Probabilistische Primzahltests

Nicolas Rachinsky

29.1.2003

Zusammenfassung

Obwohl inzwischen bekannt ist, daß deterministische, in polynomialer Zeit ablaufende Algorithmen existieren, um festzustellen, ob eine Zahl prim ist, sind probabilistische Verfahren immer noch um Größenordnungen schneller, und außerdem leichter zu implementieren.

1 Einleitung

Oft benötigt man große „zufällige“ Primzahlen (z.B. in der Kryptographie – RSA). Diese kann man erhalten, indem man zufällige (ungerade) Zahlen der gewünschten Größenordnung auf Primalität testet, bis man auf eine Primzahl stößt. Wenn man einen schnellen Algorithmus für diesen Test hat, dann ist dieser Ansatz erfolgreich, da für die Verteilungsfunktion der Primzahlen $\pi(n)$, die angibt, wieviele Primzahlen kleiner oder gleich n existieren, folgender Zusammenhang gilt.

Satz 1

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

Diese Abschätzung ist sogar schon für kleine n sehr gut. Deshalb ist die Wahrscheinlichkeit mit einem zufällig gewählten n eine Primzahl zu erhalten in etwa $1/\ln n$[1]. Da es reicht, ungerade Zahlen zu betrachten, muß man um etwa eine Zahl mit 150 Dezimalstellen zu erhalten (das entspricht etwa 500 Binärstellen) nur in etwa 175 Zahlen probieren.

Im folgenden wird es um Algorithmen gehen, die feststellen, ob ein solcher Kandidat eine Primzahl ist, oder nicht.

2 Zeitkomplexität von zahlentheoretischen Algorithmen

Bei der Laufzeitbetrachtung von Algorithmen für zahlentheoretische Fragen betrachtet man die Laufzeit üblicherweise nicht in Abhängigkeit von

[1] Im folgenden verwende ich ln für den natürlichen Logarithmus und lg für den Logarithmus zur Basis zwei.

der Eingabe(-zahl) n, sondern ihrer Bitbreite $\lg n$. Dadurch ist es weiterhin sinnvoll anzunehmen, daß die grundlegenden arithmetischen Operationen eine feste Zeit brauchen (solange sie in einem polynomialem Zusammenhang mit der Eingabegröße stehen), was mit der sonst üblichen Betrachtung für große n falsch ist, da bei größeren Bitbreiten die Prozessoren die Berechnungen nicht „auf einmal“ durchführen können. Der ggT kann mit diesem Kostenmaß in polynomialer Zeit berechnet werden [1].

Laufzeit des naiven Primzahltestalgorithmus

Der naive Algorithmus zum Testen, ob eine Zahl prim ist, besteht darin, alle potentiellen Teiler (2 und die ungeraden Zahlen von 3 bis $\sqrt{n}$) durchzuprobieren. Die Laufzeit dieses Algorithmus ist offensichtlich $O(\sqrt{n})$. Mit obigen Kostenmaß ist das eine exponentielle Laufzeit, da $\sqrt{n} = 2^{\lg(n)/2}$.

3 Komplexität von PRIMALITÄT

Im August 2002 wurde bewiesen, daß PRIMALITÄT $\in \mathbf{P}$. Der Beweis findet sich in [3].

4 Probabilistische Primzahltests

Um einen probabilistischen Algorithmus für PRIMALITÄT oder ZUSAMMENGESETZTHEIT[2] zu finden, ist es nützlich eine Menge potentieller „Zeugen“ für die gewünschte Eigenschaft zu haben, in der genügend solcher „Zeugen“ sind. Da die bekannten „Zeugen“ für PRIMALITÄT sehr komplex sind, ist es schwer, hier einen Algorithmus zu finden[3]. Für ZHEIT könnte man annehmen, daß $\mathbb{Z}_n$[4] für zusammengesetzte n viele Elemente hat, die nicht relativ prim zu n sind. Ein solches Element würde zeigen, daß n nicht prim ist. Man sieht jedoch schnell, daß dieser Test nicht gut ist. Zum Beispiel ist er für $n = qp$, mit q und p zwei annähernd gleichgroßen Primzahlen unbrauchbar. Es gibt in diesem Fall nur $(q-1)+(p-1)$ „Zeugen“, da nur Vielfache von q und p nicht relativ prim zu n sind. Somit ist der Anteil der „Zeugen“ in $\mathbb{Z}_n$ nur $\frac{(q-1)+(p-1)}{qp} \approx \frac{2q}{q^2} = \frac{2}{q}$.

Besser verwendbar scheint hier *Fermats kleiner Satz*.

Satz 2 (Fermats kleiner Satz)

$$n \text{ prim} \Rightarrow \forall a \in \mathbb{Z}_n^*.a^{n-1} \equiv 1 \pmod{n}$$

[2] im Folgenden ZHEIT

[3] Es gibt solche Algorithmen – allerdings sind diese äußerst komplex und haben deshalb für die praktische Anwendung keine Bedeutung[1].

[4] $\mathbb{Z}_n = \{0, 1, \ldots, n-1\} \quad \mathbb{Z}_n^* = \{k | k \in \{1, \ldots, n-1\} \ \wedge \ k, n \text{ relativ prim}\}$

Falls die Umkehrung gälte, ließe sich hierüber ein solcher Test konstruieren. Es gibt jedoch *Pseudoprimzahlen* (auch *Carmichael Zahlen*), für die dieser Satz ebenfalls gilt.

Definition 1 (Carmichael Zahl) *Eine zusammengesetzte Zahl n für die gilt*

$$\forall a \in \mathbb{Z}_n^*.a^{n-1} \equiv 1 \pmod{n}$$

heißt Carmichael Zahl.

N.B. Die kleinste *Carmichael Zahl* ist 561 ($= 3 \cdot 11 \cdot 17$) [1].

4.1 Ein Primzahltest

Der folgende Algorithmus ist ein **BPP** Algorithmus, der die beiden obigen Ansätze enthält.

Algorithmus 1

Eingabe: *Ungerade Zahl n und t.*
Ausgabe: *PRIM oder ZUSAMMENGESETZT (ZG)*

1. if n ist der Form[5] k^l mit $k, l \in \mathbb{N}, l > 1$ then return ZG
2. wähle $b_1, b_2, \ldots, b_t$ unabhängig aus $\mathbb{Z}_n \setminus \{0\}$
3. if für eines der b_i der $\text{ggT}(b_i, n) \neq 1$ then return ZG
4. berechne $r_i := b_i^{\frac{n-1}{2}} \mod n$ für alle $1 \leq i \leq t$
5. if eines der $r_i \not\equiv \pm 1 \pmod{n}$ then return ZG
6. if alle $r_i \equiv 1 \pmod{n}$ then return ZG else return PRIM

Satz 3 *Die Wahrscheinlichkeit. daß sich Algorithmus 1 irrt, ist höchstens $O(1/2^t)$.*

Beweis: Falls n prim ist, kann er nur Schritt 6 eine falsche Ausgabe liefern. Er irrt sich genau dann, wenn alle b_i Quadrate in $\mathbb{Z}_n$ sind[6]. Die Wahrscheinlichkeit für ein zufälliges Element aus $\mathbb{Z}_n$, das nicht 0 ist, ein Quadrat zu sein, ist genau $1/2$, da n eine Primzahl ist.

Falls n eine zusammengesetzte Zahl ist, dann kann er ebenfalls nur in Schritt 6 ein falsches Ergebnis liefern, und nur falls n keine Potenz einer

[5] k und l lassen sich in polynomialer Zeit bestimmen.

[6] Zur Erinnerung: In $\mathbb{Z}/(p), p$ prim, gilt $x^{\frac{p-1}{2}} \mod p = \pm 1$. Es ist genau dann $+1$, wenn x ein Quadrat in $\mathbb{Z}/(p)$ ist. In $\mathbb{Z}/(p)$ gibt es genausoviele Quadrate wie Nicht-Quadrate.

Primzahl ist. In diesem Fall gilt aber der folgende Satz, nach dem die Wahrscheinlichkeit für ein zufälliges gewähltes Element von $\mathbb{Z}_n \setminus \{0\}$, mit $\frac{n-1}{2}$ potenziert ± 1 zu ergeben, höchstens 1/2 ist, da ein $r_i \equiv -1 \pmod{n}$ (sei o.B.d.A. $r_1 = -1$). Also ist die Wahrscheinlichkeit, daß dies auf alle übrigen (d.h. $i > 1$) b_i zutrifft höchstens $\frac{1}{2^{t-1}}$. □

Satz 4 *Falls n eine ungerade, zusammengesetzte Zahl, die keine Potenz einer Primzahl ist, ist und ein $a \in \mathbb{Z}_n^*$ existiert, für das gilt*

$$a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$$

dann gilt für $S_n := \{x \in \mathbb{Z}_n^* | x^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}\}$

$$|S_n| \leq \frac{1}{2}|\mathbb{Z}_n^*|.$$

Beweis: Sei $p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t} = n$ die Primfaktorzerlegung von n, wobei $t \geq 2$ ist. Sei $q := p_1^{k_1}, m := n/q$, dann ist $\mathrm{ggT}(m, q) = 1$ und m ist kein trivialer Faktor von n. Mit dem *Chinesischen Restsatz* wählen wir nun ein $b \in \mathbb{Z}_n^*$, das folgende Gleichungen erfüllt:

$$\begin{aligned} b &\equiv a \pmod{q} \\ b &\equiv 1 \pmod{m} \end{aligned}$$

Dann gilt auch:

$$\begin{aligned} b^{\frac{n-1}{2}} \equiv a^{\frac{n-1}{2}} &\equiv -1 \pmod{q} \\ b^{\frac{n-1}{2}} &\equiv 1 \pmod{m} \end{aligned}$$

Falls $b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ wäre, dann wäre auch $b^{\frac{n-1}{2}} \equiv 1 \pmod{m}$ und somit auch $b^{\frac{n-1}{2}} \equiv 1 \pmod{q}$. Ebenso für $b^{\frac{n-1}{2}} \equiv -1 \pmod{n}$. Also ist $b^{\frac{n-1}{2}} \not\equiv \pm 1 \pmod{n}$, d.h. $b \notin S_n$. Da man leicht sieht, daß S_n eine Untergruppe von $\mathbb{Z}_n^*$ ist, gilt $|S_n| \big| |\mathbb{Z}_n^*|$ (Lagrange). Da es eine echte Untergruppe ist, gilt $|\mathbb{Z}_n^*|/|S_n| > 1$ (und da die linke Seite eine ganze Zahl ist $|\mathbb{Z}_n^*|/|S_n| \geq 2$) und damit die Behauptung. □

4.2 Ein RP Primzahltest (von Solovay und Strassen)

Zunächst benötigen wir eine erweiterte Fassung des Legendre-Symbols. Zur Erinnerung, für alle Primzahlen $n > 2$ und alle $a \in \mathbb{Z}_n^*$, ist das Legendre-Symbol $\left[\frac{a}{n}\right] \equiv a^{\frac{n-1}{2}} \pmod{n}$. Das Jacobi-Symbol ist für alle ungeraden n definiert, und für alle Primzahlen ist es dasselbe wie das Legendre-Symbol, weshalb wir dasselbe Symbol verwenden.

Definition 2 (Jacobi-Symbol) *Sei für n ungerade mit der Zerlegung in Primfaktoren $p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}$. Dann ist für alle a mit $\mathrm{ggT}(a, n) = 1$ das Jacobi-Symbol wie folgt definiert*

$$\left[\frac{a}{n}\right] := \prod_{i=1}^{t} \left[\frac{a}{p_i}\right]^{k_i}.$$

Wie das Legendre-Symbol ist der Wert des Jacobi-Symbols entweder 1 oder -1. Obwohl es auf den ersten Blick so scheint, als ob zur Berechnung die Primfaktorzerlegung von n nötig wäre, gibt es einen Algorithmus, der seinen Wert in polynomialer Zeit ohne diese Zerlegung ausrechnet, wobei die folgenden Eigenschaften verwendet werden.

Satz 5 (Eigenschaften des Jacobi-Symbols) *Das Jacobi-Symbol erfüllt die folgenden Bedingungen.*

1. $\left[\frac{ab}{n}\right] = \left[\frac{a}{n}\right]\left[\frac{b}{n}\right]$
2. $a \equiv b \pmod{n} \Rightarrow \left[\frac{a}{n}\right] = \left[\frac{b}{n}\right]$
3. a ungerade $\wedge\, a, n$ rel. prim $\Rightarrow \left[\frac{a}{n}\right] = (-1)^{\frac{a-1}{2}\frac{n-1}{2}}\left[\frac{n}{a}\right]$
4. $\left[\frac{1}{n}\right] = 1$
5. $\left[\frac{2}{n}\right] = \begin{cases} -1 & \text{für } n \equiv 3 \text{ oder } 5 \pmod{8} \\ 1 & \text{für } n \equiv 1 \text{ oder } 7 \pmod{8} \end{cases}$

Damit läßt sich folgender Algorithmus (aus **RP**) für die ZHEIT konstruieren. Er gibt PRIM oder ZUSAMMENGESETZT zurück, je nachdem, wie er „geraten“ hat. Er gibt ZUSAMMENGESETZT nur dann zurück, wenn die Zahl tatsächlich zusammengesetzt ist. Er kann jedoch PRIM zurückgeben, obwohl die Zahl es nicht ist, d.h. PRIM bedeutet „vermutlich prim“ während ZUSAMMENGESETZT „auf jeden Fall zusammengesetzt“ heißt.

Der Algorithmus basiert darauf, daß für n prim gilt $\left[\frac{a}{n}\right] \equiv a^{\frac{n-1}{2}} \pmod{n}$, während es für ein zusammengesetztes n ein große Anzahl $a \in \mathbb{Z}_n^*$ gibt, für die $\left[\frac{a}{n}\right] \not\equiv a^{\frac{n-1}{2}} \pmod{n}$.

Algorithmus 2

Eingabe: *Ungerade Zahl n.*
Ausgabe: *PRIM oder ZUSAMMENGESETZT*

1. wähle a zufällig aus $\mathbb{Z}_n \setminus \{0\}$
2. if $\mathrm{ggT}(a, n) \neq 1$ then return ZUSAMMENGESETZT
3. if $\left[\frac{a}{n}\right] \equiv a^{\frac{n-1}{2}} \pmod{n}$ then return PRIM
   else return ZUSAMMENGESETZT

Dieser Algorithmus hat immer recht, wenn er ZUSAMMENGESETZT liefert, da er dann ein $a \in \mathbb{Z}_n^*$ gefunden hat, für das gilt $\mathrm{ggT}(a, n) \neq 1$ oder $\left[\frac{a}{n}\right] \not\equiv a^{\frac{n-1}{2}} \pmod{n}$, was nur für ein zusammengesetztes n der Fall ist. Wir zeigen jetzt, daß die Wahrscheinlichkeit, daß der Algorithmus PRIM zurückliefert, obwohl es nicht stimmt, $\leq \frac{1}{2}$ ist.

Definition 3 *Für jede ungerade Zahl n sei J_n wie folgt definiert.*

$$J_n := \left\{ a \in \mathbb{Z}_n^* | \left[\frac{a}{n}\right] \equiv a^{\frac{n-1}{2}} \pmod{n} \right\}$$

Für n prim gilt $J_n = \mathbb{Z}_n^*$. Im Folgenden wird gezeigt, daß es für ein zusammengesetztes n viel kleiner ist.

Satz 6 *Für zusammengesetztes, ungerades n gilt: $|J_n| \leq \frac{1}{2}|\mathbb{Z}_n^*|$.*

Beweis: J_n ist eine Untergruppe von $\mathbb{Z}_n^*$ (bzgl. der Multiplikation in $\mathbb{Z}/(n)$), was man mit den Eigenschaften in Satz 5 schnell sieht. Es bleibt zu zeigen, daß es eine echte Untergruppe ist. Nehmen wir an, es gäbe ein zusammengesetztes n mit $J_n = \mathbb{Z}_n^*$. Sei die Primfaktorzerlegung von $n = p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}$. Im folgenden sei $q := p_1^{k_1}$ und $m := p_2^{k_2} \cdots p_t^{k_t}$. Jetzt fixieren wir ein erzeugendes Element[7] g von $\mathbb{Z}_q^*$, und wählen $a \in \mathbb{Z}_n^*$ so, daß folgende Gleichungen erfüllt sind:

$$\begin{aligned} a &\equiv g \pmod{q} \\ a &\equiv 1 \pmod{m} \end{aligned}$$

Nach dem *Chinesischen Restsatz* existiert ein solches a. Außerdem gilt $\forall i \geq 2. a \equiv 1 \pmod{p_i}$, da $p_i | m$ und $m | (a-1)$.

Jetzt betrachten wir zwei Fälle, abhängig von der Primfaktorzerlegung von n. Zuerst betrachten wir den Fall, daß $k_1 = 1$ ist. $\left[\frac{a}{n}\right]$ läßt sich jetzt wie folgt berechnen.

$$\begin{aligned} \left[\tfrac{a}{n}\right] &= \prod_{i=1}^{t} \left[\tfrac{a}{p_i}\right]^{k_i} && \text{nach Definition} \\ &= \left[\tfrac{a}{q}\right] \prod_{i=2}^{t} \left[\tfrac{a}{p_i}\right]^{k_i} && \text{da } q = p_1,\ k_1 = 1 \\ &= \left[\tfrac{g}{q}\right] \prod_{i=2}^{t} \left[\tfrac{1}{p_i}\right]^{k_i} && \text{Eigenschaft 2} \\ &= \left[\tfrac{g}{q}\right] && \text{Eigenschaft 4} \end{aligned}$$

Da das Legendre- und das Jacobi-Symbol für Primzahlen übereinstimmen und ein erzeugendes Element kein Quadrat in $\mathbb{Z}_q^*$ sein kann[8], ist $\left[\frac{a}{n}\right] = \left[\frac{g}{q}\right] = -1$. Da (nach Annahme) $J_n = \mathbb{Z}_n^*$ gilt $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$ und da

[7] Ein solches existiert, da $\mathbb{Z}_q^*$ eine zyklische Gruppe ist, *Theorem 31.32* in [2].

[8] **Beweis:** Für $\mathbb{Z}_3^*$ gilt dies (Durchprobieren). Für $q > 3$ funktioniert folgender Beweis. Sei $g \in \mathbb{Z}_q^*$ mit $< g^2 >= \mathbb{Z}_q^*$. Es gilt $(g^2)^{q-1} = g^{2q-2} = 1$ außerdem $\exists l \leq (q-1) :\ g^l = 1 \Rightarrow g^{2l} = (g^2)^l = 1 \Rightarrow l \geq (q-1)$, also $l = q-1$, also ist l gerade (da q ungerade, da auch n ungerade ist), und es gilt $(g^2)^{\frac{l}{2}} = 1$, was im Widerspruch zu $\mathrm{ord}(g^2) = q-1$, da g^2 erzeugendes Element, steht.

$m|n$ auch $a^{\frac{n-1}{2}} \equiv -1 \pmod{m}$, was unserer Wahl von a ($a \equiv 1 \pmod{m}$) widerspricht.

Im anderen Fall ist $k_1 \geq 2$. Da (nach Annahme) $J_n = \mathbb{Z}_n^*$, gilt auch

$$\begin{aligned} & a^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n} \\ \Rightarrow \quad & a^{n-1} \equiv 1 \pmod{n} \\ \Rightarrow \quad & g^{n-1} \equiv 1 \pmod{q} \quad \text{da } q|n \text{ und } a \equiv g \pmod{q} \end{aligned}$$

Da $\mathbb{Z}_q^*$ von g erzeugt wird, hat g die Ordnung $\Phi(q)$ und es gilt (wegen $g^{n-1} \equiv 1 \pmod{q}$) $\Phi(q)|n-1$. Da $k_1 \geq 2$ gilt auch $p_1|\Phi(q)$ und damit $p_1|(n-1)$. Somit teilt p_1 sowohl $n-1$ als n, was für eine Primzahl nicht sein kann. □

Hieraus folgt sofort

Satz 7 *Algorithmus 2 gibt für eine Primzahl immer PRIM zurück und für eine zusammengesetzte Zahl ZUSAMMENGESETZT mit einer Wahrscheinlichkeit von mindestens* $1/2$.

Dieser Algorithmus ist also ein **RP** Algorithmus für ZHEIT. Wie üblich kann dieser Algorithmus solange wiederholt werden, bis die Wahrscheinlichkeit für einen Fehler klein genug ist. Man hat somit einen *Monte Carlo Algorithmus*.

4.3 Miller-Rabin Primzahltest

Dies ist ebenfalls ein **RP** Algorithmus für ZHEIT. Hier wird der *kleine Satz von Fermat* verwendet. Es gibt allerdings noch zusätzliche Schritte, um *Carmichael Zahlen* ebenfalls zu erkennen.

Algorithmus 3 (Miller-Rabin)

Eingabe: *Ungerade Zahl* n.
Ausgabe: *PRIM oder ZUSAMMENGESETZT (ZG)*

1. wähle a zufällig aus $\{1, 2, \ldots, n-1\}$
2. Berechne u ungerade und $t \geq 1$ so, daß $n - 1 = 2^t u$
3. $x_0 := a^u \mod n$
4. for $i := 1$ to t do
5. $\quad x_i := x_{i-1}^2 \mod n$
6. $\quad$ if $x_i = 1 \wedge x_{i-1} \neq 1 \wedge x_{i-1} \neq n-1$ then return ZG
7. endfor

8. if $x_t \neq 1$ then return ZG

9. return PRIM

Dieser Algorithmus gibt nur ZG aus, wenn n zusammengesetzt ist. Er arbeitet folgendermaßen:
Zuerst wird $n-1$ in eine Potenz von 2 und ein ungerades u zerlegt. Dann wird $a^{n-1} \mod n$ schrittweise berechnet (zuerst wird $a^u \mod n$ ausgerechnet und dann t-mal quadriert, dies klappt wegen der Wahl von u und t). Falls dieses Quadrieren nicht 1 ergibt, dann kann n nicht prim sein, da der Satz von Fermat verletzt ist (das ist der zweite Fall in dem ZG zurückgegeben wird). Nach jedem Quadrieren wird überprüft, ob eine nichttriviale Quadratwurzel von 1 gefunden wurde, falls eine solche gefunden wurde, dann kann n keine Primzahl sein[9] (das ist der erste Fall, in dem ZG zurückgeliefert wird).

Eine andere Betrachtung – die sich im folgenden Beweis als hilfreich erweist – ist, den Algorithmus als Funktion der Folge $X = (x_1, \ldots, x_t)$ (natürlich X in Abhängigkeit von a) zu betrachten, wobei jedoch nicht in allen Fällen alle Werte berechnet werden. In diesen Fällen wären jedoch alle x_i ab der Stelle des Abbrechens 1, weshalb wir sie jetzt als 1 betrachten. Es gibt dann 4 Fälle

1. $X = (\ldots, d)$ mit $d \neq 1$. In diesem Fall wird ZG zurückgegeben, da n gegen den *kleinen Satz von Fermat* verstößt.

2. $X = (1, 1, \ldots, 1)$ In diesem Fall wird PRIM zurückgegeben.

3. $X = (\ldots, -1, 1, \ldots, 1)$ In diesem Fall wird PRIM zurückgegeben. Die Folge endet mit 1 und das letzte Element, das nicht 1 ist, ist -1.

4. $X = (\ldots, d, 1, \ldots, 1)$ mit $d \neq 1$. In diesem Fall wird ZG zurückgegeben, da d eine nichttriviale Quadratwurzel von 1 ist.

Satz 8 *Für ein ungerades, zusammengesetztes n gibt es höchstens $(n-1)/2$ mögliche Werte für a, für die obiger Algorithmus PRIM zurückliefert.*

Beweis: Zuerst zeigen wir, daß ein solches a in $\mathbb{Z}_n^*$ liegen muß. Da es $a^{n-1} \equiv 1 \pmod{n}$ erfüllt, ist a ein invertierbares Element aus $\mathbb{Z}_n$ (mit dem Inversen a^{n-2}), d.h. es liegt in $\mathbb{Z}_n^*$. Nun wird gezeigt, daß eine Menge $B \subseteq \mathbb{Z}_n^*$, die alle a enthält, für die obiger Algorithmus ein falsches Ergebnis liefert, existiert, die eine echte Untergruppe von $\mathbb{Z}_n^*$ ist, womit der Satz gezeigt ist, da (wie vorher) $|B| \leq \frac{1}{2}|\mathbb{Z}_n^*|$ und $|\mathbb{Z}_n^*| \leq |\mathbb{Z}_n \setminus \{0\}| = n-1$.

[9] die trivialen Quadratwurzeln 1 und $-1 = n-1$ gibt es immer, es kann, falls n eine Primzahl ist, keine weiteren geben, da dann $\mathbb{Z}/(n)$ ein Körper ist, in dem jedes Element (außer der Null) keine oder zwei Quadratwurzeln hat.

Fall 1: Es existiert ein $x \in \mathbb{Z}_n^*$ mit $x^{n-1} \not\equiv 1 \pmod{n}$. Dies ist der Fall, wenn n keine *Carmichael Zahl* ist. Sei $B := \{b \in \mathbb{Z}_n^* | b^{n-1} \equiv 1 \pmod{n}\}$. B ist nicht leer, da sie 1 enthält. Man sieht leicht, daß B unter Multiplikation abgeschlossen ist, und für alle Elemente das Inverse enthält[10]. Da außerdem $x \notin B$, ist B eine echte Untergruppe.

Fall 2: $\forall x \in \mathbb{Z}_n^*.x^{n-1} \equiv 1 \pmod{n}$. Dies ist der Fall, wenn n eine *Carmichael Zahl* ist. In diesem Fall kann n keine Potenz einer Primzahl sein. Um dies zu zeigen nehmen wir das Gegenteil an $n = p^e$ mit p prim und $e > 1$. Da n ungerade ist, ist p ebenfalls ungerade. Dann ist $\mathbb{Z}_n^*$ eine zyklische Gruppe[11], sei g ein erzeugendes Element. Dann gilt $|\mathbb{Z}_n^*| = \mathrm{ord}_n(g) = \Phi(n) = p^e - p^{e-1}$. Außerdem haben wir (wegen Fall 2) $g^{n-1} \equiv 1 \pmod{n}$, woraus folgt $n-1 \equiv 0 \pmod{\Phi(n)}$ bzw. $\Phi(n) = (p-1)p^{e-1} | p^e - 1 = n - 1$. Für $e > 1$ ist das ein Widerspruch, da $(p-1)p^{e-1}$ durch die Primzahl p teilbar ist, aber $p^e - 1$ nicht. Also läßt sich n in $n_1 n_2 = n$ zerlegen, wobei n_1, n_2 relativ prim, ungerade und > 1 sind. Zur Erinnerung, der Algorithmus wählt u und t so, daß $n - 1 = 2^t u$ mit $t \geq 1$ und u ungerade. Für ein a wird dann die Folge

$$X = (a^u, a^{2u}, \ldots, a^{2^t u})$$

(alles modulo n) berechnet.

Wir nennen ein Paar (v, j) von ganzen Zahlen **akzeptierend**, wenn $v \in \mathbb{Z}_n^*, j \in \{0, 1, \ldots, t\}$ und $v^{2^j u} \equiv -1 \pmod{n}$. Solche Paare existieren; da u ungerade ist, kann man $v = n - 1, j = 0$ wählen, um ein solches Paar zu erhalten. Jetzt wählen wir das größte j, für das ein solches Paar existiert und fixieren ein v so, daß (v, j) akzeptierend ist. Sei

$$B := \{x \in \mathbb{Z}_x^* | x^{2^j u} \equiv \pm 1 \pmod{n}\}$$

B ist unter der Multiplikation abgeschlossen, es enthält die 1 und (wie vorher) sieht man leicht, daß die Inversen auch enthalten sind. Also ist B eine Untergruppe von $\mathbb{Z}_n^*$. Außerdem enthält B jedes a, für das – fälschlicherweise – PRIM zurückgeliefert wird. Jedes solche a erzeugt ein X, das entweder aus lauter 1 besteht, oder -1 spätestens an der $j$ten Stelle enthält (nach der Wahl von j), in beiden Fällen ist $a \in B$.

Jetzt wird noch gezeigt, daß $B \neq \mathbb{Z}_n^*$. Da $v^{2^j u} \equiv -1 \pmod{n}$ gilt nach dem *Chinesischen Restsatz* auch $v^{2^j u} \equiv -1 \pmod{n_1}$. Ebenso gibt es ein w, das folgende Gleichungen löst.

$$\begin{aligned} w &\equiv v \pmod{n_1} \\ w &\equiv 1 \pmod{n_2} \end{aligned}$$

und damit

$$\begin{aligned} w^{2^j u} &\equiv -1 \pmod{n_1} \\ w^{2^j u} &\equiv 1 \pmod{n_2} \end{aligned}$$

[10] Sei $a \in B \Rightarrow a^{n-1} \equiv 1$. Sei $ab \equiv 1$, dann auch $1 \equiv (ab)^{n-1} \equiv a^{n-1}b^{n-1} \equiv 1b^{n-1}$.

[11] *Theorem 31.32* in [2].

Nach dem *Chinesischen Restsatz* folgt $w^{2^j u} \not\equiv 1 \pmod{n_1} \Rightarrow w^{2^j u} \not\equiv 1 \pmod{n}$ und $w^{2^j u} \not\equiv -1 \pmod{n_2} \Rightarrow w^{2^j u} \not\equiv -1 \pmod{n}$, und damit $w \notin B$. Nun muß noch gezeigt werden, daß w in $\mathbb{Z}_n^*$ liegt. Da $v \in \mathbb{Z}_n^*$ gilt $\mathrm{ggT}(v, n) = 1$ und damit $\mathrm{ggT}(v, n_1) = 1$. Da $w \equiv v \pmod{n_1}$ und damit $\mathrm{ggT}(w, n_1) = 1$. Außerdem gilt $w \equiv 1 \pmod{n_2}$ und damit $\mathrm{ggT}(w, n_2) = 1$. Damit erhält man aber auch $\mathrm{ggT}(w, n_1 n_2) = \mathrm{ggT}(w, n) = 1$, d.h. $w \in \mathbb{Z}_n^*$. Damit haben wir gezeigt, daß B eine echte Untergruppe von $\mathbb{Z}_n^*$ ist. □

Hieraus folgt sofort

Satz 9 *Algorithmus 3 gibt für eine Primzahl immer PRIM zurück und für eine zusammengesetzte Zahl ZUSAMMENGESETZT mit einer Wahrscheinlichkeit von mindestens* $1/2$.

Laufzeitvergleich mit dem AKS Algorithmus aus P

In [4] geht es um die Performance Optimierung von Primtests in **P**, dort ist ein Beispiel für die Laufzeit angegeben, für eine Primzahl mit 30 (Dezimal-)Stellen brauchten sie „about a day“. Im Gegensatz dazu braucht die Perl Implementierung aus dem Anhang von Miller-Rabin für eine 30 dezimalstellige Primzahl etwa vier Sekunden (bei 100 Iterationen, die Wahrscheinlichkeit für ein falsches Ergebnis ist also höchstens 2^{-100}), wobei der für Miller-Rabin verwendete Rechner höchstens doppelt so schnell sein sollte. Derzeit sind also die probabilistischen Algorithmen für den praktischen Einsatz deutlich besser geeignet.

A Implementierung von Miller-Rabin in Perl

A.1 Beispiele

```
> ./prim.pl 100 1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000201 100
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000201 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000203 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000205 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000207 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000209 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000211 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000213 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000215 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000217 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000219 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000221 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000223 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000225 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000227 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000229 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000231 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000233 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000235 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000237 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000239 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000241 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000243 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000245 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000247 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000249 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000251 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000253 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000255 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000257 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000259 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000261 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000263 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000265 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000267 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000269 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000271 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000273 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000275 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000277 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000279 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000281 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000283 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000285 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000287 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000289 is PRIME
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000291 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000293 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000295 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000297 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000299 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000301 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000303 is PRIME
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000305 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000307 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000309 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000311 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000313 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000315 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000317 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000319 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000321 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000323 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000325 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000327 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000329 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000331 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000333 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000335 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000337 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000339 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000341 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000343 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000345 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000347 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000349 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000351 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000353 is COMPOSITE
```

```
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000355 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000357 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000359 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000361 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000363 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000365 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000367 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000369 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000371 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000373 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000375 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000377 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000379 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000381 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000383 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000385 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000387 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000389 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000391 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000393 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000395 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000397 is COMPOSITE
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000399 is COMPOSITE
151.826u 1.029s 3:16.16 77.9%   855+2289k 0+0io 0pf+0w
```

In den folgenden zwei Beispielen steht [...] anstelle von 96 Nullern.

```
> time ./prim.pl 100 1[...]289
1[...]289 is PRIME
50.615u 0.375s 1:04.49 79.0%    853+2282k 0+0io 0pf+0w

> time ./prim.pl 100 1[...]287
1[...]287 is COMPOSITE
0.566u 0.000s 0:00.65 86.1%     878+2210k 0+0io 0pf+0w

$ for i in `jot 100`; do ./prim.pl 1 561;done|grep -c PRIME
6
```

A.2 Source

```
#!/usr/bin/perl

use Math::BigInt ':constant';
use strict;

# Anzahl Durchlaeufe pro Zahl
my $s=new Math::BigInt $ARGV[0];

# erste zu ueberpruefende Zahl
my $n=new Math::BigInt $ARGV[1];

die "odd number required" unless $n%2;


# $t Zahlen ab $n testen
```

```
my $t;

if(@ARGV>2)
{
        $t=new Math::BigInt $ARGV[2];
}
else
{
        $t=1;
}


for(;$t>0;$t--,$n+=2)
{
        print "$n is ";

        if(miller_rabin($n,$s))
        {print "COMPOSITE"}
        else
        {print "PRIME"}

        print "\n";
}


sub miller_rabin($$)
{
        my($n,$s)=@_;
        my($j,$a);

        for($j=1;$j<=$s;$j++)
        {
                $a=random($n-1);
                if(witness($a,$n))
                {
                        return 1;
                }
        }
        return 0;
}


sub witness($$)
{
```

```
        my($a,$n)=@_;
        my($i,$x,$xl);
        my($t,$u)=get_t_u($n-1);

        $x=$a->bmodpow($u,$n);

        for($i=1;$i<=$t;$i++)
        {
                $xl=$x;
                $x=($xl**2)%$n;
                if($x==1 && $xl!=1 && $xl!=$n-1)
                {return 1}
        }

        if($x!=1)
        {return 1}

        return 0;
}

sub get_t_u($)
{
        my($n)=@_;
        my $t;
        $n/=2;
        for($t=1;$n%2==0;$t++)
        {$n/=2}
        return ($t,$n);
}

sub random ($)
{
        my($m)=@_;
        my $ret=0;
        my $p;
        $m-=1;

        for($p=1;$m>0;$p*=1000000000)
        {
                $ret+=(int(rand($m%1000000000+1)))*$p;
                $m/=1000000000;
        }
        return $ret+1;
}
```

Literatur

[1] R. Motwani, P. Raghavan: *Randomized Algorithms*, Cambridge University Press, 19XX, ISBN 0-521-47465-5

[2] T.H. Cormen, C.E. Leisserson, R.L. Rivest, C. Stein: *Introduction to Algorithms*, MIT-Press, 2001, ISBN 0-07-013151-1

[3] Manindra Agrawal, Neeraj Kayal and Nitin Saxena: *PRIMES is in P*
Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur-208016, INDIA
http://www.cse.iitk.ac.in/primality.pdf.

[4] R. Crandall and J. Papadopoulos: *On the implementation of AKS-class primality tests*
http://developer.apple.com/hardware/ve/pdf/aks3.pdf