdot 2^{n-1}/n} = \frac{5n}{12},$$

nach der Dusart-Abschätzung für $|\Pi_n|$. Eine Runde kostet $O(n\ell)$ arithmetische Operationen und $O(n^3\ell)$ Bitoperationen. □

Bemerkung: Bei der Erzeugung zufälliger Primzahlen mit n Bits für kryptographische Zwecke wird man aus Effizienzgründen nicht Algorithmus 5.3.1 mit $\ell = \Theta(n)$ anwenden, der $\Omega(\ell n^2)$ Multiplikationen benötigt, sondern jede gewählte Zahl N zunächst auf Teilbarkeit durch Primzahlen z. B. kleiner als n testen (da viele zusammengesetzte Zahlen N kleine Teiler haben, werden diese sehr rasch als zusammengesetzt erkannt) und nachher eine Kombination zweier verschiedener Primzahltests (z. B. Miller-Rabin und „Lucas Strong Probable Prime Test“) jeweils mit nur einer oder zwei Iterationen anwenden. Dies erfordert nur $O(n)$ „volle“ Multiplikationen. Die Dichte der zusammengesetzten Zahlen, die von einem solchen Test nicht erkannt werden, ist als sehr gering einzuschätzen. Über eine klassische experimentelle Untersuchung hierzu berichtet die kurze Notiz

`people.csail.mit.edu/rivest/Rivest-FindingFourMillionLargeRandomPrimes.ps`

von Ron Rivest (bekannt von „RSA“ und von „Cormen, Leiserson, Rivest und Stein“).

5.4 Quadratische Reste und Quadratwurzeln modulo p

Wir betrachten hier den Begriff des quadratischen Rests und der Quadratwurzeln modulo p, für Primzahlen p.

Beispiel: Für $p = 13$ und $p = 19$ betrachten wir die Quadrate der Zahlen in $\mathbb{Z}_p^*$.

a	1	2	3	4	5	6	7	8	9	10	11	12
a^2	1	4	9	16	25	36	49	64	81	100	121	144
$a^2 \bmod 13$	1	4	9	3	12	10	10	12	3	9	4	1

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$a^2 \bmod 19$	1	4	9	16	6	17	11	7	5	5	7	11	17	6	16	9	4	1

Wir bemerken, dass in beiden Fällen jede Zahl entweder keinmal oder zweimal als Quadrat vorkommt, dass infolgedessen genau $\frac{1}{2}(p-1)$ Zahlen in $\mathbb{Z}_p^*$ als Quadratzahlen auftauchen. Im Fall $p = 13$ ist $12 = -1 \bmod 13$ eine Quadratzahl, im Fall $p = 19$ ist $18 = -1 \bmod 19$ keine Quadratzahl. Es stellt sich die Frage, wie man zu gegebenem p und a ermitteln kann, ob a in $\mathbb{Z}_p$ ein Quadrat ist (einfach, wenn man weiß, wie es geht) und wie man gegebenenfalls ein c mit $c^2 \bmod p = a$ finden kann (eventuell nicht ganz so einfach).

Bemerkung: In $\mathbb{Z}_p$ spielt $p - 1$ die Rolle von -1, dem additiven Inversen von 1.

Definition 5.4.1
Für eine ungerade Primzahl p sei

$$\mathrm{QR}_p := \{a \in \mathbb{Z}_p^* \mid \exists b \in \mathbb{Z}_p^* \colon b^2 \bmod p = a\}$$

die Menge der „quadratischen Reste“ in $\mathbb{Z}_p^*$.

Wir stellen einige auch für sich wichtige Hilfsaussagen bereit.

Fakt 5.4.2
Wenn p eine ungerade Primzahl ist, dann gilt:

(a) Wenn $d \geq 1$ ist, dann hat ein Polynom $f(X) = X^d + a_{d-1}X^{d-1} + \cdots + a_1X + a_0$ in $\mathbb{Z}_p$ höchstens d Nullstellen.

(b) Für $s \geq 1$ gibt es in $\mathbb{Z}_p^*$ höchstens s Elemente b mit $b^s \bmod p = 1$.

Beweis: (a) Dies ist eine Tatsache, die in Körpern allgemein gilt. Man kann sich recht leicht überlegen, dass man für Nullstellen $b_1, \ldots, b_k$ aus $f(X)$ den Faktor $(X - b_1) \cdots (X - b_k)$ ausklammern kann, d. h. $f(X) = (X - b_1) \cdots (X - b_k) \cdot g(X)$ schreiben kann. Dann kann k nicht größer als der Grad d von $f(X)$ sein.
(b) Wenn $b_1, \ldots, b_{s+1}$ verschiedene Lösungen wären, hätte das Polynom $X^s - 1$ Grad s, aber $> s$ Nullstellen, was (a) widerspricht. □

Wir verallgemeinern Lemma 5.2.8 wie folgt:

Lemma 5.4.3
Wenn p eine ungerade Primzahl ist und $a \in \mathrm{QR}_p$, dann gibt es genau zwei Zahlen $b \in \mathbb{Z}_p$ mit $b^2 \bmod p = a$. Daher gilt: $|\mathrm{QR}_p| = \frac{1}{2}(p-1)$.

Beweis: Nach Definition von QR_p gibt es ein solches b. Dann gilt auch $(p-b)^2 \bmod p = (p^2 - 2pb + b^2) \bmod p = b^2 \bmod p = a$. Weil p ungerade ist, gilt $b - (p - b) \bmod p = 2b \bmod p \neq 0$, also $b \neq p - b$. Weitere Quadratwurzeln von a kann es nicht geben, da sonst das Polynom $X^2 - a$ vom Grad 2 mehr als zwei Nullstellen hätte, was nach Fakt 5.4.2(a) nicht möglich ist. □

Die folgende Behauptung erlaubt es, effizient zu testen, ob a ein quadratischer Rest modulo p ist.

Proposition 5.4.4 ***Eulersches Kriterium***
Für jede ungerade Primzahl p und jedes $a \in \mathbb{Z}_p^*$ gilt:

(a) $a^{(p-1)/2} \bmod p \in \{1, -1\}$.

(b) $a \in \mathrm{QR}_p \;\Leftrightarrow\; a^{(p-1)/2} \bmod p = 1$.

Beweis von Proposition 5.4.4:

(a) Nach dem kleinen Satz von Fermat (Fakt 5.1.18) gilt $a^{p-1} \bmod p = 1$ für jedes $a \in \mathbb{Z}_p^*$. Daraus folgt

$$\left(a^{(p-1)/2} \bmod p\right)^2 \bmod p = 1 \text{ für jedes } a \in \mathbb{Z}_p^*.$$

Mit Lemma 5.4.3 folgt, dass $a^{(p-1)/2} \bmod p$ eine der beiden Quadratwurzeln von 1 ist, also gleich 1 oder gleich -1 sein muss.

(b) Wir definieren $A := \{a \in \mathbb{Z}_p^* \mid a^{(p-1)/2} \bmod p = 1\}$. Unser Ziel ist zu zeigen, dass $\mathrm{QR}_p = A$ ist. Eine Inklusion ist leicht einzusehen:

Beh. 1: $\mathrm{QR}_p \subseteq A$. (Wenn $a \in \mathrm{QR}_p$, also $a = b^2 \bmod p$ für ein b, dann gilt

$$a^{(p-1)/2} \bmod p = (b^2 \bmod p)^{(p-1)/2} \bmod p = b^{p-1} \bmod p = 1,$$

nach dem kleinen Satz von Fermat. Also ist $a \in A$.)

Um die Gleichheit zu zeigen, benutzen wir ein Kardinalitätsargument.

Beh. 2: $|\mathrm{QR}_p| = \frac{1}{2}(p-1)$. Dies gilt, weil nach Lemma 5.4.3 jeder quadratische Rest genau zwei verschiedene Quadratwurzeln hat und natürlich jede der $p-1$ Zahlen b in $\mathbb{Z}_p^*$ Quadratwurzel von $b^2 \bmod p$ ist.

Beh. 3: $|A| \leq \frac{1}{2}(p-1)$. (Nach Fakt 5.4.2 (mit $s = \frac{1}{2}(p-1)$) gilt, dass höchstens $\frac{1}{2}(p-1)$ viele Elemente von $\mathbb{Z}_p^*$ die Gleichung $a^{(p-1)/2} \bmod p = 1$ erfüllen.)

Aus Beh. 1, 2 und 3 folgt $\mathrm{QR}_p = A$, wie gewünscht. □

Mit diesem Kriterium kann man mit einer schnellen Exponentiation testen, ob eine Zahl a quadratischer Rest modulo p ist oder nicht.

Algorithmus 5.4.5 ***Quadratzahltest modulo p***
Input: Primzahl p, Zahl a mit $1 < a < p$
Methode:
(1) Berechne $z = a^{(p-1)/2} \bmod p$; // Schnelle Exponentiation
(2) **if** $z = 1$ **then return** „Quadrat“ **else return** „Nichtquadrat“.

Wir rechnen versuchsweise in $\mathbb{Z}_{19}$, für $18 \notin \mathrm{QR}_{19}$ und $17 \in \mathrm{QR}_{19}$:

$18^9 \bmod 19 = (((-1)^8 \bmod 19) \cdot 18) \bmod 19 = 18 \neq 1$;
$17^9 \bmod 19 = (((-2)^8 \bmod 19) \cdot (-2)) \bmod 19 = (256 \cdot (-2)) \bmod 19 = (9 \cdot (-2)) \bmod 19 = 1$.

Wir notieren eine leichte Folgerung aus dem Eulerschen Kriterium:

Korollar 5.4.6
Sei p eine ungerade Primzahl. Dann gilt:

(a) Für $a, b \in \mathbb{Z}_p^*$ und das Produkt $c = ab \bmod p$ gilt:

$$\begin{aligned} a, b \in \mathrm{QR}_p &\Rightarrow c \in \mathrm{QR}_p; \\ a, b \notin \mathrm{QR}_p &\Rightarrow c \in \mathrm{QR}_p; \\ a \in \mathrm{QR}_p, b \notin \mathrm{QR}_p &\Rightarrow c \notin \mathrm{QR}_p. \end{aligned}$$

(b) Für $a \in \mathbb{Z}_p^*$ und a^{-1} in $\mathbb{Z}_p^*$, das multiplikative Inverse von a, gilt:

$$a \in \mathrm{QR}_p \quad \Leftrightarrow \quad a^{-1} \in \mathrm{QR}_p.$$

((a) folgt durch Anwendung von Exponentiationsregeln und Proposition 5.4.4; (b) folgt aus $a \cdot a^{-1} = 1 \in \mathrm{QR}_p$ und (a).)

Wenn das so einfach geht – kann man dann zu einer Quadratzahl a auch leicht eine Quadratwurzel b modulo p berechnen?

Erstaunlicherweise gibt es hier zwei sehr verschiedene Fälle. Wir betrachten das Beispiel $p = 19$. Die Quadrate in $\mathbb{Z}_p^*$ sind die neun Zahlen $1, 4, 5, 6, 7, 9, 11, 16, 17$. Wir berechnen $a^5 \bmod 19$ für diese Quadratzahlen und quadrieren das Resultat:

a	1	4	5	6	7	9	11	16	17
$b = a^5 \bmod 19$	1	17	9	5	11	16	7	4	6
$b^2 \bmod 19$	1	4	5	6	7	9	11	16	17

Wir sehen: $a^5 \bmod 19$ ist für jede der Quadratzahlen a eine Quadratwurzel. (Die andere ist dann natürlich $(19 - a^5) \bmod 19$.) Dies ist kein Zufall. Die Zahl 5 ergibt sich aus $p = 19$ als $\frac{1}{4}(p+1)$. Diese Operation ist für alle Primzahlen möglich, die um 1 unter einem Vielfachen von 4 liegen.

Proposition 5.4.7
Wenn p eine Primzahl ist derart, dass $p + 1$ durch 4 teilbar ist, dann ist für jedes $a \in \mathrm{QR}_p$ die Zahl $a^{(p+1)/4} \bmod p$ eine Quadratwurzel von a.

Beweis:

$$((a^{(p+1)/4}) \bmod p)^2 \bmod p = (a^{(p+1)/2}) \bmod p = (((a^{(p-1)/2}) \bmod p) \cdot a) \bmod p = a,$$

wobei wir Proposition 5.4.4(b) benutzt haben. □

Mit anderen Worten: Für die Primzahlen $p \in \{3, 7, 11, 19, 23, 31, 43, 47, 59, \dots\}$ läuft das Ziehen von Quadratwurzeln auf eine schnelle Exponentiation mit Exponent $\frac{1}{4}(p+1)$ hinaus. (Bitkomplexität $O((\log p)^3)$.)

Wie steht es mit den anderen Primzahlen $5, 13, 17, 29, 37, 41, 53, \dots$? Gibt es einen ähnlich effizienten Algorithmus zum Finden von Quadratwurzeln? Hier gibt es nochmals zwei Unterfälle. Wenn man leicht einen beliebigen nichtquadratischen Rest modulo p finden kann, dann kann man mit Hilfe eines deterministischen Algorithmus, der in Anhang C beschrieben ist, mit $O((\log p)^2)$ arithmetischen Operationen Quadratwurzeln finden. Dies ist der Fall, wenn $p \equiv 5 \pmod 8$, also für $5, 13, 29, 37, 53, \dots$, weil dann stets $2 \notin \mathrm{QR}_p$ gilt (Ergebnis der Zahlentheorie im Zusammenhang mit dem „Quadratischen Reziprozitätsgesetz“). Es bleiben die Primzahlen p mit $p \equiv 1 \pmod 8$, beispielsweise $17, 41, 73, \dots, 257, \dots$. Interessanterweise kennt man für solche Primzahlen keinen *deterministischen* Algorithmus, der Quadratwurzeln modulo p berechnet und dazu Zeit $(\log p)^{O(1)}$ braucht. Wir betrachten hier einen ***randomisierten*** Algorithmus[18] für diese Aufgabe. Er funktioniert für alle Primzahlen $p \geq 3$; angesichts Prop. 5.4.7 wird man ihn aber nur verwenden, wenn $p \equiv 1 \pmod 4$ ist.

Idee: Gegeben ein $a \in \mathrm{QR}_p$, bezeichne eine Quadratwurzel von a mit $\diamond$. Die andere Quadratwurzel von a ist dann $-\diamond$. Rechne mit $\diamond$ (in $\mathbb{Z}_p$) wie mit einer Unbekannten. Es entstehen Ausdrücke $\alpha + \beta \cdot \diamond$, die man addieren, subtrahieren und multiplizieren kann. Höhere Potenzen von $\diamond$ entstehen nicht, wenn $\diamond^2$ durch a ersetzt wird. Es gelten folgende Regeln:

$$\begin{aligned} (\alpha + \beta \cdot \diamond) \pm (\gamma + \delta \cdot \diamond) &= (\alpha \pm \gamma) + (\beta \pm \delta) \cdot \diamond; \\ (\alpha + \beta \cdot \diamond) \cdot (\gamma + \delta \cdot \diamond) &= (\alpha\gamma + \beta\delta \cdot a) + (\alpha\delta + \beta\gamma) \cdot \diamond. \end{aligned} \tag{5.4.6}$$

Wenn man diese Operationen *implementieren* will, stellt man ein Element $\alpha + \beta \cdot \diamond$ als (α, β) dar und rechnet wie in (5.4.6). Mit den in (5.4.6) angegebenen Operationen bildet die Menge der Paare (a, b) in $\mathbb{Z}_p \times \mathbb{Z}_p$ einen Ring mit 1. Insbesondere besitzt die Multiplikation ein neutrales Element, nämlich $(1, 0)$, und die Multiplikation ist

[18] Der Algorithmus wurde 1907 von dem italienischen Mathematiker Michele Cipolla (1880–1947) gefunden.

kommutativ und assoziativ (prüft man leicht nach). Daher kann man das schnelle Exponentiationsverfahren aus Algorithmus 5.1.19 anwenden, um „symbolisch“ die Potenz $(\alpha+\beta\cdot\diamond)^i$ in $\log i$ Runden mit jeweils höchstens 10 Multiplikationen modulo p zu berechnen.

Wesentlich ist die folgende Grundeigenschaft. Wenn man in der abstrakten Notation eine Gleichheit ausgerechnet hat, dann gilt sie für beide Quadratwurzeln von a:

Wenn sich mit den Regeln von (5.4.6) $(\alpha+\beta\cdot\diamond)^i$ zu $\gamma+\delta\cdot\diamond$ berechnet und wenn $w^2 \bmod p = a$ ist, dann gilt $(\alpha+\beta\cdot w)^i = \gamma+\delta\cdot w$.

(Man setzt w für $\diamond$ ein und verfolgt die symbolische Rechnung, aber nun in $\mathbb{Z}_p$.)

Algorithmus 5.4.8 ***Algorithmus von Cipolla: Quadratwurzeln***

Input: Primzahl $p \geq 3$, $a \in \{1,\ldots,p-1\}$ mit $a^{(p-1)/2} \bmod p = 1$.

Methode:

```
      //  1–4: Finde b mit b^2 = a oder b^2 − a ∉ QR_p:
1     repeat
2       Wähle b uniform zufällig aus {1,...,p−1};
3       if b^2 mod p = a then return {b,−b};  //  großes Glück gehabt!
4     until (b^2 − a)^((p−1)/2) mod p = p − 1;  //  b^2 − a ist nichtquadratischer Rest
5     (c + d·◇) ← (b + 1·◇)^((p−1)/2);
6           //  Schnelle Exponentiation mit Ausdrücken α + β·◇, Regeln (5.4.6)
7     u ← d^(−1) mod p  //  erweiterter Euklidischer Algorithmus
8     return {u,−u}.
```

Korrektheit: w sei eine Quadratwurzel von a; die andere ist dann $-w$. Nach der obigen Vorüberlegung gilt (in $\mathbb{Z}_p$ gerechnet), weil sowohl $w^2 = a$ als auch $(-w)^2 = a$ gilt:

$$\begin{aligned} c+dw &= (b+w)^{(p-1)/2} \in \{1,-1\} \text{ und} \\ c-dw &= (b-w)^{(p-1)/2} \in \{1,-1\}. \end{aligned} \tag{5.4.7}$$

Wir addieren diese beiden Gleichungen und erhalten:

$$2\cdot c = (b+w)^{(p-1)/2} + (b-w)^{(p-1)/2}. \tag{5.4.8}$$

Andererseits ist $(b^2 - a)^{(p-1)/2} = -1$ (wegen Zeile 4 im Algorithmus). Wir setzen $a = w^2$ ein und erhalten

$$-1 = (b^2 - w^2)^{(p-1)/2} = (b + w)^{(p-1)/2} \cdot (b - w)^{(p-1)/2}.$$

Wegen (5.4.7) muss einer der Faktoren gleich 1 und der andere gleich -1 sein:

$$\{(b + w)^{(p-1)/2}, (b - w)^{(p-1)/2}\} = \{1, -1\}.$$

Mit (5.4.8) folgt $2 \cdot c = 0$, und damit $c = 0$, weil in $\mathbb{Z}_p$ gilt, dass $2 = 1 + 1 \neq 0$ ist. Subtraktion der Gleichungen in (5.4.7) ergibt

$$2dw = (b + w)^{(p-1)/2} - (b - w)^{(p-1)/2}.$$

Die beiden Terme in der Differenz sind 1 und -1 (in irgendeiner Reihenfolge), also ist $2dw \in \{2, -2\}$, oder (wieder weil $2 \neq 0$):

$$dw \in \{1, -1\}, \text{ d.h.: } w \in \{d^{-1}, -d^{-1}\},$$

wobei das multiplikative Inverse in $\mathbb{Z}_p$ berechnet wird. Weil nach der Ausgangssituation w und $-w$ die Quadratwurzeln von a sind, sind auf jeden Fall $u = d^{-1}$ und $-u = -d^{-1}$ diese Wurzeln.

Rechenzeit: In der Laufzeit des Algorithmus gibt es drei entscheidende Komponenten: (i) Das Finden eines Elements b derart, dass b Quadratwurzel von a (sehr unwahrscheinlich) oder $b^2 - a \notin \mathrm{QR}_p$ ist; (ii) die Exponentiation zum Finden von $(b + 1 \cdot \diamond)^{(p-1)/2}$ (die höchstens $\log p$ Schleifendurchläufe mit je höchstens 10 modularen Multiplikationen erfordert); (iii) der Aufwand für den erweiterten Euklidischen Algorithmus. Teile (ii) und (iii) haben Kosten, die einer festen Anzahl modularer Exponentiationen entsprechen, also $O(\log p)$ Multiplikationen und Additionen. Für Teil (i) ist zu überlegen, was die Dichte

$$\frac{|G|}{p-1} = \frac{|\{b \in \mathbb{Z}_p^* \mid b^2 = a \ \vee \ b^2 - a \notin \mathrm{QR}_p\}|}{p-1}$$

der Menge G der „guten“ Elemente in $\mathbb{Z}_p^*$ ist.

Behauptung: $|G| \geq \frac{1}{2}(p + 1)$.
Beweis der Behauptung: Die Elemente w und $-w$ sind verschiedene und jedenfalls gute Elemente. Es geht also um die Anzahl der guten Elemente in der Menge („*Nicht-Wurzeln*“)

$$NW := \mathbb{Z}_p^* - \{w, -w\}.$$

Wir definieren eine Funktion $f\colon NW \to \mathbb{Z}_p$ durch

$$f(b) := \frac{b+w}{b-w} \quad \text{(Arithmetik in } \mathbb{Z}_p\text{)}.$$

Diese Funktion ist wohldefiniert, da $b \neq w$, und sie nimmt den Wert 0 nicht an, da $b \neq -w$. Weiterhin wird der Wert 1 nicht angenommen (wenn $b+w = b-w$, wäre $2w = 0$, also $w = 0$), und der Wert -1 auch nicht (wenn $b+w = -(b-w)$, wäre $2b = 0$, also $b = 0$, was $b \in \mathbb{Z}_p^*$ widerspricht). Also gilt $f\colon NW \to \{2, \ldots, p-2\}$. Man sieht leicht, dass f injektiv ist:

$$\begin{aligned}
\frac{b_1+w}{b_1-w} = \frac{b_2+w}{b_2-w} \quad &\Rightarrow \quad (b_1+w)(b_2-w) = (b_2+w)(b_1-w) \\
&\Rightarrow \quad b_1b_2 - wb_1 + wb_2 - w^2 = b_1b_2 + wb_1 - wb_2 - w^2 \\
&\Rightarrow \quad 2w(b_2-b_1) = 0 \\
&\Rightarrow \quad b_1 = b_2,
\end{aligned}$$

wobei wir $2 \neq 0$ und $w \neq 0$ benutzt haben. Aus der Injektivität folgt, dass f eine Bijektion zwischen NW und $\{2, \ldots, p-2\}$ ist. In der letzteren Menge gibt es höchstens $(p-1)/2 - 1 = \frac{1}{2}(p-3)$ quadratische Reste (weil 1 ein quadratischer Rest ist). Wir haben:

$$b^2 - a \in \mathrm{QR}_p \quad \Leftrightarrow \quad \frac{b+w}{b-w} \in \mathrm{QR}_p.$$

(Das gilt, weil $b^2 - a = (b+w)(b-w)$ ist, also $(b+w)/(b-w) = (b^2-a)/(b-w)^2$, und nach Korollar 5.4.6 die Menge der (nicht-)quadratischen Reste in $\mathbb{Z}_p$ unter Division mit quadratischen Resten (hier $(b-w)^2$) abgeschlossen ist.) Wegen dieser Eigenschaft von f ist

$$|G| = 2 + |\{u \in NW \mid f(u) \notin \mathrm{QR}_p\}| \geq 2 + (p-3) - \frac{1}{2}(p-3) = \frac{1}{2}(p+1).$$

Damit ist die Behauptung bewiesen. – Die Wahrscheinlichkeit, in Zeile 2 ein Element von $|G|$ zu wählen, ist also mindestens $(p+1)/(2(p-1)) > \frac{1}{2}$. Damit ist die erwartete Anzahl von Versuchen, bis ein geeignetes b gefunden wird, kleiner als 2. □

Wir fassen unsere Ergebnisse im folgenden Satz zusammen:

Satz 5.4.9
Für eine Primzahl p und einen quadratischen Rest a modulo p berechnet Algorithmus 5.4.8 die beiden Quadratwurzeln von a. Die erwartete Anzahl von Runden im ersten Teil ist $O(1)$; der erwartete Aufwand ist $O((\log p)^3)$ Bitoperationen.

Bemerkung: Für Zahlen der Form $N = p \cdot q$ für verschiedene Primzahlen p, q ist kein effizienter Algorithmus zur Ermittlung von Quadratwurzeln modulo N bekannt (auch kein randomisierter!), wenn nur N und die Quadratzahl a (modulo N), nicht aber die Faktoren p und q Teil der Eingabe sind. Diese Tatsache ist die Grundlage der Sicherheit des *Rabin-Kryptosystems*.

Dieses „Public-Key-Kryptosystem“ funktioniert wie folgt. Wir nehmen vereinfachend an, dass Nachrichten Bitstrings sind, die wir als natürliche Zahlen auffassen. Alice ist der Absender, Bob der Empfänger. Bob bereitet das System vor, indem er zwei verschiedene (große) Primzahlen p und q wählt, zum Beispiel mit einem Algorithmus wie 5.3.1. Er hält p und q geheim und stellt $N = p \cdot q$ als „öffentlichen Schlüssel“ zur Verfügung. Alice kennt also N. Wenn sie Bob eine Nachricht $x < N$ schicken möchte, berechnet sie den „Kryptotext“ $y = x^2 \bmod N$ und sendet ihn (als Bitstring) an Bob. Zum Entschlüsseln von y geht Bob wie folgt vor: Er benutzt einen Algorithmus zum Ziehen von Quadratwurzeln modulo einer Primzahl, um Zahlen $u < p$ mit $u^2 \equiv y \pmod{p}$ und $v < q$ mit $v^2 \equiv y \pmod{q}$ zu berechnen. Dann berechnet er mit Hilfe der effizienten Version des Chinesischen Restsatzes die vier Zahlen $x_1, x_2, x_3, x_4 < N$, die folgende Kongruenzen erfüllen:

$$\begin{aligned}
x_1 &\equiv u \pmod{p} \text{ und } x_1 \equiv v \pmod{q},\\
x_2 &\equiv u \pmod{p} \text{ und } x_2 \equiv -v \pmod{q},\\
x_3 &\equiv -u \pmod{p} \text{ und } x_3 \equiv v \pmod{q},\\
x_4 &\equiv -u \pmod{p} \text{ und } x_4 \equiv -v \pmod{q}.
\end{aligned}$$

Eine dieser vier Zahlen (Bitstrings) ist die Nachricht x, die Alice geschickt hat. Man geht davon aus, dass Bob aufgrund von eingebauten Redundanzen die richtige Nachricht aussuchen kann.

Das Rabin-Kryptosystem ist praktisch nicht besonders wichtig, aber es hat eine interessante theoretische Eigenschaft. Wenn ein Angreifer einen effizienten Algorithmus hat (also einen mit Rechenzeit polynomiell in $\log N$), der es ihm erlaubt, aus beliebigen Kryptotexten y den zugehörigen Klartext zu ermitteln, dann kann dieser Angreifer effizient die Faktoren p und q berechnen. Da die letzte Aufgabe nach heutigem Stand des Wissens für genügend große Zahlen p und q als nicht effizient lösbar gilt (es ist kein Polynomialzeitalgorithmus für dieses Problem bekannt), nimmt man an, dass es keinen effizienten Algorithmus gibt, der das Rabin-Kyptosystem brechen kann. (Details zu diesen Überlegungen: Vorlesung „Kryptographie“.)

A Beweise und Bemerkungen für Abschnitt 5.1

Bemerkung zu Fakt 5.1.2: Wenn man es genau nimmt, sind die Elemente von $\mathbb{Z}_m$ nicht Zahlen, sondern Äquivalenzklassen

$$[a] = \{qm + a \mid q \in \mathbb{Z}\}, \text{ für } 0 \leq a < m,$$

zur Äquivalenzrelation

$$x \equiv y \pmod{m}, \quad (\text{auch: } x \equiv y\ (m) \ \text{ oder } \ x \equiv_m y\ ),$$

die durch „m ist Teiler von $x - y$“ definiert ist (vgl. Fakt. 5.1.4). Die Operationen werden auf den Repräsentanten ausgeführt, zum Beispiel gilt in $\mathbb{Z}_{11}$, dass $[4] + [7] = [11] = [0]$ ist, also sind $[4]$ und $[7]$ zueinander invers. Allgemeiner gilt in jedem $\mathbb{Z}_m$ die Gleichheit $[a] + [m - a] = [m] = [0]$, d. h., $[m - a]$ ist das additive Inverse $-[a]$ zu $[a]$. Multiplikativ gilt in $\mathbb{Z}_m$, dass $[1]$ neutrales Element ist, und dass $[1] \cdot [1] = [1]$ und $[m - 1] \cdot [m - 1] = [m(m - 2) + 1] = [1]$, also $[1]$ und $-[1] = [m - 1]$ (triviale) Quadratwurzeln der 1 sind.

Es ist bequem, die eckigen Klammern wegzulassen und für $[i]$ einfach i zu schreiben. Man muss dann nur aufpassen, dass man Ausdrücke wie „-1“ richtig interpretiert, nämlich als das additive Inverse von 1 in $\mathbb{Z}_m$, also $[m - 1]$.

Zum *Beweis* von Lemma 5.1.3: Diese Aussagen lassen sich mathematisch besser in der Sprechweise der Äquivalenzklassen verstehen. Dazu muss man zunächst ordentlich definieren: $[x] + [y] := [x + y]$ und $[x] \cdot [y] := [x \cdot y]$. (Die Summe von $[x]$ und $[y]$ in $\mathbb{Z}_m$ ist die Äquivalenzklasse von $x + y$; das Produkt von $[x]$ und $[y]$ in $\mathbb{Z}_m$ ist die Äquivalenzklasse von $x \cdot y$.) Damit diese Definition funktioniert, muss man zunächst überprüfen, dass das Ergebnis nicht von den irgendwie gewählten Repräsentanten x und y abhängt, sondern tatsächlich nur von den Klassen $[x]$ und $[y]$. (Wenn wir zum Beispiel $m = 7$ nehmen, dann sollte $[10 \cdot 17]$ dasselbe sein wie $[(-4) \cdot 3]$. Dies ist der Fall: $170 \equiv 2 \equiv -12 \pmod{7}$.) Diese Eigenschaft nennt man die *Wohldefiniertheit*:

$$x \equiv x' \wedge y \equiv y' \pmod{m} \ \Rightarrow \ x + y \equiv x' + y' \pmod{m}.$$

Diese Überprüfung ist ganz leicht: Wenn $x' = x + km$ und $y' = y + \ell m$, dann ist $x' + y' = x + y + (k + \ell)m$ und $x' \cdot y' = x \cdot y + (ky + x\ell + k\ell m)m$. Unser Lemma ist nun im Wesentlichen eine spezielle Formulierung dieser Wohldefiniertheit: Man kann beliebige Repräsentanten x und y aus $\mathbb{Z}$ oder die speziellen Repräsentanten

$x' = (x \bmod m)$ und $y' = (y \bmod m)$ aus $[m]$ benutzen. Wegen der allgemeinen Wohldefiniertheit gilt dann $[x+y] = [x'+y']$ sowie $[x \cdot y] = [x' \cdot y']$, und das heißt $x + y \equiv x' + y' \pmod m$ sowie $x \cdot y \equiv x' \cdot y' \pmod m$. Mit dem folgenden Fakt 5.1.4 ergibt sich

(i) $(x+y) \bmod m = (x'+y') \bmod m$,

(ii) $(x \cdot y) \bmod m = (x' \cdot y') \bmod m$,

wie gewünscht. □

Beweis von Fakt 5.1.4: Wenn $x \bmod m = y \bmod m$ gilt, gibt es Quotienten q und q' mit $x = qm + r$ und $y = q'm + r$, für den gemeinsamen Rest r. Damit gilt $x - y = (q - q')m$. Wenn umgekehrt $x - y = mz$ für eine ganze Zahl z und $x = qm + r$ und $y = q'm + r'$ für Quotienten q, q' und Reste r, r', dann ist $r - r' = (x-y) + (q'-q)m = (z + q' - q)m$ durch m teilbar, und $|r - r'| < m$. Daher muss $r - r' = 0$ gelten. □

Beweis von Fakt 5.1.8:

Die Korrektheit folgt aus den Vorüberlegungen. Wir diskutieren also nur die Rechenzeit. Wenn $|x| < |y|$, hat der erste Schleifendurchlauf nur den Effekt, die beiden Zahlen zu vertauschen. Wir ignorieren diesen trivialen Schleifendurchlauf. Nachher nimmt die Zahl b in `b` in jeder Runde strikt ab, also terminiert der Algorithmus. Um einzusehen, dass er sogar sehr schnell terminiert, bemerken wir Folgendes. Betrachte den Beginn eines Schleifendurchlaufs. Der Inhalt von `a` sei a, der Inhalt von `b` sei b, mit $a \geq b > 0$. Nach einem Schleifendurchlauf enthält `a` den Wert $a' = b$ und `b` den Wert $b' = a \bmod b$. Falls $b' = 0$, endet der Algorithmus. Sonst wird noch ein Durchlauf ausgeführt, an dessen Ende `a` den Wert $b' = a \bmod b$ enthält. Wir behaupten: $b' < \frac{1}{2}a$. Um dies zu beweisen, betrachten wir zwei Fälle: Wenn $b > \frac{1}{2}a$ ist, gilt $b' = a \bmod b = a - b < \frac{1}{2}a$. Wenn $b \leq \frac{1}{2}a$ ist, gilt $b' = a \bmod b < b \leq \frac{1}{2}a$. – Also wird der Wert in `a` in jeweils zwei Durchläufen mindestens halbiert. Nach dem ersten Schleifendurchlauf enthält `a` den Wert $\min\{x, y\}$. Also kann es höchstens $\log(\min\{x, y\})$ weitere Schleifendurchläufe geben; danach ist der Wert in `b` auf jeden Fall 0. □

Bemerkungen zu Lemma 5.1.9: In (b), also auch im Spezialfall (a), ergibt sich die Existenz von s und t direkt aus dem erweiterten Euklidischen Algorithmus. Interessant ist noch die folgende Umkehrung von (a): Wenn wir $1 = sx + ty$ schreiben

können, für ganze Zahlen s, t, dann sind x und y teilerfremd. Der Grund ist einfach: Wenn $c \mid x$ und $c \mid y$ gilt, dann folgt $c \mid sx + ty$, also $c \mid 1$. Daher ist $c \in \{-1, 1\}$.

Beweis von Fakt 5.1.15: (i) Assoziativgesetz und Kommutativgesetz für Multiplikation modulo m gelten in $\mathbb{Z}_m$ allgemein. (ii) 1 ist neutrales Element sogar in $\mathbb{Z}_m$. (iii) Wir zeigen, dass $\mathbb{Z}_m^*$ unter Multiplikation abgeschlossen ist. Seien also $a, b \in \mathbb{Z}_m^*$, d. h. $\mathrm{ggT}(a, m) = \mathrm{ggT}(b, m) = 1$. Nach Lemma 5.1.9(a) kann man $1 = xa + um = yb + vm$ schreiben, für ganze Zahlen x, u, y, v. Dies ergibt

$$1 = (xa + um)(yb + vm) = (xy)(ab) + (xav + uyb + umv)m.$$

Daraus folgt, dass ab und m teilerfremd sind, also $ab \bmod m \in \mathbb{Z}_m^*$. (iv) Nach Lemma 5.1.13 hat jedes Element von $\mathbb{Z}_m^*$ ein multiplikatives Inverses $b \in \mathbb{Z}_m$. Weil natürlich a auch das Inverse von b ist, folgt nach Lemma 5.1.13 auch, dass $b \in \mathbb{Z}_m^*$ ist. □

Beweis von Prop. 5.1.17: Die zweite Äquivalenz ist klar: Der Ring $\mathbb{Z}_m$ ist nach Definition ein Körper genau dann wenn jedes Element von $\mathbb{Z}_m - \{0\}$ ein multiplikatives Inverses besitzt.

Erste Äquivalenz:

„⇒“: Sei m eine Primzahl. Dann ist jedes $a \in \{1, \ldots, m-1\} = \mathbb{Z}_m - \{0\}$ zu m teilerfremd; nach der Definition folgt $\mathbb{Z}_m^* = \mathbb{Z}_m - \{0\}$.

„⇐“: Sei m eine zusammengesetzte Zahl, etwa durch k mit $2 \leq k < m$ teilbar. Dann ist $k = \mathrm{ggT}(k, m) > 1$, also hat k nach Lemma 5.1.13 kein multiplikatives Inverses modulo m. (Man sieht auch direkt, dass $kb \bmod m = kb - qm$ durch k teilbar ist, also für kein b gleich 1 sein kann.) □

B Mehr über Carmichaelzahlen

Wie kann man einsehen, dass $561 = 3 \cdot 11 \cdot 17$, $1105 = 5 \cdot 13 \cdot 17$ und $1729 = 7 \cdot 13 \cdot 19$ Carmichaelzahlen sind? In diesem Abschnitt stellen wir (nur für Interessierte) ein einfaches Kriterium bereit, das es erlaubt, Zahlen auf die Eigenschaft „Carmichaelzahl“ zu testen, *wenn ihre Primfaktorzerlegung gegeben ist*. Das folgende Lemma (mitsamt Beweis) ist eine leichte Verallgemeinerung von Lemma 5.2.7.

Lemma B.0.1
Es sei N eine Carmichaelzahl und p ein Primfaktor von N. Wir schreiben $N = p^\ell \cdot M$ für eine zu p teilerfremde Zahl $M \geq 1$. Dann gilt $\ell = 1$. (Man sagt: N ist *quadratfrei*; kein Primfaktor kommt in N mit einer Vielfachheit größer als 1 vor.)

Beweis: Indirekt. Angenommen, $\ell \geq 2$. Wir geben eine Zahl $a \in \mathbb{Z}_N^*$ an, die $a^{N-1} \bmod N \neq 1$ erfüllt. Dazu benutzen wir den Chinesischen Restsatz. Weil p^ℓ und M teilerfremd sind, gibt es nach dem CRS eine Zahl $a \in \mathbb{Z}_N$ mit $a \bmod p^\ell = p^{\ell-1} + 1$ und $a \bmod M = 1$. Weil a teilerfremd zu p^ℓ und teilerfremd zu M ist, gilt $a \in \mathbb{Z}_N^*$, nach Prop. 5.1.23.

Wir zeigen gleich, dass $a^{N-1} \bmod p^\ell \neq 1$ gilt. Nach dem Chinesischen Restsatz folgt daraus, dass $a^{N-1} \bmod N \neq 1$ gilt, wie gewünscht. Wir rechnen modulo p^ℓ, mit der binomischen Formel:

$$\begin{aligned} a^{N-1} &\equiv (p^{\ell-1} + 1)^{N-1} \\ &\equiv \sum_{0 \leq j \leq N-1} \binom{N-1}{j} (p^{\ell-1})^j \\ &\equiv 1 + (N-1) \cdot p^{\ell-1} \pmod{p^\ell}. \end{aligned} \tag{B.0.9}$$

(Die letzte Äquivalenz ergibt sich daraus, dass für $j \geq 2$ gilt, dass $(\ell-1)j \geq \ell$ ist, also der Faktor $(p^{\ell-1})^j = p^{(\ell-1)j}$ durch p^ℓ teilbar ist, der entsprechende Summand also modulo p^ℓ wegfällt.) Weil $N - 1 = p^\ell \cdot M - 1$ nicht durch p teilbar ist, ist $(N-1) \cdot p^{\ell-1}$ nicht durch p^ℓ teilbar. Damit folgt aus (B.0.9), dass $a^{N-1} \not\equiv 1 \pmod{p^\ell}$ ist, also $a^{N-1} \bmod p^\ell \neq 1$. □

Das Lemma bedeutet also, dass in der Primfaktorzerlegung einer Carmichaelzahl kein Primfaktor mehrfach vorkommt.

Lemma B.0.2
Es sei N eine Carmichaelzahl und p ein Primfaktor von N. Dann gilt $p - 1 \mid N - 1$.

Beweis: Hier benötigen wir etwas Gruppentheorie und Zahlentheorie. Wir schreiben $N = p \cdot M$ für $M = N/p > 1$. Nach Lemma B.0.1 sind dann M und p teilerfremd. Nach dem Chinesischen Restsatz sind $\mathbb{Z}_N^*$ und $\mathbb{Z}_p^* \times \mathbb{Z}_M^*$ isomorph. Weil p eine Primzahl ist, ist (nach einem einfachen Satz der Zahlentheorie) $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ eine *zyklische* Gruppe. Das bedeutet, dass es in $\mathbb{Z}_p^*$ ein Element g (ein „erzeugendes Element“) mit

$\mathbb{Z}_p^* = \{1, g, g^2, \ldots, g^{p-2}\}$ gibt (Rechnung in $\mathbb{Z}_p^*$, also modulo p). Für dieses Element gilt:

$$g^j = 1 \;\Leftrightarrow\; p-1 \mid j, \text{ für alle ganzen Zahlen } j. \qquad (*)$$

Der Chinesische Restsatz liefert uns ein Element $a \in \mathbb{Z}_N^*$ mit $a \bmod p = g$ und $a \bmod M = 1$. Weil N Carmichaelzahl ist, gilt $a^{N-1} \bmod N = 1$, also nach Wahl von a und dem Chinesischen Restsatz $g^{N-1} \bmod p = a^{N-1} \bmod p = 1$. Nach $(*)$ folgt $p-1 \mid N-1$. □

Man sieht nun ganz leicht, dass Produkte von zwei Primzahlen nie Carmichaelzahlen sein können.

Lemma B.0.3
(a) Alle Primfaktoren einer Carmichaelzahl N sind kleiner als $\sqrt{N}$.
(b) Jede Carmichaelzahl hat mindestens drei Primfaktoren.

Beweis: (b) folgt direkt aus (a). Der Beweis von (a) ist indirekt. Annahme: N ist Carmichaelzahl und N hat einen Primfaktor $p > \sqrt{N}$. (Der Fall $p = \sqrt{N}$ ist nach Lemma B.0.1 ausgeschlossen.) Dann ist $p > N/p$. Offensichtlich gilt $p-1 \mid (p-1)(N/p)$, also $p-1 \mid N - N/p$. Nach Lemma B.0.2 gilt $p-1 \mid N-1$. Es folgt $p-1 \mid N/p - 1$, also $p-1 \leq N/p - 1$, ein Widerspruch. □

Die Situation bei der Carmichaelzahl $561 = 3 \cdot 11 \cdot 17$ ist also typisch: Es gibt mindestens drei verschiedene Primfaktoren, hier 3, 11 und 17, und es gilt, dass $2 = 3-1$, $10 = 11-1$ und $16 = 17-1$ Teiler von $560 = 561-1$ sind. (Man prüfe die entsprechenden Gleichungen auch für $1105 = 5 \cdot 13 \cdot 17$ und $1729 = 7 \cdot 13 \cdot 19$ nach!) Es stellt sich heraus, dass diese Bedingung Carmichaelzahlen vollständig charakterisiert.

Proposition B.0.4
N ist eine Carmichaelzahl genau dann wenn N mindestens drei Primfaktoren hat, die alle verschieden sind, und wenn für jeden Primfaktor p von N gilt: $p-1 \mid N-1$.

Beweis: Dass jede Carmichaelzahl die genannten Eigenschaften hat, haben wir schon gesehen. Sei nun $N = p_1 \ldots p_r$ eine Zahl mit verschiedenen Primfaktoren $p_1, \ldots, p_r$, so dass $p_i - 1 \mid N-1$ für $1 \leq i \leq r$. Es sei $a \in \mathbb{Z}_N^*$ beliebig. Dann gilt für jedes i:

$$a^{N-1} \bmod p_i = (a^{p_i-1} \bmod p_i)^{(N-1)/(p_i-1)} \bmod p_i.$$

Nach dem kleinen Satz von Fermat gilt $a^{p_i-1} \bmod p_i = 1$, und damit $a^{N-1} \bmod p_i = 1$. Die verallgemeinerte Version des Chinesischen Restsatzes für r teilerfremde Faktoren besagt, dass $\mathbb{Z}_N$ und $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_r}$ isomorph sind, über die Funktion $\Phi(b) = (b \bmod p_1, \ldots, b \bmod p_r)$, die 1 auf $(1, \ldots, 1)$ abbildet. Es folgt $a^{N-1} \bmod N = 1$. □

Proposition B.0.4 führt dazu, dass für Zahlen N *mit gegebener Primfaktorzerlegung* effizient entschieden werden kann, ob N Carmichaelzahl ist. Wir müssen nur für jeden Primfaktor p von N prüfen, ob p in der Primfaktorzerlegung nur einfach vorkommt und ob $p-1$ die Zahl $N-1$ teilt. Aber Achtung: Wenn die Primfaktoren von N nicht bekannt sind, funktioniert dieser Test nicht.

In diesem Fall können wir wie folgt vorgehen, für eine gegebene ungerade Zahl $N \geq 3$ und eine Wiederholungszahl ℓ:

- Für zufällig gewählte Zahlen $a_1, a_2, \ldots$ aus $\{1, \ldots, N-1\}$ tue Folgendes:
  - Teste, ob $\mathrm{ggT}(a_j, N) = 1$ gilt. Falls nein: Notiere, dass N zusammengesetzt ist. Falls ja:
  - Lasse den Miller-Rabin-Test auf N mit a_j ablaufen, mit folgender Modifikation: Alle Zahlen $b_i = a_j^{u \cdot 2^i} \bmod N$, $i = 0, 1, \ldots, k$, werden berechnet, und es wird am Ende auch geprüft, ob a_j F-Zeuge ist.
  - Falls a_j F-Zeuge ist, wird Ausgabe „2“ ausgegeben und angehalten.
  - Falls a_j MR-Zeuge ist, wird notiert, dass N zusammengesetzt ist.
- Wenn nie mit Ausgabe „2“ abgebrochen wird, wird die Schleife so lange wiederholt, bis ℓ viele a_j mit $\mathrm{ggT}(a_j, N) = 1$ untersucht worden sind.
- Wenn nie festgestellt wurde, dass N zusammengesetzt ist, ist die Ausgabe „0“.
- Wenn festgestellt wurde, dass N zusammengesetzt ist, aber in jeder Berechnung mit einem a_j, das teilerfremd zu N ist, das Ergebnis $b_k = a_j^{N-1} \bmod N = 1$ ist (also a_j kein F-Zeuge ist), ist die Ausgabe „1“.

Dieser Algorithmus verhält sich wie folgt:

(a) Wenn N eine Primzahl ist, ist die Ausgabe 0.

(b) Wenn N eine Carmichaelzahl ist, dann ist die Ausgabe 0 oder 1, und es gilt $\mathbf{Pr}(\text{Ausgabe ist } \neq 1) \leq 4^{-\ell}$.

(c) Wenn N eine zusammengesetzte Zahl, aber keine Carmichael-Zahl ist, sind Ausgaben 0, 1 und 2 möglich, und es gilt $\mathbf{Pr}(\text{Ausgabe ist } \neq 2) \leq 2^{-\ell}$.

Zu (a): Ausgabe 1 wird nur erzeugt, wenn ein MR-Zeuge gefunden wurde. Ausgabe 2 wird nur erzeugt, wenn ein F-Zeuge gefunden wurde. In beiden Fällen ist N keine Primzahl.
(b) Ausgabe 2 kann nur für Zahlen vorkommen, die zusammengesetzt sind, aber keine Carmichaelzahl. Die Ausgabe 0 entsteht nur dann, wenn bei den ℓ Versuchen mit a_j, die teilerfremd zu N sind, niemals ein MR-Zeuge gefunden wird. Wir hatten in Abschnitt 5.2.5 gesagt, dass die Wahrscheinlichkeit hierfür in einem Versuch höchstens $\frac{1}{4}$ ist, für alle ℓ Versuche zusammen also $4^{-\ell}$.
(c) Ausgabe 0 oder 1 kann nur entstehen, wenn jedes der getesteten a_j mit $\mathrm{ggT}(a_j, N) = 1$ ein F-Lügner ist. Nach Satz 5.2.4 ist dies für jedes neu gewählte a_j mit Wahrscheinlichkeit höchstens $\frac{1}{2}$ der Fall. Die Wahrscheinlichkeit, dass in ℓ Versuchen jedesmal ein F-Lügner gewählt wird, ist höchstens $2^{-\ell}$.

Dieser Test zur Klassifizierung von Zahlen als Primzahlen, Carmichaelzahlen und sonstige zusammengesetzte Zahlen ist also ein Monte-Carlo-Algorithmus mit allgemeiner Ausgabe, wie in Definition 3.2.4 betrachtet.

Ein effizienter deterministischer Test für die Eigenschaft „N ist Carmichaelzahl“ ist nicht bekannt.

C Quadratwurzeln modulo Primzahl p, alternativer Algorithmus

Wir betrachten hier nur den Fall $p \equiv 1 \pmod 4$. Nach Prop. 5.4.7 ist das Problem des Quadratwurzelziehens für die anderen Primzahlen p leicht deterministisch zu lösen.[19]

Algorithmus C.0.1 ***Quadratwurzeln, randomisiert***
Input: Primzahl p mit $p-1$ durch 4 teilbar;
$a \in \{1, \ldots, p-1\}$ mit $a^{(p-1)/2} \bmod p = 1$.
Methode:
// Finde $b \in \mathbb{Z}_p^* - \mathrm{QR}_p$:
1 **repeat**
2 $b \leftarrow$ Zufallszahl in $\{2, \ldots, p-2\}$
3 **until** $b^{(p-1)/2} \bmod p = p-1$;
4 Schreibe $\frac{1}{2}(p-1) = u \cdot 2^k$, u ungerade, $k \geq 1$;
// Berechne $s_0, \ldots, s_k$ gerade mit $a^{u \cdot 2^{k-i}} \cdot b^{s_i} \bmod p = 1$:
5 $\mathtt{s} \leftarrow 0$;
6 **for** $\mathtt{i}$ **from** 1 **to** k **do**
7 **if** $a^{u \cdot 2^{k-\mathtt{i}}} \cdot b^{\mathtt{s}/2} \bmod p = 1$
8 **then** $\mathtt{s} \leftarrow \mathtt{s}/2$
9 **else** $\mathtt{s} \leftarrow \mathtt{s}/2 + (p-1)/2$;
10 **return** $a^{(u+1)/2} \cdot b^{\mathtt{s}/2} \bmod p$.

Bemerkung: Wenn $p \equiv 5 \pmod 8$, dann ist 2 ein nichtquadratischer Rest modulo p (Zahlentheorie, quadratisches Reziprozitätsgesetz). Man kann dann Zeilen 1–3 durch $b \leftarrow 2$ ersetzen und erhält einen deterministischen Algorithmus. Für $p \equiv 1 \pmod 8$ ist keine einfache, direkte Methode zum Finden eines nichtquadratischen Restes bekannt.

Erklärung, Korrektheitsbeweis, Zeitanalyse: Wir setzen voraus, dass ein vorab durchgeführter Quadratzahltest (Algorithmus 5.4.5) ergeben hat, dass

$$a^{(p-1)/2} \bmod p = 1 \tag{C.0.10}$$

[19] Der hier angegebene Algorithmus wurde 1891 von Alberto Tonelli (1849–1920), einem italienischen Mathematiker, gefunden. Daniel Shanks hat ihn 1972 erneut gefunden. Er wird daher oft als Algorithmus von Tonelli und Shanks bezeichnet.

ist und daher a in $\mathbb{Z}_p$ eine Quadratwurzel besitzt. In Zeilen 1–3 wird mit randomisierter Suche eine Zahl $b \in \mathbb{Z}_p^*$ gesucht, die kein quadratischer Rest ist. Weil $(p-1)/2$ gerade ist, ist $(-1)^{(p-1)/2} \bmod p = 1$, also -1 ein quadratischer Rest; wir müssen also 1 und $p-1$ gar nicht testen. Unter den $p-3$ Zahlen in $\{2, \ldots, p-2\}$ sind genau $\frac{1}{2}(p-1)$ viele Nichtquadrate; das ist mehr als die Hälfte, also wird nach einer erwarteten Rundenzahl von < 2 ein passendes b gefunden. (Da der Rest des Algorithmus deterministisch ist, handelt es sich insgesamt um einen lupenreinen Las-Vegas-Algorithmus.)

Bemerkung: In Kapitel 4.1 haben wir ein randomisiertes Suchverfahren betrachtet, das in einer Menge A mit einer nichtleeren Teilmenge B, die nur indirekt, durch einen Test „ist $x \in B$?“ gegeben ist, ein Element von B sucht.

Man beachte, dass das in Zeilen 1–3 gelöste Problem, ein Element der Menge $\mathbb{Z}_p^* - \mathrm{QR}_p$ zu finden, genau dieser Aufgabenstellung entspricht:

- A ist $\mathbb{Z}_p^*$;
- B ist $\mathbb{Z}_p^* - \mathrm{QR}_p$;
- der Test „ist $b \in \mathbb{Z}_p^* - \mathrm{QR}_p$?“ wird in Zeile 3 durchgeführt (Eulersches Kriterium).

Die Menge B hat Dichte etwas größer als $\frac{1}{2}$ in $\mathbb{Z}_p^*$. Man beachte, dass tatsächlich, wie in Kapitel 4 abstrakt angenommen, keine explizite Darstellung von $\mathbb{Z}_p^* - \mathrm{QR}_p$ bekannt ist.

Zu Zeile 4: Weil $p-1$ durch 4 teilbar ist, ist $\frac{1}{2}(p-1)$ gerade, hat also eine Darstellung wie angegeben. – Startend in Zeile 5 berechnen wir in der Variablen `s` eine Folge $s_0, s_1, \ldots, s_k$ mit folgender Eigenschaft:

$$a^{u \cdot 2^{k-i}} \cdot b^{s_i} \bmod p = 1 \quad \text{und} \quad s_i \ \text{ ist gerade} \ . \tag{C.0.11}$$

Die Eigenschaft (C.0.11) beweisen wir durch Induktion über die Schleifendurchläufe $i = 0, 1, \ldots, k$.
$i = 0$: $s_0 = 0$ ist gerade und $a^{u \cdot 2^k} \cdot b^0 \bmod p = a^{(p-1)/2} \bmod p = 1$, wegen (C.0.10).
$0 \le i < k$, „$i \to i+1$“: Nach Induktionsvoraussetzung gilt (C.0.11) für i. Wir betrachten Schleifendurchlauf $i+1$. In Zeile 7 wird berechnet:

$$f := a^{u \cdot 2^{k-(i+1)}} \cdot b^{s_i/2} \bmod p.$$

Die Zahl f ist eine Quadratwurzel von 1, denn $f^2 \bmod p = a^{u \cdot 2^{k-i}} \cdot b^{s_i} \bmod p = 1$ (nach (C.0.11) für i). Daher gilt $f \in \{1, -1\}$. Diese beiden Fälle werden in Zeilen 7–9 betrachtet.

1. Fall: $f = 1$. – Dann wird $s_{i+1} = s_i/2$, und $a^{u \cdot 2^{k-(i+1)}} \cdot b^{s_{i+1}} \bmod p = f = 1$.

2. Fall: $f = -1$. – Dann wird $s_{i+1} = s_i/2 + (p-1)/2$, und

$$a^{u \cdot 2^{k-(i+1)}} \cdot b^{s_{i+1}} \bmod p = f \cdot b^{(p-1)/2} \bmod p = (-1) \cdot (-1) \bmod p = 1.$$

(Wir benutzen hier, dass b kein quadratischer Rest ist, also $b^{(p-1)/2} \bmod p = -1$ ist.)

Es fehlt nur noch der Nachweis, dass s_{i+1} gerade ist. Annahme: s_{i+1} ungerade. Dann gilt:

$$a^{u \cdot 2^{k-(i+1)}} \cdot b^{s_{i+1}} \bmod p = 1.$$

Dabei ist die rechte Seite (die Zahl 1) ein quadratischer Rest, ebenso wie $a^{u \cdot 2^{k-(i+1)}} \bmod p$. Der Faktor $b^{s_{i+1}} \bmod p$ hingegen ist als ungerade Potenz von $b \notin \mathrm{QR}_p$ nicht in QR_p. Dies ist ein Widerspruch zu Korollar 5.4.6.

In Zeile 10 des Algorithmus wird die Zahl $t = a^{(u+1)/2} \cdot b^{s_k/2} \bmod p$ als Ergebnis ausgegeben. Wir berechnen:

$$t^2 \bmod p = a^{u+1} \cdot b^{s_k} \bmod p = a \cdot (a^u \cdot b^{s_k}) \bmod p = a,$$

wobei wir (C.0.11) (für $i = k$) angewendet haben. Daher ist t eine Quadratwurzel von a.

Laufzeit: Es werden maximal $\log p$ Runden durchgeführt; in jeder Runde ist eine schnelle Exponentiation mit $O(\log p)$ Multiplikationen modulo p durchzuführen. Die Anzahl der Multiplikationen und Divisionen ist also $O((\log p)^2)$, die Anzahl der Bitoperationen $O((\log p)^4)$.