

Ausarbeitung zu:

Proseminar

Mathematische und logische
Perlen der Informatik

Pratts Primalitätszertifikat

Bearbeitet von Maksym Marchenko

Betreut von Dipl.-Inf. Martin Wildmoser

2005

Inhalt:

1 Primzahl

2 Primzahltest

3 Wer ist V.Pratt?

4 Generator

5 Kleiner Satz von Fermat

6 Primzahlcheck nach Pratt

Generator-Lemma

Primzahl-Inferenzsystem nach Pratt

7 Effizienz

8 Implementierung

Literatur

1. Primzahl

Eine Primzahl ist eine natürliche Zahl, die selbst größer als 1 ist und von allen ganzen Zahlen größer als Null nur durch die Zahl 1 und sich selbst ganzzahlig (d.h. ohne Rest) teilbar ist.

Die derzeit größte bekannte Primzahl ist $2^{224.036.583}-1$, eine Zahl mit 7.235.733 Dezimalstellen, gefunden im Mai 2004 von dem US-Wissenschaftler Josh Findley im Rahmen von George Woltmans GIMPS-Projekt. Für den ersten Primzahlbeweis einer Zahl mit mehr als 10 Millionen Stellen hat die Electronic Frontier Foundation einen Preis von 100.000 US-Dollar ausgeschrieben.

Wozu braucht man Primzahlen ?

Primzahlen finden Anwendung in der Codierungstheorie (Reed-Solomon Code), in der Kryptographie (RSA), sowie in der Mathematik (Zahlentheorie, als Grundlage der Mathematik)

Wie viele gibt's davon?

Euklid hat sich mit der Frage beschäftigt, ob es endlich viele oder unendlich viele Primzahlen gibt. Der Satz von Euklid besagt, dass es unendlich viele Primzahlen gibt. Zum Beweis zeigt er, dass die Annahme, es gebe nur endlich viele Primzahlen, zu einem Widerspruch führt. Nach ihm haben das noch einige andere gezeigt.

2. Primzahltest

Als Primzahltest bezeichnet man ein mathematisches Verfahren, mit dem ermittelt wird, ob eine gegebene Zahl eine Primzahl ist oder nicht.

In der Praxis werden Primzahltests insbesondere bei Verschlüsselungsverfahren in der Kryptographie eingesetzt. Algorithmen wie RSA benötigen Primzahlen in einer Größenordnung von etwa 1000 Stellen in binärer Darstellung. In diesem Bereich gibt es so viele Primzahlen, dass es nicht effizient wäre, diese in einer Liste zu speichern und bei Bedarf einfach darauf zuzugreifen. Auch aus sicherheitstechnischen Gründen ist dieser Ansatz nicht unbedingt empfehlenswert: potentielle Angreifer könnten sich die Struktur der Speicherung zunutze machen, wenn sie das Verschlüsselungsverfahren knacken wollen. Statt der Verwendung einer bekannten Primzahl rät der Algorithmus (mit ein paar Tricks) eine "beliebige" Zahl und stellt mit Hilfe eines Primzahltests möglichst schnell fest, ob diese tatsächlich prim ist.

Das einfache Durchtesten auf Teilbarkeit (Brute Force).

Bei diesem Verfahren ist es nicht notwendig, irgend eine Primzahl zu kennen. Der "naive" Programmierer wird ein Programm folgender Art schreiben:

```
input n
composite=0
do i=2 to (n-1)
```

```

    if (n mod i = 0) then composite = 1
end
if (composite = 0) then print n; ' ist eine Primzahl'

```

Dabei prüft man bei einer Zahl n , ob sie durch eine natürliche Zahl i zwischen 2 und $n-1$ teilbar ist. Ist n durch kein i teilbar, so handelt es sich um eine Primzahl.

Dabei ist es gar nicht notwendig, bis $n-1$ zu testen. Es reicht, bis zu $n^{0,5}$ testen:

Zum testen von Primzahlen existieren auch noch fortgeschrittene Verfahren, auf die hier aber nicht näher eingegangen wird:

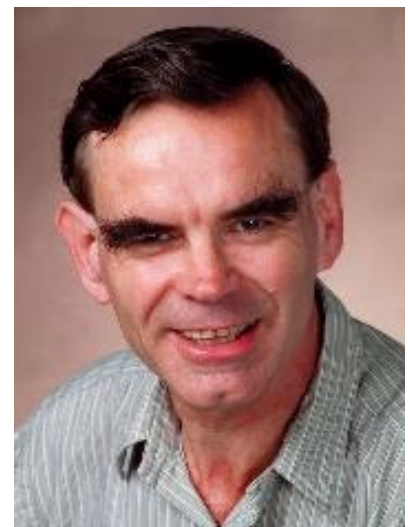
- Rabin-Miller-Test / Solovay-Strassen-Test (probabilistisch) [6]
- Die AKS-Methode (Primzahltest in Polynomialzeit) [7]
- Willans/Wormell Formeln[8]

Das Thema dieser Arbeit ist das Zertifizieren von Primzahlen. Die Idee dabei ist, dass man mit Hilfe von Zusatzinformationen (dem Zertifikat) sehr viel leichter überprüfen kann, ob eine gegebene Zahl prim ist.

3. Wer ist Vaughan Pratt?

Ein entsprechendes Verfahren gibt es von Pratt.

- Informatik Professor an der Universität Stanford
- Mitbegründer von Sun Microsystems in 1982
- Designer von Sun's logo und Pixrect graphics system
- arbeitete im Gebiet natürlicher Sprachen, Algorithmenanalysis und Programmenlogik (1970), Concurrency Modeling, Computergrafik und digitale Typography (1980).
- Bachelor und Master Abschluss von Universität Sydney
- M.Sc. Thesis: Translation of English into Logical Expressions an Sydney University, May 1970. bei Jan B. Hext
- Ph.D. Thesis: Shellsort and Sorting Networks, Stanford University, January 1972 an Stanford University, bei Donald Knuth



Seine Homepage-Adresse ist im Literaturteil zu finden [3].

4. Generator

Bevor wir zum Kern der Arbeit kommen, benötigen wir spezielle Begriffe aus der Algebra.

Definition Generator: $g \in \mathbb{Z}_p \setminus \{0\}$, heißt **Generator** von $\mathbb{Z}_p \setminus \{0\}$

$$\leftrightarrow \mathbb{Z}_p \setminus \{0\} = \{g^0, g^1, \dots, g^{p-2}\} \quad (1)$$

Lemma:

$$g \text{ ist Generator von } \mathbb{Z}_p \setminus \{0\} \leftrightarrow g^x \neq 1 \text{ für alle } x \in \{1, \dots, p-2\} \text{ und } g^{p-1} \equiv 1 \pmod{p} \quad (2)$$

Der Beweis findet sich in [2]. Die Argumentation basiert darauf, daß es einen Zyklus mit weniger als $p-1$ Elementen gibt, falls g kein Generator ist. Da jeder Zyklus die 1 enthält, kann in diesem Falle die rechte Seite nicht erfüllt werden.

Obwohl ein Generator jedes Element erzeugt, ist nicht jedes Element auch ein Generator. In \mathbb{Z}_7 ist 3 beispielsweise ein Generator, während 2 keiner ist.

Beispiel: 3 generiert \mathbb{Z}_7

$3^0=1$	$3^2=9=2 \pmod{7}$	$3^4=81=4 \pmod{7}$
$3^1=3$	$3^3=27=6 \pmod{7}$	$3^5=243=5 \pmod{7}$
		$3^6=729=1 \pmod{7}$

2 ist kein Generator von \mathbb{Z}_7

$2^0=1$	$2^2=4$	$2^4=16=2 \pmod{7}$
$2^1=2$	$2^3=8=1 \pmod{7}$	$2^5=32=4 \pmod{7}$
		$2^6=64=1 \pmod{7}$

5. Kleiner Satz von Fermat

	Für alle	$p \in \mathbb{N}$ mit $p \geq 2$	gilt:
p	Primzahl	$\Leftrightarrow a^{p-1} \equiv 1 \pmod{p}$	für alle $a \in \mathbb{Z}_p \setminus \{0\}$.

Lemma:

$$p \text{ Primzahl} \quad \Leftrightarrow \quad \mathbb{Z}_p \setminus \{0\} \text{ hat einen Generator } g.$$

Beweis:

- \Leftarrow Sei g ein Generator von $\mathbb{Z}_p \setminus \{0\}$
- (1) Für alle $a \in \mathbb{Z}_p$ gibt es ein $k \in \{0, \dots, p-2\}$ so daß $a = g^k$
 - (2) $g^{p-1} \equiv 1 \pmod{p}$ gilt für jeden Generator
 - (3) $a^{p-1} = (g^k)^{p-1} = (g^{p-1})^k = 1^k = 1 \quad \square$

⇒ Satz 5.82, Satz 5.92 in [2]

Der erste Satz besagt, daß es für jede Primzahlpotenz einen endlichen Körper gibt. Der zweite Satz sagt, daß die multiplikative Gruppe eines endlichen Körpers einen Generator hat. □

Anwendungsbeispiel:

Aus

$$3^6 = 1 \pmod{7} \quad (3)$$

und

$$3^x \neq 1 \pmod{7} \text{ für alle } x=1, \dots, 5 \quad (4)$$

folgt (wegen „kleiner Fermat“) 7 ist eine Primzahl.

6. Primzahlcheck nach Pratt

Pratts Verfahren stellt eine Verbesserung des Fermatlemmas dar. Die Idee ist genau wie vorhin Primalität mit Hilfe eines Generators nachzuweisen. Das Problem besteht also darin, nachzuweisen daß eine gegebene Zahl g ein Generator von $\mathbb{Z}_p \setminus \{0\}$ ist.

Anstatt zu Testen daß $g^k \neq 1$ für alle $k \in \{0, \dots, p-2\}$, reicht es aus mit $g^{\frac{p-1}{q}} \neq 1$ alle Primteiler q von $p-1$ zu testen. Das sind erheblich weniger Fälle.

Generator-Lemma:

$$\begin{aligned} & x^k \neq 1 \pmod{p} \text{ für alle } k \in \{1, \dots, p-2\} \text{ g.d.w.} \\ & \left(x^{\frac{p-1}{q}} \neq 1 \pmod{p} \text{ für alle Primteiler } q \text{ von } p-1 \right. \\ & \quad \left. \text{und } x^{p-1} = 1 \pmod{p} \right) \end{aligned}$$

Beweis:

Beweis: alternativer Beweis in [1].

$$\Rightarrow \frac{p-1}{q} \in \{1, \dots, p-2\} \quad \square$$

⇐ (1) Sei i die kleinste Zahl < 0 so daß $x^i = 1 \pmod{p}$. D.h. i ist die Ordnung von x in $\mathbb{Z}_p \setminus \{0\}$.

(2) Wähle c und d so, daß $p-1 = c \cdot i + d$ und

(2') $0 \leq d < i$

(3) Es gilt $x^{ci} = (x^i)^c = 1^c = 1 \pmod{p}$ (wegen (1))

(4) Es gilt $x^{ci+d} = x^{p-1} = 1 \pmod{p}$ (wegen (2) und Annahme)

(5) Es gilt $x^{ci} * x^d = x^{ci+d} = 1 \pmod{p}$

$$1 * x^d = 1 \pmod{p} \implies x^d = 1$$

(6) $d = 0$ (weil sonst i nicht minimal (2') und (5))

(7) $i \mid p-1$, d.h. es gibt k , so daß $p-1 = k * i$ (wegen (2) und (6))

(8) $\frac{p-1}{q} \pmod{i} \neq 0$ für alle Primteiler q von $p-1$

Beweis durch Widerspruch:

Angenommen $i \mid \frac{p-1}{q}$, $x^{\frac{p-1}{q}} \neq 1 \pmod{p}$ wegen Annahme

Aber von $i \mid \frac{p-1}{q}$ folgt, das es ein z gibt mit $\frac{p-1}{q} = z * i$, d.h. $x^{\frac{p-1}{q}} = x^{zi} = (x^i)^z = 1^z = 1$.

Dies ist ein Widerspruch zu $x^{\frac{p-1}{q}} \neq 1 \pmod{p}$ \square

(9) Fallunterscheidung:

1. Fall $k=1$: $i = p-1$ (aus (7), wenn $k=1$ ist)

2. Fall $k>1$: wegen des Fundamentalsatzes der Algebra hat k eine

Primfaktorzerlegung der Form $k = q_1^{k_1} \dots q_n^{k_n}$. Dann gilt

$$\frac{p-1}{q} = \frac{k * i}{q} = \frac{q_1^{k_1} * \dots * q_n^{k_n} * i}{q}$$

d.h. $i \mid \frac{p-1}{q}$, was Widerspruch zu (8) ist \square

Das Generator-Lemma liefert somit einen effizienten Nachweis für Primalität.

Anwendungsbeispiel:

Aus $3^6 \equiv 1 \pmod{7}$ (3)

und $6 = 2 * 3$ (5)

und 2 ist Primzahl (6)

und 3 ist Primzahl (7)

und $3^{6/2} = 3^3 = 27 \equiv 6 \pmod{7} \neq 1$ (8)

und $3^{6/3} = 3^2 = 9 \equiv 2 \pmod{7} \neq 1$ (9)

folgt 7 ist eine Primzahl

Primzahl-Inferenzsystem nach Pratt

In dem Paper von Pratt wird das Generator-Lemma nicht explizit erwähnt. Dieses Lemma wurde als Teil dieser Arbeit selbständig hergeleitet um den Zusammenhang zum kleinen Satz von Fermat zu verdeutlichen. Pratt verwendet stattdessen ein Inferenzsystem an, mit dem Primzahlität korrekt und vollständig nachgewiesen werden kann.

Formeln:

Prime p „ p ist Primzahl“

(p,g,x) „für jeden Primteiler q von x gilt $g^{\frac{x}{q}} \neq 1 \pmod{p}$ “

Regeln:

$p, g \in \mathbb{N} \vdash (p, g, 1)$	(Ax)
$g^{\frac{x}{q}} \neq 1 \pmod{p}, (p, g, x), \text{Prime } q \vdash (p, g, xq)$	(R ₁)
$g^{p-1} = 1 \pmod{p}, (p, x, p-1) \vdash \text{Prime } p$	(R ₂)

Beispiel:

(1)	(2,1,1)	Axiom;
(2)	Prime 2	(1), R ₂ , $1^1 = 1 \pmod{2}$ Axiom;
(3)	(3,2,1)	(3),(2), R ₁ , $2^1 = 2 \pmod{3}$
(4)	(3,2,2)	(4), R ₂ , $2^2 = 1 \pmod{3}$
(5)	Prime 3	Axiom;
(6)	(7,3,1)	(6),(2), R ₁ , $3^4 = 4 \pmod{7}$
(7)	(7,3,2)	(7),(2), R ₁ , $3^4 = 4 \pmod{7}$
(8)	(7,3,6)	(8), R ₂ , $3^4 = 1 \pmod{7}$
(9)	Prime 7	<u>Zertifikat</u>

Theorem : $\vdash \text{Prime } p \equiv p \text{ ist eine Primzahl}$

Korrektheit: $\vdash \text{Prime } p \Rightarrow p \text{ ist eine Primzahl}$

Beweis: letzte Regel war R₂ mit einem geeigneten Generator g .

d.h. $g^{p-1} = 1 \pmod{p}$ (F₁)

d.h. $\vdash (p, g, p-1)$ (F₂)

Wegen F_2 wissen wir, dass $p-1$ als Produkt von Primfaktoren $p-1=q_1*q_2*\dots*q_m$ darstellbar ist und $\vdash (p,g,p-1)$ aus Regeln der Form R_1 ableitbar ist.

$$g^{\frac{p-1}{q}} \not\equiv 1 \pmod{p} \quad (F_3)$$

d.h. p ist Primzahl (wg. Generator-Lemma, kl.Fermat)

Vollständigkeit: p ist eine Primzahl $\Rightarrow \vdash \text{Prime } p$

Induktion über p :

Falls p Primzahl ist, dann hat Z_p einen Generator g .

Induktionsanfang: Fangen wir an mit Axiom $(p,g,1)$ für solchen Generator g .

Induktionshypothese: jeder Primfaktor q von $p-1$ ist als Theorem ableitbar,

d.h. $\vdash \text{Prime } q$. Für jeden Primfaktor q gilt $x^{\frac{p-1}{q}} \not\equiv 1 \pmod{p}$;

weil $\frac{p-1}{q} \in \{1, \dots, p-2\}$ und x ein Generator ist.

Eine Deduktion von (p,g,x) , wo x ein Produkt von Primfaktoren der Zahl $p-1$, ist möglich. Es folgt $(p,g,p-1)$ kann abgeleitet werden, und weil $g^{p-1} \equiv 1 \pmod{p}$, können wir Prime p ableiten.

7. Effizienz

In Pratts Paper findet sich ein Beweis, daß falls $\vdash \text{Prime } p$ gilt, es immer eine Ableitung in höchstens $6 * \lg p - 4$ Zeilen gibt. Damit ist garantiert, daß das Überprüfen eines Primzahlzertifikates effizient möglich ist. Ganz anders dies aber für die Generation solcher Zertifikate. Zur Zeit sind keine Verfahren bekannt, mit dem man effizient Generatoren finden kann.

8. Implementierung

Teil dieser Arbeit war die Implementierung in OCaml.

Wie man aus dem Beispiel sieht, sind Zertifikatcheck und Zertifikatgenerierung die Hauptteile der Implementierung.

Der Zertifikatcheck basiert auf Modulorechnung und Potenzberechnung. Das Erste ist schon in OCaml vordefiniert. Die vordefinierte Potenzberechnung ($**$) gilt aber nur für Reelle Zahlen. Wegen strenger Typisierung in OCaml, können wir diese Potenzberechnung nicht benutzen, da wir mit ganzen Zahlen arbeiten. Daher definieren wir für uns diese Funktion selber. Wir verwenden den effizienten „binary square and multiply“ Algorithmus.

```
let expo (m,n) = match n with
| 0 -> 1
| n when (n mod 2 = 0) -> expo (m, n/2) * expo (m, n/2)
| n -> m * expo (m, (n-1) / 2) * expo (m, (n-1) / 2) ; ;
```

Die Zertifikatgenerierung besteht im wesentlichen auch aus zwei Teilen – Primfaktorzerlegung und Generatorsuche.

Für Generatorsuche ist zur Zeit keine effizienter Algorithmus bekannt (wobei Pratt Papier über Effizienz der Generatorsuche auch keine Bemerkungen enthält). Man kann aber alle Zahlen ausprobieren, bis die richtige gefunden wird.

Für die Primfaktorzerlegung wenden wir die heuristische Pollard-Rho Methode an [5]. Die sieht folgendermaßen aus:

```
let rho n =
let i = 1;
let x = rand(0, n-1);
let y = x;
let k = 2;
while true do
  i:=i+1;
  x = (x*x - 1) mod n;
  d = ggT(y-x, n);
  if d!=1 && d!=n then insertElement(d, Primliste)
  if i=k then
    begin
      y:=x;
      k:=2*k
    end
done;;
```

Den ggT können wir effizient mit Euklids Algorithmus berechnen:

```
let rec ggT (a, b) =
if (a mod b = 0) then a
else ggT(b, a mod b);;
```

Die Pollard-Rho Methode garantiert nicht, daß wir die Zahlen in absteigender Reihenfolge bekommen – die werden wir aber in Zertifikatcheck brauchen. Deshalb sortieren wir die Primliste aus Pollard-Rho nachträglich mit quicksort

```
let rec quicksort liste = match with
| [] -> []
| [e] -> [e]
| (e::rest) -> quicksort(lowerpart(e, liste)) @
  equalpart(e, liste) @ quicksort(upperpart(e, liste));;

let rec lowerpart (el, liste) = match (el, liste) with
| (el, []) -> []
| (el, a::rest) when e<el -> a::lowerpart (el, rest)
| (el, a::rest) -> lowerpart (el, rest);;
```

wobei upperpart und equalpart analog zu lowerpart zu definieren sind.

Literatur:

- 1 V.R.Pratt, „Every prime has a succinct certificate“, SIAM J. Computing 4 (1975), 214-220
- 2 A.Steger, „Diskrete Strukturen - Band 1“, Springer Verlag (2001), 256
- 3 E.Chailoux, P.Manoury, B.Pagano, „Developing Applications with Objective Caml“, O'Reilly (1999), 732
- 4 Wim van Dam, „Primes $\in NP$ “, UC Berkley
- 5 Cormen T.H., Leiserson C.E., Rivest R. L., Introduction to Algorithms 8th printing 1992, MIT press
- 6 <http://mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html>
- 7 <http://plus.maths.org/issue22/news/prime/>
- 8 Hacker's Delight By Henry S. Warren Jr..Published by Addison Wesley Professional.
- 9 <http://boole.stanford.edu/pratt.html>
- 10 <http://de.wikipedia.org/wiki/Primzahlen>
- 11 http://de.wikipedia.org/wiki/Kleiner_Fermatscher_Satz
- 12 <http://www.cs.rutgers.edu/~chvatal/notes/ppp/ppp.html>
- 13 <http://www.primzahlen.de/>