

- [3] Wieb Bosma, John J. Cannon, Catherine Playoust, *The Magma Algebra System I: The User Language*, J. Symb. Comput. **24** (1997), 235–265.
- [4] Harm Derksen, Gregor Kemper, *Computational Invariant Theory*, Encyclopaedia of Mathematical Sciences **130**, Springer-Verlag, Berlin, Heidelberg, New York 2002.
- [5] Theo de Jong, *An Algorithm for Computing the Integral Closure*, J. Symb. Comput. **26** (1998), 273–277.
- [6] Gregor Kemper, *Computing Invariants of Reductive Groups in Positive Characteristic*, Transformation Groups **8** (2003), 159–176.
- [7] Martin Kreuzer, Lorenzo Robbiano, *Computational Commutative Algebra 1*, Springer-Verlag, Berlin 2000.
- [8] P. E. Newstead, *Introduction to Moduli Problems and Orbit Spaces*, Springer-Verlag, Berlin, Heidelberg, New York 1978.
- [9] Vladimir L. Popov, *On Hilbert's Theorem on Invariants*, Dokl. Akad. Nauk SSSR **249** (1979), English translation Soviet Math. Dokl. **20** (1979), 1318–1322.
- [10] Bernd Sturmfels, *Algorithms in Invariant Theory*, Springer-Verlag, Wien, New York 1993.
- [11] Wolmer V. Vasconcelos, *Computational Methods in Commutative Algebra and Algebraic Geometry*, Algorithms and Computation in Mathematics **2**, Springer-Verlag, Berlin, Heidelberg, New York 1998.

Der folgende Artikel ist unter gleichem Titel in den Mitteilungen der Deutschen Mathematiker-Vereinigung, Heft 4-2003, erschienen. Wir bedanken uns für die Erlaubnis, ihn hier abdrucken zu dürfen.

Primzahl-Rekordjagd

Günter M. Ziegler (Berlin)

ziegler@math.tu-berlin.de



Der Dezember 2003 beschert uns mehrere Primzahl-Rekorde. So wurde unter der Regie von Jens Franke (Bonn) das RSA-576 Entschlüsselungsproblem gelöst: die Faktorisierung einer 174-stelligen Dezimalzahl. Die größte bekannte Primzahl ist ebenfalls neu, eine Mersennesche Primzahl mit insgesamt 6.320430 Stellen:

$$M = 2^{20.996.011} - 1.$$

Die Medien (unter anderem Spiegel-Online vom 3. Dezember) schreiben die Entdeckung einem Studenten der Verfahrenstechnik an der Michigan State University namens Michael Shafer zu – aber das ist nur ein Teil der Wahrheit.

Mersennesche Zahlen

Seit Januar 1996 läuft im Internet eine Suche nach immer größeren Mersenneschen Primzahlen. In dem verteilten Rechenprojekt unter dem Titel GIMPS („Great Internet Mersenne Prime Search“, www.mersenne.org), können Freiwillige übers Internet die GIMPS-Computerprogramme abrufen und „ihre“ Zahlen zum Testen zugeteilt bekommen, ihre PCs damit Sklavenarbeit leisten lassen, und die Rückmeldung übers Internet abliefern.

Mönches Marin Mersenne (1588-1648) heißen die Zahlen der Form $M_n = 2^n - 1$ Mersennesche Primzahlen – wenn sie prim sind. Dafür ist notwendig (schöne Übungsaufgabe aus der elementaren Zahlentheorie), dass n selbst prim ist. Aber hinreichend ist das nicht: $n = 11$ liefert das erste Gegenbeispiel. Im Jahr 1644 behauptete Mersenne, dass M_n für $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127$ und 257 prim sei, aber keine andere Primzahl unter 257 (womit er exakt fünfmal danebengelegt hat).

Zur Erinnerung: zu Ehren des französischen

Mersennesche Primzahlen sind ziemlich selten:



Marin Mersenne, 1588 – 1648 (Quelle: <http://www-groups.dcs.st-and.ac.uk/~history/PictDisplay/Mersenne.html>)

Man weiß nicht, ob es unendlich viele gibt, und man kennt inzwischen die ersten 38 davon, und nur zwei weitere, darunter die neu gefundene $M_{20.996.011}$, die auch die größte bekannte Primzahl überhaupt ist.

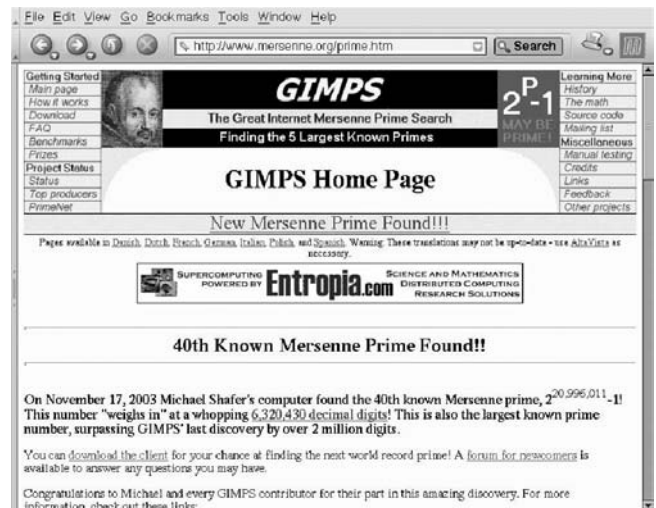
Dass man Zahlen mit mehr als 6 Millionen Stellen effektiv auf Primalität testen kann, ist die eigentliche wissenschaftliche (und programmiererische) Höchstleistung hinter dem neuen Rekord – dass $n = 20.996.011$ prim sein muss, ist ja nur eine klitzekleine Aufwärmübung für den neuen Rekord.

Primalitätstests

Nun weiß man seit Kurzem, dass es exakte Primzahltests gibt, die in Polynomzeit laufen – siehe *DMV Mitteilungen* 4/2002, S. 14–21. Diese stellen einen theoretischen Durchbruch dar, sind aber für den Einsatz in der Praxis (noch) nicht geeignet. Im GIMPS-Projekt wird für jedes prime n eine Kaskade von klassischeren Tests durchlaufen, die unter www.mersenne.org/math.htm sehr schön und kapierbar beschrieben werden.¹ In *Phase I* sucht man nach kleinen Primteilern q von $2^n - 1$. Diese müssen (wieder eine hübsche Übungsaufgabe) $q \equiv 1 \pmod{2n}$ und $q \equiv \pm 1 \pmod{8}$ erfüllen. Mithilfe eines auf solche Faktoren zugeschnittenen „Sieb des Eratosthenes“ werden dann Primteiler von M_n bis ca. 40.000 erkannt. Dabei kann ausgenutzt werden, dass Teilbarkeitstests für Zahlen vom Typ $2^n - 1$ in Binärrithmetik sehr effektiv durchgeführt werden können.

¹Zur algorithmischen Primzahltheorie empfehlen die Experten RICHARD CRANDELL & CARL POMERANCE: “Prime Numbers. A Computational Perspective”, Springer-Verlag, New York 2001. Aus Computeralgebra-Perspektive finden sich Primzahltests (und sehr viel mehr Spannendes) in JOACHIM VON ZUR GATHEN & JÜRGEN GERHARD: “Modern Computer Algebra”, Cambridge University Press, 2. Auflage 2003.

In *Phase II* wird dann ein Spezialfall der sogenannten $(p-1)$ -Methode von Pollard (1974) verwendet, mit der man Faktoren $q = 2kn + 1$ finden kann, für die $q - 1 = 2kn$ aus vielen kleinen Primfaktoren besteht, oder aber (in einer verbesserten Version) bis auf einen etwas größeren Primfaktor stark zusammengesetzt ist: Wenn man q sucht, so dass alle Primfaktoren kleiner als B sind, so bildet man dafür das Produkt $E := \prod_{p < B} p$ aller Primzahlen, die kleiner als B sind, und berechnet dann $x := 3^{E/2n}$. Im ggT von $x - 1$ und $2^n - 1$ fängt man dann den gesuchten Teiler von $2^n - 1$.



<http://www.mersenne.org>

Erst in *Phase III* verwendet man dann ein Verfahren, mit dem man sicher entscheiden kann, ob $2^n - 1$ prim ist, den sogenannten Lucas–Lehmer Test (1878, 1930/1935) für Mersenne-Zahlen: M_n ist genau dann prim, wenn $\ell_{n-1} \equiv 0 \pmod{M_n}$ gilt, wobei die ℓ_k durch $\ell_1 = 4$ und $\ell_n = \ell_{n-1}^2 - 2$ rekursiv definiert werden. Um das effektiv zu berechnen, muss man riesige Zahlen schnell modulo $2^n - 1$ quadrieren. Dazu werden die Zahlen in große Blöcke unterteilt, und dann arbeitet man mit Spezialversionen einer schnellen Fourier Transformation („Fast Fourier Transform“, FFT), in diesem Fall mit einer FFT bezüglich einer irrationalen Basis, die von Richard Crandell und Barry Fagin (*Mathematics of Computation* 1994) eingeführt wurde. Auf den WWW-Seiten des *Mathematica*-Projekts mathworld.wolfram.com, die die aktuelle Rekordmeldung verbreiten, wird suggeriert, GIMPS würde mit einer *Mathematica*-Implementierung arbeiten, aber das ist eine arge Dehnung der Tatsachen. (Es hat nur Crandell die Methode auch für die Primzahltests von *Mathematica* implementiert.) In der Tat arbeitet GIMPS mit hochoptimiertem Assembler-Code, aus Prozessorarchitekturgründen in Gleitkommaarithmetik, deren Fehler getrennt erkannt und aufgefangen werden müssen.

Primalität und Faktorisierung

Phasen I und II des GIMPS-Verfahrens spucken also im Fall von zusammengesetztem M_n wirklich Teiler aus – wenn sie welche finden –, die dritte und entscheidende Phase aber nicht mehr. Die Antwort heißt da dann nur noch „zusammengesetzt!“, ohne einen expliziten (Prim-)Teiler als Beweis. Es wird also ein Primalitätstest durchgeführt, aber kein vollständiges Faktorisierungsverfahren.

Und das ist auch gut so: Nicht einmal für den Spezialfall von Mersenne-Zahlen kennt man effektive Verfahren zum Faktorisieren. Ein Verfahren, mit dem man beliebige Zahlen mit ein paar Hundert Stellen faktorisieren könnte, wäre interessant und bedrohlich, weil die kryptographischen Verfahren, die die Sicherheit von Online-Banking und Internet garantieren sollen, darauf beruhen, dass das Faktorisieren und verwandte Probleme (wie die Berechnung von „diskreten Logarithmen“) offenbar schwer sind.

RSA

Ein Beispiel dafür ist das von Ron Rivest, Adi Shamir und Leonard Adleman 1978 publizierte Verschlüsselungsverfahren „mit öffentlichen Schlüsseln“, das sich inzwischen in fast jedem elementaren Zahlentheorie-Lehrbuch findet, gleichzeitig aber auch in der Praxis vielfältig zum Einsatz kommt – siehe die Homepage <http://www.rsasecurity.co> der Firma von Rivest, Shamir und Adleman. Die Sicherheit des Verfahrens gegen unerlaubtes Entschlüsseln hängt davon ab, dass es mit heutiger Technologie sehr schwer ist, Produkte von Zahlen mit 150-200 Stellen in ihre Primfaktoren zu zerlegen. Die Firma „RSA Securities“ hat sogar Preise auf Beispielp Probleme² ausgesetzt. Der erste davon ist/war ein Preis von 10.000 Dollar für das Faktorisieren der Zahl „RSA-576“

188198812920607963838697239461650439807163563
379417382700763356422988859715234665485319060
606504743045317388011303396716199692321205734
031879550656996221305168759307650257059

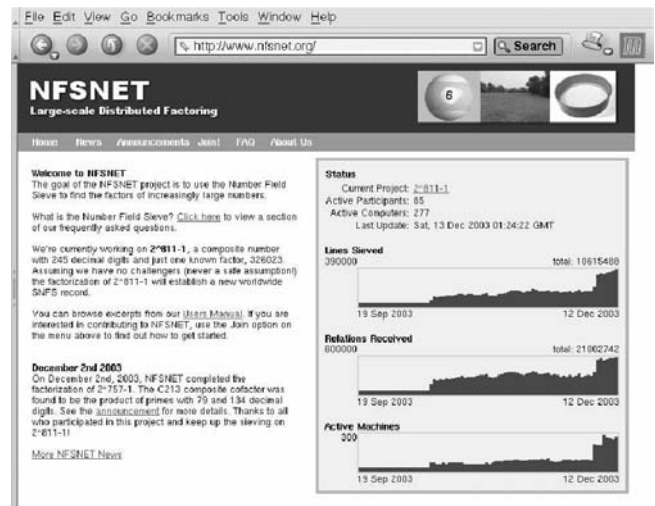
mit 174 Dezimalziffern, bzw. 576 Binärziffern (bits). Und dieses Problem hat Jens Franke von der Universität Bonn jetzt geknackt, wie *Heise Online* am 8. Dezember gemeldet hat: die Zahl hat Faktoren

398075086424064937397125500550386491199064362
342526708406385189575946388957261768583317
und

472772146107435302536223071973048224632914695
302097116459852171130520711256363590397527

(mit je 87 Ziffern), und die sind prim – was wiederum mit den aktuellen Methoden ganz leicht zu zeigen ist. Franke verwendete dabei das „General Number Field Sieve (GNFS)“. Dieses wurde von Lenstra, Lenstra, Manasse & Pollard 1990 eingeführt, und hat eine Laufzeit von $\exp(O(\sqrt[3]{n} \log n))$ für n -stellige Zahlen; es ist also nicht ganz polynomial, aber *fast*. Unter Verwendung des GNFS wurden auch schon die kleineren Testproble-

me von RSA-100 bis RSA-512 geknackt (letzteres im August 1999).



<http://www.nfsnet.org>

Und es gibt noch mehr aktuelle Rekorde, die sich ebenfalls aufs Faktorisieren beziehen: Unter Anderem versucht man eben Mersenne-Zahlen nicht nur auf Primalität zu untersuchen, sondern auch vollständig in Primfaktoren zu zerlegen. NFSNET (<http://www.nfsnet.org>) ist auch ein Internet-Projekt, dem es nun (Erfolgsmeldung vom 2. Dezember) in Internet-Gemeinschaftsarbeit gelang die Mersennesche Zahl $2^{757} - 1$ vollständig zu faktorisieren: Die Primfaktoren 9815263 und 561595591 dieser Zahl kannte man schon länger, aber der 212-stellige Rest war ein hartes Stück Arbeit: er wurde jetzt in die Primfaktoren 572213702200206782424822797509585774915131282 7809388406962346253182128916964593

und

240338216409835080887362734030059654466890023
563443321305650666431938139011197710904242694
12054543072714914742665677774247325292327559
zerlegt. Dieser Erfolg basiert auf dem „Special Number Field Sieve (SNFS)“ – einer schnelleren Spezialversion des GNFS, die nur für spezielle Zahlen, etwa vom Typ $b^n \pm 1$, anwendbar ist.

Rekordjagd

Die Rekordjagd geht weiter. Die „Electronic Frontier Foundation“ (<http://www EFF.org/>) hat schon im Jahr 2000 einmal 50.000 Dollar für die erste Primzahl mit einer Million Stellen ausgezahlt. Für die Identifikation einer Primzahl mit mehr als 10 Millionen Dezimalstellen hat sie 100.000 Dollar ausgesetzt. Dies heizt die Stimmung an, und das GIMPS-Projekt sucht Mitstreiter, die ihre Computer für die Rekordjagd einsetzen wollen.

Genauso ist man natürlich hinter den größeren RSA-Problemen hinterher; als nächstes wartet da RSA-640,

²<http://www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html>

eine Zahl mit 193 Dezimalstellen, auf ihre Zerlegung. Darauf sind 20.000 Dollar ausgesetzt.

Und die nächste Mersenne-Zahl auf der Abschuss- bzw. Zerlegungsliste von NFSNET ist $2^{811} - 1$. Auch für dieses Projekt werden noch Mitstreiter gesucht.

Viele arme kleine PCs werden also mit Zahlen gefüttert und mit Primzahltests und mit Zerlegungsverfahren gequält werden, nur damit Herrchen vielleicht einen Teil des Ruhms (und des Preisgeldes) einkassieren kann.

Neues über Systeme

GiNaC – eine C++-Bibliothek für symbolisches Rechnen

Christian Bauer (Mainz)

Über GiNaC Handelsübliche Computeralgebrasysteme bieten typischerweise weitreichende algebraische Fähigkeiten und eine umfassende Sammlung an Funktionen aus allen Bereichen der reinen und angewandten Mathematik, eingebettet in eine interaktive Benutzerumgebung mit einfachen Programmiermöglichkeiten. GiNaC (ein rekursives Akronym für „*GiNaC is not a computer algebra system*“) verfolgt hier einen anderen Ansatz: Es erweitert die existierende Programmiersprache C++ um Klassen und Funktionen zum symbolischen Rechnen.

Ein Beispiel Hier ist ein vollständiges C++-Programm, das mit Hilfe von GiNaC Laguerre-Polynome nach der Formel von Rodrigues berechnet und diese im L^AT_EX-Format ausgibt:

```
#include <iostream>
using std::cout; using std::endl;

#include <ginac/ginac.h>
using namespace GiNaC;

ex laguerre(const symbol & x, unsigned n)
\{
    ex L_n = diff(pow(x, n)*exp(-x), x, n)
              / (exp(-x) * factorial(n));
    return L_n.normal();
\}

int main()
\{
    symbol x("x");
    cout << latex << laguerre(x, 4) << endl;
\}
```

Die Ausgabe des Programms $(1 + \frac{3}{2}x^2 - \frac{1}{6}x^3 - 3x)$ kann direkt in L^AT_EX-Dokumente übernommen werden.

Die wichtigsten von GiNaC zur Verfügung gestellten Datentypen sind die Klasse `ex` (kurz für „*expression*“), die einen beliebigen algebraischen Ausdruck speichert und die Klasse `symbol`, die eine symbolische Variable repräsentiert. Da die Namen von C++-Variablen nach der Kompilierung nicht mehr zugänglich sind,

muss beim Erzeugen von Symbolen ein Name angegeben werden, der zur Ausgabe des Symbols verwendet wird, in diesem Fall „`x`“.

Symbolische Ausdrücke lassen sich mit GiNaC in genau der gleichen Weise hinschreiben wie bei rein numerischen Ausdrücken in C++ üblich. Die Funktion `diff(e, x, n)` berechnet die n -fache symbolische Ableitung des Ausdrucks e nach der Variablen x und die Methode `.normal()` bringt rationale Ausdrücke auf die Form eines vollständig gekürzten Bruches, wobei in diesem Fall der gemeinsame Faktor e^{-x} herausdividiert wird.

Was kann GiNaC (und was kann es nicht)? Das Ziel des GiNaC-Projekts ist die Entwicklung eines offenen und erweiterbaren Werkzeugs, mit dem symbolische Rechnungen direkt in C++ implementiert werden können. Die wichtigsten Eigenschaften von GiNaC umfassen: eine sehr schnelle komplexe Arithmetik basierend auf der CLN-Bibliothek, effiziente Handhabung selbst großer (mehrere tausend Terme) multivariater Polynome und rationaler Funktionen, symbolisches Differenzieren, Entwicklung von Funktionen in Taylor- und Laurentreihen, Matrizen, Vektoren und Lösen linearer Gleichungssysteme, indextragende Objekte wie z. B. Tensoren, viele vordefinierte mathematische Funktionen (trigonometrische Funktionen, Logarithmen, Fakultät etc.), variable Ausgabemöglichkeiten (z. B. als L^AT_EX-Formel oder als optimierter C++-Code für anschließende numerische Rechnungen), Speichereffizienz durch (für den Benutzer unsichtbare) Referenzzählung von Objekten und leichte Erweiterbarkeit durch eigene symbolische Funktionen und Klassen. Schließlich ist GiNaC kostenlos und im Quelltext verfügbar.

Da GiNaC ursprünglich für Anwendungen in der Hochenergiephysik entwickelt wurde, bietet es außerdem zahlreiche für dieses Aufgabenfeld spezialisierte Objekte und Funktionen an, wie z. B. Clifford- und Farbalgebren und Polylogarithmen.

GiNaC ist zur Entwicklung von umfangreichen Applikationen vorgesehen, bei denen symbolische Rech-