Primzahltestverfahren

Vortrag von
Prof. Hans-Gert Gräbe, Uni Leipzig

am St.-Augustin-Gymnasium Grimma
1. März 2007

Der einfachste Primzahltest - trialDivision

```
primeTestByTrialDivision:=proc(m:Dom::Integer) local z;
begin
  if (m<3) then return(bool(m=2)) end_if;
  z:=2;
  while z*z<=m do
      if m mod z = 0 then return(FALSE) end_if; z:=z+1;
  end_while;
  TRUE;
end:
```

Testbespiel Rep-Units

```
primeTestByTrialDivision(1111);
```

FALSE

Eine Formel für Rep-Units

```
R:=n->(10^n-1)/9;
```

$$n \to \frac{10^n - 1}{9}$$

Untersuchung solcher Zahlen

```
primeTestByTrialDivision(R(2));
primeTestByTrialDivision(R(3));
primeTestByTrialDivision(R(4));
primeTestByTrialDivision(R(5));
```

```
TRUE

FALSE

FALSE

FALSE
```

Ist langweilig, kann der Computer besser in einer Schleife. Noch besser map und Liste verwenden.

```
u:=map([R(n)$n=2..11],primeTestByTrialDivision);
```

```
[TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE]
```

Funktion primeTestByTrialDivision modifizieren, so dass kleinster Faktor ausgegeben wird.
Gleich als Tabelle ausgeben.

```
u:=map([n$n=2..11],n->[n,R(n),primeTestByTrialDivision(R(n))]):
Dom::Matrix()(u);
```

$$\begin{pmatrix} 2 & 11 & \text{TRUE} \\ 3 & 111 & \text{FALSE} \\ 4 & 1111 & \text{FALSE} \\ 5 & 11111 & \text{FALSE} \\ 6 & 111111 & \text{FALSE} \\ 7 & 1111111 & \text{FALSE} \\ 8 & 11111111 & \text{FALSE} \\ 9 & 111111111 & \text{FALSE} \\ 10 & 1111111111 & \text{FALSE} \\ 11 & 11111111111 & \text{FALSE} \end{pmatrix}$$

An diesen Beispielen sehen wir

2|n => R(n) ist durch 11 teilbar

3|n => R(n) ist durch 3 teilbar

```
TestListe:=[(6*n-1,6*n+1)$n=1..5];
```

```
[5, 7, 11, 13, 17, 19, 23, 25, 29, 31]
```

```
n:=5:
[n,R(n),primeTestByTrialDivision(R(n))];
```

```
[5, 11111, FALSE]
```

```
ifactor(R(5));
```

```
ifactor(R(5));
```

$$41 \cdot 271$$

Die Rechnung für R(17) kann nicht zu Ende geführt werden, da R(17) einen sehr großen Faktor hat.

Beobachtung: `primeTestByTrialDivision` funktioniert dann besonders schlecht, wenn n keine kleinen Primfaktoren hat.
Die Funktion erkennt zusammengesetzte Zahlen mit kleinen Faktoren gut.

Zwei Ideen

Aufwand kann um 2/3 verringert werden:

```
moreEfficientPrimeTestByTrialDivision:=proc(m:Dom::Integer) local z;
begin
  if (m<2) then return(FALSE)
    elif (m mod 2 = 0) then return(bool(m=2));
    elif (m mod 3 = 0) then return(bool(m=3))
  end_if;
  z:=5;
  while z*z<=m do
      if m mod z = 0 then return(FALSE) end_if; z:=z+2;
      if m mod z = 0 then return(FALSE) end_if; z:=z+4;
  end_while;
  TRUE;
end;
```

Ansatz kann als Vorsortierung verwendet werden, um uninteressante Zahlen auszuschließen:

```
smallPrimes:=select([i$i=1..100],isprime);
smallPrimesTest:=proc(m:Dom::Integer) local i;
begin
  if (m<2) then return(FALSE) end_if;
  for i in smallPrimes do
       if (m mod i = 0) then return(bool(m=i)) end_if;
  end_for;
  FAIL;
end:
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83$$

Wie groß ist der Prozentsatz an "interessanten" Zahlen?

Wie groß ist der Prozentsatz an "interessanten" Zahlen?

```
n:=10^3:u:=[i$i=2..n]:
nops(select(u,u->smallPrimesTest(u)=FAIL))/nops(u);
float(%);
```

$$\frac{143}{999}$$

$$0.1431431431$$

Aufgabe: Finden Sie eine Näherungsformel für diesen Prozentsatz.

Hinweis: Zeigen Sie, dass die folgende Formel die gesuchte ist.

```
_mult((1-1/p) $ p in smallPrimes);float(%);
```

$$\frac{337785458319471925002240000}{280745566149397514974281352 7}$$

$$0.1203172905$$

Frage: Gibt es ein Verfahren, das die Primzahlen unter den "interessanten" Zahlen einigermaßen zuverlässig erkennt?

Noch einmal Rep-Units

Satz: Ist a ein Teiler von b, so ist auch R(a) ein Teiler von R(b).
Beweis: Teile die b Einsen in Blöcke zu je a Einsen auf.

Eine Rep-Unit R(n) ist also höchstens dann eine Primzahl, wenn n selbst eine Primzahl ist.
In der folgenden Liste sind p=2 und p=3 ausgelassen.

```
notAllSmallPrimes:=select([n$n=5..100],isprime);
```

$$[5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,$$

Allerdings gibt es sehr wenige Rep-Units, die wirklich Primzahlen sind.

```
map(notAllSmallPrimes,p->[p,R(p),isprime(R(p))]):
Dom::Matrix()(%);
```

Mersennesche Zahlen

Als Mersennesche Zahlen bezeichnet man Zahlen der Form 2^p-1.

Als Mersennesche Zahlen bezeichnet man Zahlen der Form 2^p-1.
Die größten heute bekannten Primzahlen sind Mersennesche Zahlen.

```
M:=u->2^u-1;
```

$$u \to 2^u - 1$$

Satz: Ist a ein Teiler von b, so ist auch M(a) ein Teiler von M(b).
Beweis: Die Mersenneschen Zahlen sind die Rep-Units im Zweiersystem.

M(n) ist also höchstens dann eine Primzahl, wenn n selbst eine Primzahl ist.

```
u:=map(smallPrimes,p->[p,isprime(M(p))]):
Dom::Matrix()(u);
```

2	TRUE
3	TRUE
5	TRUE
7	TRUE
11	FALSE
13	TRUE
17	TRUE
19	TRUE
23	FALSE
29	FALSE
31	TRUE
37	FALSE
41	FALSE
43	FALSE
47	FALSE
53	FALSE
59	FALSE
61	TRUE
67	FALSE
71	FALSE
73	FALSE
79	FALSE
83	FALSE
89	TRUE
97	FALSE

Rechnen mit Resten

Welchen Rest lässt die Rep-Unit mit 17 Stellen bei der Division durch 41?

```
R(17) mod 41
```

11

```
(R(17)-11)/41
```

271002710027100

Und die Fermatzahl F(30) = 2^(2^30) + 1 mit 2^{30} Stellen im Zweiersystem?

Und die Fermatzahl F(30) = 2^(2^30) + 1 mit 2^{30} Stellen im Zweiersystem?
Der Computer kann das nicht direkt ausrechnen, die Zahlen werden zu groß.

```
2^(2^30) mod 41;
Error: Overflow/underflow in arithmetical operation
```

Kann man den Rest bestimmen, ohne die Zahl selbst auszurechnen?
Gibt es Gesetzmäßigkeiten für Potenzreste?

```
(2^n mod 41)$n=1..30
 2, 4, 8, 16, 32, 23, 5, 10, 20, 40, 39, 37, 33, 25, 9, 18, 36, 31, 21, 1, 2, 4, 8, 16, 3
```

Wir sehen, dass $2^{20} = 1 \pmod{41}$ gilt, also auch $2^{20 \cdot k} = 1 \pmod{41}$.

```
2^30 mod 20
 4
```

Es gilt $2^{30} = 4 \pmod{20}$, also $2^{30} = 20 \cdot k + 4$ und folglich $F(30) = 2^{20 \cdot k+4} = 2^4 = 16 \pmod{41}$.
Rechnet man also mit Resten und nicht mit den Zahlen selbst, so lässt sich das Ergebnis schnell bestimmen.
MuPAD kann in solchen Restklassenbereichen rechnen:

```
Z:=Dom::IntegerMod(41):
Z(2)^(2^30);
 16 mod 41
```

Exkurs Rechnen mit Resten, Kürzungsregel und prime Restklassen.

Kürzungsregel als Eigenschaft der
Multiplikationsabbildung m_a: u -> a*u (mod n)

```
n:=12;
primeReste:=select([u$u=1..(n-1)],u->igcd(u,n)=1);
map(primeReste,u->[u,5*u mod n]): Dom::Matrix()(%);
 12
 [1, 5, 7, 11]
```

$$\begin{pmatrix} 1 & 5 \\ 5 & 1 \\ 7 & 11 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 5 \\ 5 & 1 \\ 7 & 11 \\ 11 & 7 \end{pmatrix}$$

Satz: Ist m: Z -> Z eine Abbildung einer endlichen Menge in sich,
so sind die folgenden drei Eigenschaften von m äquivalent:

(1) m ist injektiv, d.h. jedes Element hat höchstens ein Urbild.
(2) m ist surjektiv, d.h. jedes Element hat mindestens ein Urbild.
(3) m ist bijektiv, d.h. jedes Element hat genau ein Urbild.

Ist a eine prime Restklasse (mod n), so erfüllt die Multiplikationsabbildung m_a diese äquivalenten Eigenschaften. Insbesondere existiert genau eine (prime) Restklasse a' mit der Eigenschaft a*a'=1 (mod n).

a' heißt die **zu a inverse Restklasse** und kann leicht über den Euklidschen Algorithmus bestimmt werden. Die Menge Z_n^* der primen Restklassen (mod n) ist damit eine **Gruppe**, d.h. eine bzgl. Multiplikation und Inversenbildung abgeschlossene Struktur.

ϕ(n) bezeichnet die Anzahl der Elemente dieser Menge (**Eulersche Phi-Funktion**). Insbesondere gilt $\phi(p)=p-1$ und $\phi(p^a)=p^a-p^{a-1}$, wenn p eine Primzahl ist.

Ordnung ord(a) eines Elements a in einer Gruppe G.

(1) $a^k=1$ => k ist Vielfaches von ord(a)
(2) $n=|G|$ => $a^n=1$

Folgerungen aus diesen ganz allgemeinen Überlegungen:

Satz von Euler: Ist a (mod n) eine prime Restklasse, so gilt $a^{\phi(n)} = 1$ (mod n).

Kleiner Satz von Fermat: Ist p eine Primzahl und 1<a<p, so gilt $a^{p-1} = 1$ (mod p).

Der Fermat-Test

Kontraposition des kleinen Satzes von Fermat:

Gilt $a^{m-1} <> 1$ (mod m) für ein a mit 1<a<m,
so ist m garantiert zusammengesetzt.

Darauf kann der folgende einfache Fermat-Test aufgebaut werden.

```
singleFermatTest:=proc(m:Dom::Integer, a:Dom::Integer) local D;
begin
D:=Dom::IntegerMod(m);
iszero(D(a)^(m-1)-1);
end;
```

```
end;
```

```
proc singleFermatTest(m, a) ... end
```

Kommt als Ergebnis FALSE heraus, dann ist die Zahl m *garantiert zusammengesetzt* und das kann einfach nachgeprüft werden: "Mach doch mal den Fermat-Test mit der Basis a und dann siehst du es ja schon".

a heißt deshalb **Fermat-Zeuge** dafür, dass m zusammengesetzt ist.

Mit a=2 können wir schon eine ganze Menge über Rep-Units aussagen.

```
TestListe:=[(6*n-1,6*n+1)$n=1..5];
map(TestListe,u->[u,R(u),singleFermatTest(R(u),2)]):
Dom::Matrix()(%);
```

```
[5, 7, 11, 13, 17, 19, 23, 25, 29, 31]
```

$$\begin{pmatrix} 5 & 11111 & \text{FALSE} \\ 7 & 1111111 & \text{FALSE} \\ 11 & 11111111111 & \text{FALSE} \\ 13 & 1111111111111 & \text{FALSE} \\ 17 & 11111111111111111 & \text{FALSE} \\ 19 & 1111111111111111111 & \text{TRUE} \\ 23 & 11111111111111111111111 & \text{TRUE} \\ 25 & 1111111111111111111111111 & \text{FALSE} \\ 29 & 11111111111111111111111111111 & \text{FALSE} \\ 31 & 1111111111111111111111111111111 & \text{FALSE} \end{pmatrix}$$

```
select(smallPrimes,u->singleFermatTest(R(u),2));
```

```
[19, 23]
```

Alle bis auf die hier ausgewürfelten sind also garantiert zusammengesetzt.
Weitere bekannte prime Rep-Units sind R(317) und R(1031).

```
singleFermatTest(R(317),3)
```

```
TRUE
```

Andere prime Rep-Units R(p) kann es nur für prime $p>10^5$ geben, aber man hat noch keine gefunden.

Primzahlzertifikate

Um genau zu **beweisen**, dass die hier gefundenen Rep-Units wirklich Primzahlen sind, kann man eine prime Restklasse a (mod m) suchen, für die m-1 die genaue Ordnung in der Gruppe der primen Restklassen ist. Für Primzahlen m weiß man, dass ein solches Element immer existiert.

ein solches Element immer existiert.

Da die Ordnung ein Teiler von m-1 ist, muss also nur geprüft werden, dass für keinen Primteiler p | m-1 bereits $a^{(m-1)/p} = 1 \pmod m$ eintreten kann.

```
m:=R(23);Z:=Dom::IntegerMod(m):
pdList:=numlib::primedivisors(m-1);
map(pdList,u->expr(Z(43)^((m-1)/u)))
```

Man kann beweisen, dass es ausreicht, für jeden Faktor p ein a_p mit

$$a_p^{(m-1)/p} <> 1 \pmod m$$

zu finden. Die Liste der Paare (p,a_p) bezeichnet man als **Primzahlzertifikat** für m, da sich leicht nachrechnen lässt, dass

(1) die angegebenen Primzahlen die Primteiler von m sind und
(2) die angegebenen a_p die Bedingung $a_p^{(m-1)/p} <> 1 \pmod m$ erfüllen.

Fermat-Test und Mersenne-Zahlen

Für die Mersennezahlen hilft a=2 in keinem Fall weiter.

```
map(notAllSmallPrimes,u->[u,M(u),singleFermatTest(M(u),2)]):
Dom::Matrix()(%);
```

5	31	TRUE
7	127	TRUE
11	2047	TRUE
13	8191	TRUE
17	131071	TRUE
19	524287	TRUE
23	8388607	TRUE
29	536870911	TRUE
31	2147483647	TRUE
37	137438953471	TRUE
41	2199023255551	TRUE
43	8796093022207	TRUE
47	140737488355327	TRUE
53	9007199254740991	TRUE
59	576460752303423487	TRUE
61	2305843009213693951	TRUE
67	147573952589676412927	TRUE
71	2361183241434822606847	TRUE
73	9444732965739290427391	TRUE
79	604462909807314587353087	TRUE
83	967140655691703339764940 7	TRUE
89	618970019642690137449562111	TRUE
97	158456325028528675187087900671	TRUE

Aufgabe: Beweisen Sie, dass a=2 nicht als Fermatzeuge für M(p) mit primem p taugt. Hinweis: Zeigen Sie, dass $2^p = 1 \pmod{2^p-1}$ gilt, 2^p-2 durch p teilbar ist und folglich immer $2^{M(p)-1} = 1 \pmod{M(p)}$ gilt.

a=3 dagegen entdeckt alle zusammengesetzten Mersennezahlen im untersuchten

a=3 dagegen entdeckt alle zusammengesetzten Mersennezahlen im untersuchten Bereich.

```
map(notAllSmallPrimes,u->[u,M(u),singleFermatTest(M(u),3),isprime(M(
Dom::Matrix()(%);
```

5	31	TRUE	TRUE
7	127	TRUE	TRUE
11	2047	FALSE	FALSE
13	8191	TRUE	TRUE
17	131071	TRUE	TRUE
19	524287	TRUE	TRUE
23	8388607	FALSE	FALSE
29	536870911	FALSE	FALSE
31	2147483647	TRUE	TRUE
37	137438953471	FALSE	FALSE
41	2199023255551	FALSE	FALSE
43	8796093022207	FALSE	FALSE
47	140737488355327	FALSE	FALSE
53	9007199254740991	FALSE	FALSE
59	576460752303423487	FALSE	FALSE
61	2305843009213693951	TRUE	TRUE
67	147573952589676412927	FALSE	FALSE
71	2361183241434822606847	FALSE	FALSE
73	9444732965739290427391	FALSE	FALSE
79	604462909807314587353087	FALSE	FALSE
83	967140655691703339764940 7	FALSE	FALSE
89	618970019642690137449562111	TRUE	TRUE
97	158456325028528675187087900671	FALSE	FALSE

Wir können den Fermat-Test mit verschiedenen zufällig gewählten Basen a_1, a_2, ..., a_k ausführen und bekommen folgendes Ergebnis:

(1) Liefert der Fermat-Test für *eine* der Basen FALSE,
so ist m *garantiert* zusammengesetzt.

(2) Liefert der Fermat-Test für *alle* Basen TRUE,
so ist m *hochgradig verdächtig*, eine Primzahl zu sein.
Genau wissen wir es aber nicht.

Einen solchen Test bezeichnet man als Las-Vegas-Variante des Fermat-Tests

```
LasVegasFermatTest:=proc(m:Dom::Integer, k:Dom::Integer)
  local D,i,a,r;
begin
  D:=Dom::IntegerMod(m);
  r:=random(m); /* r ist Zufallszahlengenerator */
  for i from 1 to k do
      a:=r();
      if igcd(a,m) <> 1 then return(FALSE) end_if;
         /* a ist keine prime Restklasse, m also zusammengesetzt */
      if not iszero(D(a)^(m-1)-1) then return(FALSE) end_if;
         /* der FermatTest schlägt zu */
  end_for;
  TRUE;
end:
```

```
end:
```

Wieviele Fermatzeugen gibt es?

Antwort: Wenn es einen Zeugen gibt, dann wenigstens 50% der Reste.

Damit ist die Wahrscheinlichkeit, bei k Versuchen keinen zu erwischen, gleich 2^{-k}.

Carmichael-Zahlen

Primzahlen können keine Fermat-Zeugen haben. Gibt es zusammengesetzte Zahlen ohne Fermat-Zeugen? Die könnte der Fermat-Test nicht von Primzahlen unterscheiden.

```
m:=561;
ifactor(m);
Z:=Dom::IntegerMod(m):
primeReste:=select([u$u=1..(m-1)],u->igcd(u,m)=1):
map({op(primeReste)},u->Z(u)^(m-1));
```

$$561$$

$$3 \cdot 11 \cdot 17$$

$$\{1 \bmod 561\}$$

Um dieses Phänomen besser zu verstehen, wollen wir zunächst den Satz von Euler für zusammengesetzte Zahlen verschärfen.

Ist $m=m_1 \cdot m_2 \cdot \ldots \cdot m_k$ eine Zerlegung in paarweise teilerfremde Faktoren und $a^n=1 \pmod{m_i}$ für $i=1,\ldots,k$, so gilt auch $a^n=1 \pmod{m}$.

Satz von Carmichael:
Ist $m=p_1^{a1} \cdot p_2^{a2} \cdot \ldots \cdot p_k^{ak}$ die Primfaktorzerlegung und $\psi(m)=kgV(\phi(p_1^{a1}),\ldots,\phi(p_k^{ak}))$, so gilt für eine prime Restklasse a (mod m) bereits $a^{\psi(m)}=1 \pmod{m}$.

Für die Eulersche Phi-Funktion selbst gilt $\phi(m)=\phi(p_1^{a1}) \cdot \ldots \cdot \phi(p_k^{ak})$, was hier nicht bewiesen werden soll.

Für $m=3 \cdot 11 \cdot 17=561$ gilt

$$\phi(m)=2 \cdot 10 \cdot 16=320,$$

dagegen

$$\psi(m)=kgV(2,10,16)=80$$

und mit $a^{80}=1 \pmod{m}$ wegen $m-1=360$ nach dem Satz von Carmichael auch

und mit $a^{80}=1 \pmod m$ wegen m-1=360 nach dem Satz von Carmichael auch

$$a^{m-1}=1 \pmod m$$

Eine zusammengesetzte Zahl m mit $\psi(m) \mid m-1$ bezeichnet man als **Carmichael-Zahl**. Diese kann der Fermat-Test prinzipiell nicht von einer Primzahl unterscheiden.

Aufgabe: Zeigen Sie, dass alle Zahlen, die aus drei Primfaktoren (6k+1)(12k+1)(18k+1) bestehen, Carmichaelzahlen sind.

```
kList:=select([k$k=1..200],k->isprime(6*k+1)
                and isprime(12*k+1) and isprime(18*k+1));
```

```
[1, 6, 35, 45, 51, 55, 56, 100, 121, 195]
```

```
CarmichaelZahlen:=map(kList,k->(6*k+1)*(12*k+1)*(18*k+1))
```

```
[1729, 294409, 56052361, 118901521, 172947529, 216821881, 228842209, 1299636
```

Warum entdeckt der `LasVegasFermatTest` manche der Zahlen trotzdem manchmal als zusammengesetzt?

```
map(CarmichaelZahlen,u->LasVegasFermatTest(u,5))
```

```
[FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE]
```

Der Rabin-Miller-Test

Wie kann man $a^{m-1} \pmod m$ schnell berechnen?

Methode des binären Potenzierens.

m ist ungerade, also $m-1=q\cdot 2^t$ für ein ungerades q und t>0. $a^{m-1} \pmod m$ wird von $b=a^q \pmod m$ durch fortgesetztes Quadrieren (mod m) berechnet. Diskussion der möglichen Alternativen.

Das führt zu folgender Verfeinerung des Fermat-Tests:

```
primeTestRabinMiller:=proc(m:Dom::Integer,c:Dom::Integer)
  local a,q,b,t,i,j,r,D;
begin
  D:=Dom::IntegerMod(m);
  q:=m-1; t:=0; r:=random(m);

  while q mod 2 = 0 do q:=q/2; t:=t+1 end_while;
```

```
  while q mod 2 = 0 do q:=q/2; t:=t+1 end_while;
        /* nun ist  m-1 = 2^t * q */
  for i from 1 to c do
    a:=r(); b:=D(a)^q;
    if iszero(b-1) or iszero(b+1) then next end_if;
    /* keine Information, wenn b = 1 oder b = -1 (mod m) */
    for j from 1 to (t-1) do
      /* nun ist  b <> 1 oder -1 (m) */
      b:=b*b;
        if iszero(b-1) then return(FALSE) end_if;
        if iszero(b+1) then break end_if;   /* keine Information */
    end_for;
    if iszero(b+1) then next;         /* keine Information */
    else return(FALSE) end_if;
  end_for;
  TRUE;
end_proc:
```

Nun werden auch die Carmichaelzahlen sicher erkannt.

```
map(CarmichaelZahlen,u->primeTestRabinMiller(u,2))
```

[FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE]

Mersennesche Primzahlen

Mersenne behauptete im Jahr 1644, dass M(q) für
q in (2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257)
prim sei und für keinen anderen Exponenten q < 257.

Mit MuPAD kann diese Behauptung heute leicht überprüft werden:

```
select([$1..1000],x->isprime(x) and isprime (2^x-1));
```

[2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607]

Mersennes Liste enthält also 4 Fehler, wobei q=67 gewöhnlich großzügig als Schreibfehler gewertet wird.

Im Gegensatz zu anderen Zahlenfolgen wie den Rep-Units, Fermat-Zahlen oder Euler-Zahlen findet man unter den Mersenne-Zahlen bei größeren Indizes immer mal wieder eine Primzahl. Deshalb sind die größten heute "namentlich" bekannten Primzahlen allesamt Mersenne-Zahlen.

Die ersten 14 Mersenneschen Primzahlen sind heute leicht zu finden. Die Zahlen

Die ersten 14 Mersenneschen Primzahlen sind heute leicht zu finden. Die Zahlen werden allerdings schnell größer. Die 14. Mersennesche Primzahl M(607) etwa hat schon 183 Dezimalstellen.

```
M(607);
length(%);
```

Die Mersenneschen Primzahlen Nummer 13 und 14 wurden erst 1952 von R.M.Robinson entdeckt, der auch die Nummer 15 bis 17 fand:

M(1.279) mit 386 Stellen,
M(2.203) mit 664 Stellen und
M(2.281) mit 687 Stellen.

Weitere Mersennesche Primzahlen wurden in den 80er Jahren vor allem von D.Slowinski und seinen Mitstreitern gefunden, so etwa die

30. Mersennesche Primzahl M(132.049) mit 39.751 Stellen.

Allerdings wurde diese Zahl erst später als Nummer 30 identifiziert, denn man hatte die Primzahl M(110.503) übersehen.

In den 90er Jahren wurde, mit der zunehmenden Leistunsgfähigkeit der Rechentechnik und insbesondere der Möglichkeit, stabile verteilte Rechnungen in lose gekoppelten Netzwerken zu organisieren, eine ganze Reihe neuer Mersennescher Primzahlen entdeckt. Während Ribenboim in der 3. Auflage (1996) seines Buchs "The new book of prime number records" noch die

33. Mersennesche Primzahl M(859.433) mit 258.716 Stellen

als größte bekannte Primzahl nennt, wurde am 12. Dezember 2003

M(20.996.011) mit 6.320.430 Stellen

als 40. Mersennesche Primzahl gefunden, was sich bis in die Tagespresse herumsprach.

Am 4. September 2006 wurde mit M(32.582.657) die 44. Mersennesche Primzahl gefunden, die mit 9.808.358 Stellen fast an die Grenze von 10 Millionen Stellen heranreicht, für deren Überschreiten die EFF (Electronic Frontier Foundation) einen Preis von 100.000 $ ausgelobt hat.

Allerdings ist die Nummerierung bei den neueren Zahlen noch nicht endgültig, da noch nicht alle Zwischenkandidaten als Nichtprimzahlen ausgeschlossen wurden.

Es bedarf deshalb einer ausgefeilten Arithmetik und guter Algorithmen, um mit solchen Riesenzahlen zu rechnen, so dass die Jagd nach neuen großen Primzahlen auch zum Test neuer Hard- und Software verwendet wird.

Die letzten Mersenneschen Primzahlen wurden allen im Rahmen eines großen Projekts zum verteilten Rechnen, dem GIMPS-Projekt (Great Internet Mersenne Primes Search, siehe http://www.mersenne.org) gefunden. Mehr dazu auch auf den Webseiten von Chris Caldwell unter http://www.utm.edu/research/primes.