Der AKS-Primzahltest

Vortrag im Rahmen des Proseminars zur linearen Algebra

Oliver Braun, Sebastian Schönnenbeck

Wintersemester 2009/10

Inhaltsverzeichnis

1. Einleitung

Das Thema dieser Ausarbeitung ist der erste deterministische Polynomialzeit-Primzahltest, der unter dem Namen AKS-Test bekannt wurde. Der Name geht zurück auf die indischen Mathematiker Agrawal, Saxena und Kayal, welche den Algorithmus 2002 der Fachwelt vorstellten [1].

In der vorliegenden Zusammenfassung werden wir uns mit dem AKS-Primzahltest in seiner ursprünglichen Form auseinandersetzen. Dabei orientieren wir uns stark am Aufbau des erwähnten Originalartikels. Im ersten Abschnitt legen wir die grundsätzlichen Bezeichung fest, welche wir im Folgenden verwenden wollen. Anschließend erläutern wir die grundsätzliche Idee hinter dem AKS-Primzahltest, geben den Algorithmus an und führen den Beweis seiner Korrektheit. Im letzten Abschnitt stellen wir zwei mögliche Implementierungen - eine für das CAS Maple und eine in C++ geschriebene Version - vor.

[1] „Primes is in P“ (2002)

2. Bezeichnungen

Neben den mathematischen Standardbezeichnungen verwenden wir folgende Bezeichnungen.

$\mathrm{ld}(a)$ (logarithmus dualis) für $\log_2(a)$.

$O_r(n)$ bezeichnet die Ordnung von n in $(\mathbb{Z}/r\mathbb{Z})^*$: Die kleinste natürliche Zahl a, sodass $n^a \equiv 1 \mod r$, also $\min\{a \in \mathbb{N} \mid n^a \equiv 1 \mod r\}$.

Wir schreiben außerdem für Polynome $q \pmod{x^r - 1, p}$, wenn q im Ring $\mathbb{F}_p[x]/(x^r - 1)\mathbb{F}_p[x]$ betrachtet wird, wo $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z},\ p \in \mathbb{P},\ q \in \mathbb{Z}[x]$.

$\phi(r)$ bezeichne die Euler'sche Phi-Funktion, welche eine natürliche Zahl r auf die Anzahl der zu r teilerfremden Zahlen kleiner r abbildet: $\phi(n) := |\{n \in \mathbb{N} \mid n < r,\ \mathrm{ggT}(r, n) = 1\}|$.

$\mathbb{P}$ bezeichne die Menge aller Primzahlen.

3. Grundidee des Algorithmus'

3.1. Der Satz von Euler-Fermat

Für teilerfremde $a, n \in \mathbb{N}$ gilt:

$$a^{\phi(n)} \equiv 1 \mod n$$

Beweis: Wir betrachten $\{r_1, ..., r_{\phi(n)}\} = (\mathbb{Z}/n\mathbb{Z})^*$, die Menge der invertierbaren Elemente $\mod n$. Für diese Elemente betrachten wir die Abbildung $(\mathbb{Z}/n\mathbb{Z})^* \to (\mathbb{Z}/n\mathbb{Z})^*,\ x \mapsto a \cdot x$ für ein $a \in (\mathbb{Z}/n\mathbb{Z})^*$.

Diese Abbildung ist eine (bijektive) Permutation der invertierbaren Elemente, da $(\mathbb{Z}/n\mathbb{Z})^*$ eine multiplikativ abgeschlossene Gruppe ist.

Daraus folgt $ax \equiv ay \Rightarrow x \equiv y$.

Damit hat man

$$\prod_{i=1}^{\phi(n)} r_i \equiv \prod_{i=1}^{\phi(n)} ar_i \equiv a^{\phi(n)} \prod_{i=1}^{\phi(n)} r_i$$

Da die r_i invertierbar sind, für alle $1 \leq i \leq \phi(n)$, folgt die Behauptung. □

Der von uns in dieser Ausarbeitung zitierte kleine Fermat'sche Satz ist ein Spezialfall dieses Satzes. Er lautet

$$a^{n-1} \equiv 1 \mod n,\ \mathrm{ggT}(a, n) = 1,\ n \in \mathbb{P}$$

Da für $n \in \mathbb{P}$ $\phi(n) = n - 1$ gilt, erhält man den kleinen Fermat'schen Satz offenbar als Konsequenz aus dem Satz von Euler-Fermat.

Der Algorithmus verwendet eine Polynom-Version des kleinen Fermat'schen Satzes. Es gilt nämlich:

3.2. Lemma: AKS-Kriterium

Seien $a, n \in \mathbb{N}$ mit $\text{ggT}(a, n) = 1$. Dann gilt:

$$n \text{ ist Primzahl.}$$
$$\Leftrightarrow$$
$$(x + a)^n \equiv x^n + a \mod n$$

Beweis:

„⇒“:

Sei n eine Primzahl. Für $1 < k < n$ gilt nun $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$. Beachte nun, dass alle Faktoren im Nenner des Quotienten echt kleiner als n sind. Da n aber keine Teiler kleiner n hat, ist der Quotient somit immer noch durch n teilbar:

$$n \mid \binom{n}{k} \; \forall \; 1 < k < n$$
$$\Rightarrow$$
$$(x + a)^n \equiv_n x^n + a^n \equiv^*_n x^n + a$$

* Kleiner Fermat'scher Satz

„⇐“:

Sei n keine Primzahl, p ein Teiler von n und $c = \max\{c \in \mathbb{N} : p^c \mid n\}$. Sei also $n = k \cdot p^c$ für ein $p \nmid k \in \mathbb{N}$. Wir betrachten nun den Binomialkoeffizienten $\binom{n}{p}$:

$$\binom{n}{p} = \frac{n!}{p! \cdot (n-p)!} = \frac{n!}{p! \cdot (kp^c - p)!}$$

Betrachte nun das Auftauchen des Primfaktors p in Zähler und Nenner. In $(kp^c - p)!$ taucht der Primfaktor p genau c mal weniger auf, als in $(kp^c)!$, da es sich bei $kp^c - p$ und (kp^c) um zwei aufeinanderfolgende Vielfache von p handelt. Kürzt man also den Primfaktor p aus diesen auftauchenden Fakultäten bleibt im Zähler die Potenz p^c erhalten.

Es ist jedoch auch $p!$ genau einmal durch p teilbar. Kürzt man diesen Faktor ebenfalls, so verbleibt im Binomialkoeffizient letztlich die Potenz p^{c-1}.

$$\Rightarrow p^c \nmid \binom{n}{p} \Rightarrow n \nmid \binom{n}{p}$$

$$\Rightarrow \binom{n}{p} \neq 0 \mod n \Rightarrow (x+a)^n \neq x^n + a \mod n$$

□

Es wäre nun möglich anhand dieser Identität einen Primzahltest für eine Zahl n zu verwirklichen, indem man eine beliebige Zahl $a \in \mathbb{Z}$ mit $\text{ggT}(a, n) = 1$ wählt und überprüft, ob $(x+a)^n \equiv x^n + a \mod n$. Dies wäre jedoch höchst ineffizient, da im Polynom n Koeffizienten überprüft werden müssten.

Die Idee des Algorithmus' ist nun, das Polynom $(x+a)^n$ nicht nur modulo n sondern auch modulo $x^r - 1$ (für ein geeignetes r) zu betrachten und zu überprüfen, ob die Identität $(x+a)^n \equiv x^n + a \pmod{x^r - 1, n}$ erfüllt ist. Das Problem ist dabei, dass - ähnlich wie beim Kleinen Fermat'schen Satz - auch einige Nicht-Primzahlen die Identität für gewisse a erfüllen.

Im Algorithmus wird die Identität nun für ein geeignetes r für eine Reihe von a gezeigt (wobei sowohl die Anzahl der Überprüfungen als auch die Größe von r nur polynomial von $\text{ld}(n)$ abhängen), was genügt, um die Primeigenschaft von n zu überprüfen.

4. Der Algorithmus

Input: $1 < n \in \mathbb{Z}$

1. If ($n = a^b$ für ein $a \in \mathbb{N}$ und $b > 1$) then „COMPOSITE“
2. Finde das kleinste r mit $o_r(n) > \mathrm{ld}^2(n)$.
3. If ($\exists a \ : \ a \leq r \ \wedge \ 1 < \mathrm{ggT}(a, n) < n$) then „COMPOSITE“
4. If ($n \leq r$) then „PRIME“
5. For $a = 1$ to $\left\lfloor \sqrt{\phi(r)} \cdot \mathrm{ld}(n) \right\rfloor$ do
   If ($(x + a)^n \neq x^n + a \pmod{x^r - 1, n}$) then „COMPOSITE“
6. „PRIME“

5. Beweis der Gültigkeit

Zunächst halten wir fest, dass der Algorithmus für eine Primzahl n immer „PRIME“ zurückgeben wird. Die Rückgabe „COMPOSITE“ kann nur in Schritt 1,3 oder 5 erfolgen. ist n prim, so werden Schritt 1 und 3 niemals „COMPOSITE“ ausgeben. Aus dem AKS-Kriterium folgt außerdem, dass auch Schritt 5 nie „COMPOSITE“ zurückgeben wird. Demnach wird also jede Primzahl auch als solche erkannt.

Betrachten wir nun den Umkehrschluss:

Sollte der Algorithmus n in Schritt 4 als Primzahl erkennen, so muss n Primzahl sein, da ansonsten Schritt 3, welcher alle Zahlen zwischen 1 und r überprüft, einen (nicht-trivialen) Primfaktor von n hätte finden müssen. Wir nehmen also im folgenden immer an, dass der Algorithmus in Schritt 6 „PRIME“ ausgegeben hat. Wir zeigen nun: daraus folgt, dass n eine Primzahl ist.

Wir benötigen nun folgende Hilfsaussage:

5.1. Lemma: Abschätzung des kleinsten gemeinsamen Vielfachen

Bezeichnet $\mathrm{kgV}(\underline{n})$ das kleinste gemeinsame Vielfache der ersten n natürlichen Zahlen, so gilt für $n \geq 7$:

$$\mathrm{kgV}(\underline{n}) \geq 2^n$$

Beweis:

Hilfsaussage:

Es gilt:

$$m \cdot \binom{n}{m} \mid \mathrm{kgV}(\underline{n}) \; \forall \; 1 \leq m \leq n$$

Beweis:

Wir betrachten zunächst das folgende Integral:

$$I = \int_0^1 x^{m-1}(1-x)^{n-m}\,\mathrm{d}x$$

$$\begin{aligned}
&= \int_0^1 x^{m-1} \cdot \sum_{k=0}^{n-m} (-1)^k \binom{n-m}{k} \cdot x^k \,\mathrm{d}x \\
&= \int_0^1 \sum_{k=0}^{n-m} (-1)^k \binom{n-m}{k} \cdot x^{m+k-1} \,\mathrm{d}x \\
&= \left[\sum_{k=0}^{n-m} (-1)^k \cdot \binom{n-m}{k} \cdot \frac{1}{m+k} \cdot x^{m+k} \right]_0^1 \\
&= \sum_{k=0}^{n-m} (-1)^k \cdot \binom{n-m}{k} \cdot \frac{1}{m+k}
\end{aligned}$$

Es gilt aber $k \leq n-m \Rightarrow k+m \leq n$ und somit $k+m \mid \mathrm{kgV}(\underline{n})$. Folglich gilt also auch $I \cdot \mathrm{kgV}(\underline{n}) \in \mathbb{N}$.

Wir zeigen nun durch vollständige Induktion nach $n-m$, dass für beliebige n und $1 \leq m \leq n$ immer gilt $I = \dfrac{1}{m \cdot \binom{n}{m}}$.

IA: Für $n-m = 0 \Leftrightarrow n = m$:

$$\int_0^1 x^{m-1} \cdot (1-x)^0 \,\mathrm{d}x = \frac{1}{m} = \frac{1}{m\binom{n}{n}}$$

IV: Es gelte:

$$\int_0^1 x^{m-1}(1-x)^{n-m} \,\mathrm{d}x = \frac{1}{m\binom{n}{m}}$$

für ein $n-m \geq 0$

IS:

$$\int_0^1 x^{m-2} \cdot (1-x)^{n-m+1} \,\mathrm{d}x$$

partielle Integration:

$$= \left[\frac{1}{m-1} \cdot x^{m-1} \cdot (1-x)^{n-m+1} \right]_0^1 + \int_0^1 \frac{1}{m-1} \cdot x^{m-1} \cdot (n-m+1) \cdot (1-x)^{n-m} \,\mathrm{d}x$$

$$= 0 + \frac{n-m+1}{m-1} \cdot \int_0^1 x^{m-1}(1-x)^{n-m} \, \mathrm{d}x$$

$$\overset{IV}{=} \frac{n-m+1}{m-1} \cdot \frac{1}{m\binom{n}{m}}$$

$$= \frac{(n-m+1) \cdot m! \cdot (n-m)!}{(m-1) \cdot m \cdot n!}$$

$$= \frac{(n-m+1)! \cdot (m-1)!}{(m-1) \cdot n!}$$

$$= \frac{1}{(m-1)\binom{n-m+1}{m-1}}$$

Die Aussage gilt also nach dem Prinzip der vollständigen Induktion. Folglich ist $I = \frac{1}{m \cdot \binom{n}{m}}$ und somit $m \cdot \binom{n}{m} \mid \mathrm{kgV}(\underline{n})$. □

Es gilt also auch:

$$(2n+1) \cdot \binom{2n}{n} = (n+1) \cdot \binom{2n+1}{n+1} \mid \mathrm{kgV}(\underline{2n+1})$$

sowie

$$n \cdot \binom{2n}{n} \mid \mathrm{kgV}(\underline{2n}) \mid \mathrm{kgV}(\underline{2n+1})$$

Da n und $2n+1$ teilerfremd sind gilt folglich auch:

$$n \cdot (2n+1) \cdot \binom{2n}{n} \mid \mathrm{kgV}(\underline{2n+1}) \leq \mathrm{kgV}(\underline{2n+2})$$

Wir zeigen also nun mit vollständiger Induktion nach n:

$$n \cdot (2n+1) \cdot \binom{2n}{n} \geq 2^{2n+2} \; \forall \; n \geq 3$$

IA: $n = 3$:

$$3 \cdot 7 \cdot \binom{6}{3} = 420 \geq 256 = 2^{2 \cdot 3+2}$$

IV: Für ein $n \in \mathbb{N} \geq 3$ gelte nun:

$$n \cdot (2n+1) \cdot \binom{2n}{n} \geq 2^{2n+2}$$

IS:

$$(n+1)(2n+3) \cdot \binom{2n+2}{n+1} = (n+1)(2n+3) \cdot \binom{2n}{n} \cdot \frac{2(2n+1)}{n+1}$$

$$= 2(n+1)(2n+3) \cdot \binom{2n}{n} \cdot \frac{2n}{n+1} = 4n(2n+3) \cdot \binom{2n}{n}$$

$$\geq 4n(2n+1) \cdot \binom{2n}{n} \overset{IV}{\geq} 4 \cdot 2^{2n+2} = 2^{2(n+1)+2}$$

Die Aussage gilt also nach dem Prinzip der vollständigen Induktion für alle $n \in \mathbb{N} \geq 3$. Folglich gilt nun für alle $n \in \mathbb{N} \geq 7$:

$$\mathrm{kgV}(\underline{n}) \geq 2^n$$

Wir zeigen nun zuerst, dass das r, welches der Algorithmus in Schritt 2 sucht, nur polynomial von $\mathrm{ld}(n)$ abhängt.

5.2. Lemma: Existenz von r

Ist $1 < n \in \mathbb{N}$ beliebig so gilt:

$$\exists\, r \leq \max\left\{3, \left\lceil \mathrm{ld}^5(n) \right\rceil\right\} : O_r(n) > \mathrm{ld}^2(n),\ \mathrm{ggT}(r,n) = 1$$

Beweis: Dies ist trivialerweise wahr für $n = 2$ und $n = 3$; $r = 3$ bzw. $r = 5$ erfüllen jeweils alle Bedingungen. Sei also im Folgenden $n > 3$. Setze (der kürzeren Schreibweise halber) $B := \lceil \mathrm{ld}^5(n) \rceil$. (Für $n > 3$ gilt: $B > 10$) Betrachte nun die kleinste Zahl r, die das folgende Produkt nicht teilt:

$$n^{\lfloor \mathrm{ld}(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \mathrm{ld}^2(n) \rfloor} (n^i - 1)$$

Man beachte, dass n und $n^i - 1$ teilerfremd sind für beliebige $i \in \mathbb{N}$. Also gilt dies auch für $n^{\lfloor \mathrm{ld}(B) \rfloor}$ und $\prod_{i=1}^{\lfloor \mathrm{ld}^2(n) \rfloor} (n^i - 1)$.

Nun wollen wir zunächst r nach oben abschätzen:

$$n^{\lfloor \mathrm{ld}(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \mathrm{ld}^2(n) \rfloor} (n^i - 1) \leq n^{\lfloor \mathrm{ld}(B) \rfloor + \frac{1}{2} \mathrm{ld}^2(n) \cdot (\mathrm{ld}^2(n)+1)}$$
$$\leq n^{\mathrm{ld}^4(n)} \leq 2^{\mathrm{ld}^5(n)} \leq 2^B \leq \mathrm{kgV}(\underline{B})$$

Die zweite Ungleichung ist dabei für $3 < n \in \mathbb{N}$ erfüllt.

Wir wissen aber nach der Abschätzung des kgV, dass das kleinste gemeinsame Vielfache der ersten B Zahlen (die erste Zahl, welche von allen Zahlen kleiner als B geteilt wird) größer oder gleich 2^B ist. Damit folgt aber, dass das obige Produkt durch mindestens eine Zahl kleiner B nicht teilbar ist.

Folglich ist also $r \leq B$.

Bemerke, dass $\mathrm{ggT}(r, n)$ nicht durch jeden Primfaktor von n teilbar sein kann, da r ansonsten $n^{\lfloor \mathrm{ld}(B) \rfloor} > B$ teilen würde. Demnach gilt: Auch $\frac{r}{\mathrm{ggT}(r,n)}$ teilt das obige Produkt nicht (die Teilbarkeit des hinteren Teiles wird durch die Division durch $\mathrm{ggT}(r, n)$ nicht beeinflusst). Da aber r minimal gewählt war gilt $\mathrm{ggT}(r, n) = 1$. Da außerdem $n^i - 1 \nmid r \; \forall \; i \in \{1, ..., \lfloor \mathrm{ld}^2(n) \rfloor\}$, ist $o_r(n) > \lfloor \mathrm{ld}^2(n) \rfloor$. □

Da $o_r(n) > 1$ muss es einen Primfaktor p von n geben, für den gilt $o_r(p) > 1$. Wir wissen, dass $p > r$, da wir den Fall betrachten, in dem der Algorithmus für n in Schritt 6 „PRIME“ ausgibt. Außerdem gilt (aus gleichem Grund) $\mathrm{ggT}(n, r) = 1$ und dementsprechend $p, n \in (\mathbb{Z}/r\mathbb{Z})^*$.

Aus Platzgründen bezeichne im Folgenden $\ell := \lfloor \sqrt{\phi(r)} \, \mathrm{ld}(n) \rfloor$.

In Schritt 5 des Algorithmus' werden genau ℓ Gleichungen überprüft. Da die Ausgabe „PRIME“ lautet (nach Voraussetzung), gilt also:

$$(x + a)^n = x^n + a \pmod{x^r - 1, n} \; \forall \; 0 \leq a \leq l$$

(Für $a = 0$ ist die Gleichung trivial) Da p ein Primfaktor von n ist, gilt auch:

$$(x + a)^n = x^n + a \pmod{x^r - 1, p} \; \forall \; 0 \leq a \leq l$$

Nach dem AKS-Kriterium gilt ohnehin:

$$(x + a)^p = x^p + a \pmod{x^r - 1, p} \; \forall \; 0 \leq a \leq l$$

Aus den vorangegangen beiden Gleichungen folgt:

$$(x + a)^{\frac{n}{p}} = x^{\frac{n}{p}} + a \pmod{x^r - 1, p} \; \forall \; 0 \leq a \leq l \qquad (*)$$

da

$$x^n + a = (x + a)^n = ((x + a)^p)^{\frac{n}{p}} = (x^p + a)^{\frac{n}{p}} \pmod{x^r - 1, p}$$

Nach entsprechender Substitution folgt $(*)$, da in $\mathbb{F}_p$ jedes Element eine eindeutige p-te Wurzel hat.

Sowohl p, als auch $\frac{n}{p}$ verhalten sich also bezüglich der obigen Gleichung wie Primzahlen. Da diese Eigenschaft im Folgenden ausgesprochen wichtig ist, wollen wir ihr einen Namen geben.

Definition:

Sei $f(x)$ ein Polynom in$\mathbb{Z}[x]$ und $m \in \mathbb{N}$. m heißt **introspektiv** bezüglich des Polynoms f, falls:

$$[f(x)]^m = f(x^m) \pmod{x^r - 1, p}$$

Im Speziellen sind p und $\frac{n}{p}$ introspektiv bezüglich $x + a$ falls $0 \leq a \leq \ell$.

Wir benötigen nun zwei Eigenschaften introspektiver Zahlen:

5.3. Lemma: Multiplikativität introspektiver Zahlen

Sei $f(X) \in K[X]$ und seien m, m' introspektiv für f. Dann ist $m \cdot m'$ ebenfalls introspektiv für f.

Beweis:

Sei $f(X) \in K[X]$, m und m' introspektiv für f. Dann gilt

$$[f(X)]^{m \cdot m'} = [f(X^m)]^{m'} \pmod{X^r - 1, p}$$

aufgrund der Introspektivität von m.
Da m' introspektiv für f ist, gilt dann auch, mit der Substitution $Y := X^m$:

$$\begin{aligned}[f(Y)]^{m'} &= f(Y^{m'}) \pmod{Y^r - 1, p} \\ &= f(X^{m \cdot m'}) \pmod{X^r - 1, p}\end{aligned}$$

wegen $(X^r - 1) \mid (X^{m \cdot r} - 1)$, denn $(X^r - 1)$ ist ein Faktor von $X^{m \cdot r} - 1$, da $(X^{mr} - 1) = (X^r - 1)(\sum_{i=0}^{m-1} X^{ri})$.

Fasst man diese zwei Gleichungen zusammen, so erhält man $[f(X)]^{m \cdot m'} = f(X^{m \cdot m'})$ $\pmod{X^r - 1, p}$. □

5.4. Lemma: Multiplikativität introspektiver Polynome

Ist $m \in \mathbb{N}$ introspektiv für $f(X)$ und $g(X) \in K[X]$, dann ist es auch introspektiv für $f(X) \cdot g(X)$.

Beweis:

$$\begin{aligned}[f(X) \cdot g(X)]^m &= [f(X)]^m \cdot [g(X)]^m \\ &= f(X^m) \cdot g(X^m) \pmod{X^r - 1, p}\end{aligned}$$

□

Für den weiteren Beweis benötigen wir zwei Gruppen. Wir definieren dazu zunächst folgende Mengen:

$$I := \left\{ \frac{n^i}{p^i} \cdot p^j \mid i, j \in \mathbb{N}_0 \right\}$$

$$P := \left\{ \prod_{a=0}^{\ell} (x+a)^{e_a} \mid e_a \in \mathbb{N}_0 \right\}$$

Man beachte, dass beide Mengen multiplikativ abgeschlossen sind und nach den beiden Lemmata über introspektive Zahlen jede Zahl in I introspektiv bezüglich jedes Polynoms in P ist.

Die erste Gruppe G sei die Gruppe aller Restklassen in I modulo r. Da $\mathrm{ggT}(n,r) = \mathrm{ggT}(p,r) = 1$ ist G eine multiplikative Untergruppe von $(\mathbb{Z}/r\mathbb{Z})^*$. Wir bezeichnen im weiteren $|G| =: t$. G wird von $\frac{n}{p}$ und p modulo r erzeugt. Da $o_r(n) > \mathrm{ld}^2(n)$, gilt also $t > \mathrm{ld}^2(n)$.

5.5. Lemma: Irreduzible Teiler cyclotomischer Polynome

Es existiert ein über $\mathbb{F}_p$ irreduzibler Teiler $h(x)$ von $x^r - 1$ vom Grad $o_r(p)$.

Beweis:

Sei $p \in \mathbb{N}$ eine Primzahl, $\mathbb{F}_p$ der dazgehörige Restklassenkörper sowie $r \in \mathbb{N}$ mit $\mathrm{ggT}(p,r) = 1$.

Wir betrachten nun das r-te cyclotomische Polynom $CP_r(x)$ über F_p. Sei dazu χ eine primitive r-te Einheitswurzel. q bezeichne den Grad des kleinsten Erweiterungskörpers von $\mathbb{F}_p$, welcher χ enthält.

Es gilt also:

$$CP_r(x) = \prod_{\substack{k=1 \\ \mathrm{ggT}(k,r)=1}}^{r} (x - \chi^k)$$

$CP_r(x)$ ist unabhängig von der Wahl von χ und teilt das Polynom $x^r - 1$, da alle χ^k im Produkt primitive r-te Einheitswurzeln über $\mathbb{F}_p$ sind.

Betrachte nun $h(x) := \prod_{k=1}^{o_r(p)} \left(x - \chi^{p^k}\right) = (x - \chi^p) \cdot \left(x - \chi^{p^2}\right) \cdot \ldots \cdot (x - \chi)$. Man beachte nun, dass alle χ^{p^k} im Produkt primitive r-te Einheitswurzeln sind (nach Frobenius-

Automorphismus), also auch in den Linearfaktoren von $CP_r(x)$ auftauchen. Da außerdem $o_r(p) \leq \phi(p)$ gilt folglich $h(x) \mid CP_r(x)$.

Zudem gilt $(h(x))^p$ hat die gleichen Koeffizienten wie $h(x)$, da die Faktoren durch den Frobenius-Automorphismus lediglich permutiert werden. Da die einzigen Elemente in $\mathbb{F}_{p^q}$, die Fixpunkte unter dem Frobenius-Automorphismus sind, bereits die Elemente von $\mathbb{F}_p$ sind, liegt $h(x)$ bereits in $\mathbb{F}_p[x]$.

Zuletzt stellen wir fest, dass $h(x)$ in $\mathbb{F}_p[x]$ irreduzibel ist, weil der Frobenius-Automorphismus „transitiv“ auf den Wurzeln von $h(x)$ ist und somit kein Teilprodukt der Faktoren von $h(x)$ bereits in $\mathbb{F}_p[x]$ liegen kann.

Also existiert ein über $\mathbb{F}_p[x]$ irreduzibler Teiler von $x^r - 1$ vom Grad $o_r(p)$. □

Zur Definition der zweiten Gruppe sei nun $h(x)$ ein solcher irreduzibler Teiler des r-ten cyclotomischen Polynoms über $\mathbb{F}_p$. $h(x)$ hat Grad $o_r(p)$. Die zweite Gruppe Γ sei die multiplikative Gruppe aller Restklassen von Polynomen in P modulo $h(x)$. Diese Gruppe wird erzeugt von $x, x+1, x+2, ..., x+l$ im Körper $F := \mathbb{F}_p[x]/h(x)\mathbb{F}_p[x]$ und ist eine Untergruppe von F.

Im Folgenden werden wir $|\Gamma|$ nach unten abschätzen:

Zunächst halten wir fest, dass $h(x)$ ein Teiler des r-ten cyclotomischen Polynoms und somit auch von $x^r - 1$ ist. Folglich ist x eine primitive r-te Einheitswurzel in F.

Nun seien $f(x)$ und $g(x)$ zwei verschiedene Polynome aus P vom Grad kleiner t. Angenommen es gelte $f(x) = g(x)$ im Körper F. Sei $m \in I$ beliebig. Es gilt also in F auch $[f(x)]^m = [g(x)]^m$. Da m introspektiv ist bezüglich f und g und $h(x) \mid CP_r(x)$ gilt also folglich auch:

$$f(x^m) = g(x^m) \in F$$

Also ist x^m eine Wurzel des Polynoms $Q(Y) = f(Y) - g(Y)$. (Da $\text{ggT}(m, r) = 1 \; \forall \, m \in G$ ist x^m außerdem eine primitive r-te Einheitswurzel in F.) Es existieren also $|G| = t$ verschiedene Wurzeln von Q in F. Nach Wahl von f und g ist aber $\text{Grad}(Q) < t$. Dies ist ein Widerspruch und folglich gilt für zwei verschiedene Polynome vom Grad kleinergleich t aus P immer $f(x) \neq g(x)$ in F.

Wir halten fest, dass

$$i \neq j \in \mathbb{F}_p, \ \forall \ 1 \leq i \neq j \leq \ell$$

da

$$\ell = \left\lfloor \sqrt{\phi(r)} log_2(n) \right\rfloor < \sqrt{r} \cdot log_2(n) < r < p$$

Also sind $x, x+1, x+2, ...x+l$ alle verschieden in F. Da außerdem Grad$(h(x)) > 1$ gilt $x + a \neq 0 \ \forall \ 0 \leq a \leq \ell$. Es gibt also mindestens $\ell + 1$ verschiedene Polynome vom Grad 1 in F. Folglich existieren mindestens

$$\sum_{i=0}^{t-1} \binom{t+\ell}{i} = \binom{t+\ell}{t-1}$$

Polynome vom Grad kleiner t in Γ.

Unter der Voraussetzung, dass es sich bei n nicht um eine Potenz von p handelt, erhalten wir auch eine obere Grenze für $|\Gamma|$.

5.6. Lemma: Obere Grenze für $|\Gamma|$

Ist n keine Potenz von p so gilt:

$$|\Gamma| \leq n^{\sqrt{t}}$$

Beweis:

Betrachten wir folgende Teilmenge von I:

$$I' = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \mid 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor \right\}$$

Ist n keine Potenz von p (hat also $\frac{n}{p}$ nicht nur p als Primfaktor), so gibt es in I' $(\lfloor \sqrt{t} \rfloor + 1)^2 > t$ verschiedene Zahlen. Da aber $|G| = t$ gibt es mindestens zwei modulo r identische Zahlen in I'. Nennen wir diese beiden Zahlen m_1 und m_2 und sei o.B.d.A. $m_1 > m_2$. Es gilt folglich:

$$x^{m_1} = x^{m_2} \pmod{x^r - 1}$$

Sei nun $f(x) \in P$ beliebig. Dann gilt:

$$[f(x)]^{m_1} = f(x^{m_1}) \pmod{x^r - 1, p}$$

$$= f(x^{m_2}) \pmod{x^r - 1, p}$$

$$= [f(x)]^{m_2} \pmod{x^r - 1, p}$$

Also:

$$[f(x)]^{m_1} = [f(x)]^{m_2}$$

im Körper F. Demnach ist $f(x)$ eine Wurzel des Polynoms $Q'(y) = y^{m_1} - y^{m_2}$ im Körper F. $f(x)$ ist aber aus den Elementen von Γ frei wählbar. Das Polynom $Q'(y)$ hat somit wenigstens $|\Gamma|$ verschiedene Wurzeln in F. Es gilt aber:

$$\text{Grad}(Q'(y)) = m_1 \leq \left(\frac{n}{p} \cdot p\right)^{\lfloor\sqrt{t}\rfloor} \leq n^{\sqrt{t}}$$

Wir wissen also nun, dass $|\Gamma| \leq n^{\sqrt{t}}$. □

5.7. Lemma: Abschätzung eines Binomialkoeffizienten

Für $n \in \mathbb{N} \geq 2$ gilt:

$$\binom{2n+1}{n} > 2^{n+1}$$

Beweis durch Induktion nach n:

IA: Für $n = 2$ gilt

$$\binom{5}{2} = 10 > 8 = 2^3$$

IV: Es gelte nun

$$\binom{2n+1}{n} > 2^{n+1}$$

für ein $n \in \mathbb{N} \geq 2$.

IS:

$$\binom{2n+3}{n+1} = \binom{2n+1}{n} \cdot \frac{(2n+2)(2n+3)}{(n+2)(n+1)}$$
$$\overset{IV}{>} 2 \cdot 2^{n+1} \cdot \frac{2n+3}{n+2} \geq 2^{n+2}$$

Die Aussage gilt somit nach dem Prinzip der vollständigen Induktion. □

Angenommen der Algorithmus gibt nun in Schritt 6 „PRIME“ aus und es ist $t = |G|$ und $\ell = \lfloor \sqrt{\phi(r)} \operatorname{ld}(n) \rfloor$. Dann gilt:

$$|\Gamma| \geq \binom{t+\ell}{t-1}$$

Nach Abschätzung von $|\Gamma|$.

$$\geq \binom{\ell + 1 + \lfloor \sqrt{t} \cdot \operatorname{ld}(n) \rfloor}{\lfloor \sqrt{t} \cdot \operatorname{ld}(n) \rfloor}$$

Da $t > \sqrt{t} \cdot \operatorname{ld}(n)$.

$$\geq \binom{2\lfloor \sqrt{t} \cdot \operatorname{ld}(n) \rfloor + 1}{\lfloor \sqrt{t} \cdot \operatorname{ld}(n) \rfloor}$$

Da $\ell = \lfloor \sqrt{\phi(r)} \cdot \operatorname{ld}(n) \rfloor \geq \lfloor \sqrt{t} \cdot \operatorname{ld}(n) \rfloor$.

$$> 2^{\lfloor \sqrt{t} \cdot \operatorname{ld}(n) \rfloor + 1}$$

Nach Lemma (5.7).

$$\geq n^{\sqrt{t}}$$

Nach Lemma 7 wissen wir, dass $|\Gamma| \leq n^{\sqrt{t}}$ falls n keine Potenz von p ist. Wir können also schließen, dass $n = p^k$ für ein $k > 0$. Da der Algorithmus in Schritt 1 nicht „COMPOSITE“ ausgegeben hat, ist aber $k = 1$ und somit $n = p$ eine Primzahl. □

6. Implementierung

Im Folgenden stellen wir zwei (suboptimale) Implementierungen des Algorithmus' vor. Eine realisiert im Computer-Algebra-System Maple 12 und eine in C++ geschriebene Version.

6.1. Implementierung in Maple

Der Code zur Implementierung lautet:

```
1 AKS := proc (n::posint)
2  local b, r;
3  r := 2;
4  with(numtheory,phi);
5  for b i from 2 to floor(log[2](n))] do
6   if floor(n^(1/b))^b = n
7    then return "COMPOSITE"
8   end if;
9  end do;
10 while Ord(r, n) <= evalf(log[2](n)^2)
11  do r := r+1
12 end do;
13 for b from 1 to ceil(r)]
14  do if not (gcd(b,n)=n or gcd(b,n)=1)
15   then return "COMPOSITE"
16   fi;
17  fi;
18 od;
19 if n <= evalf(sqrt(r))
20  then return "PRIME"
21 end if;
22 for b from 1 to ceil(sqrt(phi(r))*log[2](n))] do
23  if rem((x+b)^n-x^n+b, x^r-1, x) mod n <> 0
```

```
24     then return "COMPOSITE"
25    end if;
26   end do;
27   return "PRIME";
28 end proc;
```

Dabei verwenden wir

```
1 Ord := proc (r::posint, n::integer)
2  local o;
3  if gcd(r, n) <> 1
4   then return 0
5  end if;
6  o := 1;
7  while n^o mod r <> 1
8   do o := o+1
9  end do;
10  return o;
11 end proc;
```

Das Programm verwendet die eulersche Phi-Funktion aus dem Paket „numtheory“. Das Programm läuft (unter anderem auf Grund der Speicherineffizienz von Maple) recht ineffektiv. Das Programm „Ord“ ließe sich unter Verwendung des Satzes von Euler-Fermat noch effizienter schreiben. Insbesondere ist nicht gesichert, dass Maple die repeated-squaring Methode zu effektiven Berechnung von n^o mod r verwendet.

Jedoch wäre die Implementierung auch bei hocheffizienter Berechnung von $O_r(n)$ nicht bedeutend schneller, da der eigentlich rechenintensive Teil im Bereich der Polynom-Operationen liegt.

Außerdem existiert noch ein Bug, der dazu führt, dass beim ersten Einbinden des Codes falsche Rückgabewerte entstehen. Dies lässt sich nur durch minimale Veränderungen im Code und anschließendes Neu-Anbinden des Codes umgehen.

6.2. Implementierung in C++

Der Code zur Implementierung lautet:

```
#include <NTL/ZZXFactoring.h>
#include <NTL/ZZ_pX.h>
NTL_CLIENT
int powm(int Base,int Exp,int p)
    {
        int R=1,bs=Base;
        while(Exp>0)
            {
                if(Exp%2==1)
                    {
                        R=(R%p)*(bs%p)%p;
                    }
                bs=(bs%p)*(bs%p)%p;
                Exp >>=1;
            }
         return R;
    }
ZZ_pX powp(ZZ_pX& f, int Exp)
    {
        ZZ_pX R,bs=f;
        SetCoeff(R,0,1);
        while(Exp>0)
            {
                if(Exp%2==1)
                    {
                        R=R*bs;
                    }
                bs=bs*bs;
                Exp>>=1;
            }
```

```
31          return R;
32      }
33  int Ord(int r, int n)
34      {
35          if(GCD(n,r)!=1)
36              {
37                  return 0;
38              }
39          int z=1;
40          while(powm(n,z,r)!=1)
41              {
42                  z++;
43              }
44          return z;
45      }
46  bool AKS(int a,ZZ b)
47      {
48          ZZ_p::init(b);
49          int i;
50          for(i=2; i<log(a)/log(2);i++)
51              {
52                  if(pow(floor(pow(a,1.0/i)),i)==a)
53                  return false;
54              }
55          cerr <<"1";
56          int r=2;
57          while(Ord(r,a)<(log(a)/log(2))*(log(a)/log(2)))
58              {
59                  r++;
60                  //if(r%2==0) cerr << r;
61              }
62          cerr << "2";
63          for(i=1; i<ceil(r)+1;i++)
```

```
64              {
65                  if(!(GCD(a,i)==1  ||  GCD(a,i)==a))
66                      {
67                          return false;
68                      }
69              }
70          cerr << "3";
71          if(a<sqrt(r))
72              {
73                  return true;
74              }
75          ZZ_pX f;
76          SetCoeff(f,0,-1);
77          SetCoeff(f,r,1);
78          ZZ_pX f1,f2,q,eins;
79          SetCoeff(eins,0,1);
80          cerr << "4";
81          for(i=1;i<sqrt(r-1)*log(a)/log(2);i++)
82              {
83
84                  SetCoeff(f2,a,1);
85                  SetCoeff(f2,0,i);
86                  SetCoeff(f1,1,1);
87                  SetCoeff(f1,0,i);
88                  MulMod(q,eins,powp(f1,a)-f2,f);
89                  if(q!=0)
90                      {
91                          return false;
92                      }
93              }
94          return true;
95      }
96  int main()
```

```
97 {
98      int a;
99      ZZ b;
100     cout << "Geben sie a ein:";
101     cin >> a;
102     cout << "Geben sie a erneut ein:";
103     cin >>b;
104     if(AKS(a,b))
105         {
106             cout << "PRIME";
107         }
108    else
109     {
110         cout << "COMPOSITE";
111     }
112    return 0;
113 }
```

Wir verwenden hier das Paket NTL [2] für die modular- und Polynom-Operationen. Aufgrund einiger Inkompatibilitäten muss die zu überprüfende Zahl doppelt eingegeben werden. Außerdem wird auch hier der rechenintensive Teil suboptimal durchgeführt und bereits für verhältnismäßig kleine Zahlen ergibt sich ein Buffer-Overflow. In Ermangelung einer effizienten Implementierung der Euler'schen Phi-Funktion verwenden wir desweiteren die ausgesprochen ungünstige Abschätzung $r - 1$.

Zur Effizienzsteigerung müssten zwei wichtige Schritte unternommen werden. Zum einen ist eine Arithmetik für große Ganzzahlen nötig. Diese lässt sich beispielsweise durch die GMP-Bibliothek realsieren, welche auch mit der NTL kombiniert werden kann. Desweiteren müssten die Polynomoperationen mit Hilfe der NTL effizienter gestaltet werden.

[2] NumberTheoreticLibrary (http://www.shoup.net/ntl/)

7. Laufzeitanalyse

Algorithmen werden nach ihrer Laufzeit klassifiziert, die vom Eingabewert abhängt. Dabei gibt man mit dem Symbol $\mathcal{O}$ eine obere Schranke für die Laufzeit an.

In unserem Fall hängt die Laufzeit von der binären Länge der eingegebenen Zahl ab.

7.1. Lemma: Laufzeit des Algortihmus'

Die Laufzeit des Algorithmus' ist $\mathcal{O}(\mathrm{ld}^{21/2}(n))$.

Beweis: Der erste Schritt des Algorithmus' hat eine Laufzeit von $\mathcal{O}(\mathrm{ld}^3(n))$. Man kann a, b nach oben durch $\mathrm{ld}(n)$ abschätzen, denn

$\mathrm{ld}(n) < n \ \forall n \in \mathbb{N}$
$\Rightarrow$ 2 kommt auf jeden Fall als Basis a in der Potenz a^b vor.
Wegen $2^{\mathrm{ld}(n)} = n$ ist $\mathrm{ld}(n)$ nun eine gültige obere Schranke für a und b, denn die Potenz ist bei fester Basis monoton wachsend und wird n somit überschreiten.

Im zweiten Schritt suchen wir ein r so, dass $o_r(n) > \mathrm{ld}(n)$. Dazu kann man aufeinander folgende r untersuchen und prüfen, ob $n^k \neq 1 \mod r$ für alle $k \leq \mathrm{ld}^2(n)$. Für festes r wird dies $\mathcal{O}(\mathrm{ld}^2(n))$ Multiplikationen modulo r erfordern und somit die Gesamtzeit zu $\mathcal{O}(\mathrm{ld}^2(n) \cdot \mathrm{ld}(r))$ bestimmen. Durch Lemma 5.2 (Existenz von r) wissen wir aber, dass höchstens $\mathcal{O}(\mathrm{ld}^5(r))$ verchiedene r überprüft werden müssen, woraus sich für Schritt 2 eine Laufzeit von $\mathcal{O}(\mathrm{ld}^7(n))$ ergibt.

Laut „Primes is in P“ benötigt jede Berechnung eines ggT in Schritt 3 die Zeit $\mathcal{O}(\mathrm{ld}(n))$. Da wir dies r mal durchführen müssen ergibt sich eine Laufzeit von $\mathcal{O}(r\,\mathrm{ld}(n)) = \mathcal{O}(\mathrm{ld}^6(n))$.
Die Laufzeit des vierten Schrittes ist offenbar $\mathcal{O}(\mathrm{ld}(n))$.

Im fünften Schritt werden $\left\lfloor \sqrt{\phi(r)}\,\mathrm{ld}(n) \right\rfloor$ Gleichungen geprüft. Jede Gleichung benötigt $\mathcal{O}(\mathrm{ld}(n))$ Multiplikationen von Polynomen vom Grad r mit Koeffizienten der Größe $\mathcal{O}(\mathrm{ld}(n))$.

Somit kann jede Gleichung in $\mathcal{O}(r \operatorname{ld}^2(n))$ Schritten überprüft werden. Die Laufzeit von Schritt 5 ergibt sich also als $\mathcal{O}(r\sqrt{\phi(r)} \operatorname{ld}^3(n)) = \mathcal{O}(r^{3/2} \operatorname{ld}^3(n)) = \mathcal{O}(\operatorname{ld}^{21/2}(n))$.

Da diese Laufzeit größer ist als alle anderen vorkommenden ist, gibt sie die Komplexität des Algorithmus' an. □

Im Vergleich dazu haben beispielsweise die probabilistischen Rabin-Miller-Test und Solovay-Strassen-Test eine Laufzeit von $\mathcal{O}(k \cdot \operatorname{ld}^3(n))$ (wo k die Anzahl der geprüften a, also der Basen, ist).
Der AKS-Test läuft also deutlich länger als viele populärere Primzahltests. Es ist eher die - verhältnismäßig spät entdeckte - Existenz eines deterministischen Primzahltests, der in Polynomialzeit terminiert, die den AKS-Test interessant macht.

Es gibt seit seiner Vorstellung Bestrebungen, die Effizienz des AKS-Tests zu steigern, indem man versucht bessere Abschätzungen für r anzugeben und so die Laufzeit zu verkleinern.
Eine weitere Möglichkeit den Test zu beschleunigen liegt in der Gestalt der Gruppe Γ. Kann gezeigt werden, dass diese von einer Menge M mit $|M| < \lfloor\sqrt{\phi(n)} \operatorname{ld}(n)\rfloor$ erzeugt wird, würde dies ebenfalls Schritt 5 des Algorithmus' verkürzen.

Im Jahr 2003 haben Hendrik Lenstra und Carl Pomerance eine Abwandlung des AKS-Tests entwickelt, die eine Laufzeit von $\mathcal{O}(\operatorname{ld}^6(n))$ hat.

A. Danksagungen und Quellenangabe

„Primes is in P“, Agrawal, Saxena, Kayal, 2002

„On the implementation of AKS-class primality tests“, Crandall, Papadopoulos, 2003

„Introduction to Finite Fields and their Applications“, Lidl, Niederreiter, 1994

Wir danken Frau Dr. Natalie Naehrig vom Lehrstuhl D für Mathematik für ihre Unterstützung beim Beweis des Lemmas über cyclotomische Polynome.

Zudem danken wir dem „matheboard.de“-Mitglied *kiste* für die Untertützung beim Beweis zur Abschätzung des kgV.