Számítógépes számelmélet

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

▶ 1. A prímek eloszlása, szitálás

▶ 2. Egyszerű faktorizálási módszerek

▶ 3. Egyszerű prímtesztelési módszerek

▶ 4. Lucas–sorozatok

▶ 5. Alkalmazások

▶ 6. Számok és polinomok

▶ 7. Gyors Fourier–transzformáció

▶ 8. Elliptikus függvények

▶ 9. Számolás elliptikus görbéken

▶ 10. Faktorizálás elliptikus görbékkel

▼ 11. Prímteszt elliptikus görbékkel

```
> restart; with(numtheory);
```

[*GIgcd*, *bigomega*, *cfrac*, *cfracpol*, *cyclotomic*, *divisors*, *factorEQ*, *factorset*, *fermat*, *imagunit*, *index*, *integral_basis*, *invcfrac*, *invphi*, *issqrfree*, *jacobi*, *kronecker*, λ, *legendre*, *mcombine*, *mersenne*, *migcdex*, *minkowski*, *mipolys*, *mlog*, *mobius*, *mroot*, *msqrt*, *nearestp*, *nthconver*, *nthdenom*, *nthnumer*, *nthpow*, *order*, *pdexpand*, ϕ, π, *pprimroot*, *primroot*, *quadres*, *rootsunity*, *safeprime*, σ, *sq2factor*, *sum2sqr*, τ, *thue*] (11.1)

11.1. Tétel.

```
> #
  # This routine randomly choose an elliptic "curve" modulo n,
  # where gcd(n,6)=1. The coordinates x,y are choosen
  # randomly, the parameter a too, and b is calculated.
  # The list [x,y,a,b] is given back or a divisor d of n.
  #

  ellrand:=proc(n) local x,y,a,b,r,d,f;
  r:=rand(n); d:=n;
  while d=n do
    x:=r(n); y:=r(n); a:=r(n); b:=y^2-x^3-a*x mod n;
    d:=4*a^3+27*b^2 mod n; gcd(d,n);
  od;
  if %<n and %>1 then return % fi;
  [x,y,a,b]; end;
```

$$ellrand := \mathbf{proc}(n) \tag{11.1.1}$$

$$\begin{aligned}
&\mathbf{local}\ x, y, a, b, r, d, f;\\
&r := rand(n);\\
&d := n;\\
&\mathbf{while}\ d = n\ \mathbf{do}\\
&\quad x := r(n);\\
&\quad y := r(n);\\
&\quad a := r(n);\\
&\quad b := mod(y\hat{}2 - x\hat{}3 - a*x,\ n);\\
&\quad d := mod(4*a\hat{}3 + 27*b\hat{}2,\ n);\\
&\quad gcd(d,\ n)\\
&\mathbf{end\ do};\\
&\mathbf{if}\ \% < n\ \mathbf{and}\ 1 < \%\ \mathbf{then}\\
&\quad \mathbf{return}\ \%\\
&\mathbf{end\ if};\\
&[x, y, a, b]\\
&\mathbf{end\ proc}
\end{aligned}$$

```
> #
  # Addition on an elliptic "curve" modulo n, where gcd(n,6)=1.
  # P and Q are the points to add and a,b are the parameters.
  # The return value is the sum of the points or a divisor d of
  n.
```

```
   #

   elladd:=proc(P,Q,a,b,n) local l,d;
   if P[3]=0 then return Q fi;
   if Q[3]=0 then return P fi;
   if P=Q then return elldou(P,a,b,n) fi;
   if P[1]=Q[1] then return [0,1,0] fi;
   d:=igcdex(P[1]-Q[1],n,'l');
   if 1<d and d<n then return d fi;
   l:=(P[2]-Q[2])*l mod n;
   l^2-P[1]-Q[1] mod n;
   [%,l*(P[1]-%)-P[2] mod n,1];
   end;
```

$$elladd := \mathbf{proc}(P, Q, a, b, n) \tag{11.1.2}$$

local l, d;
if $P[3] = 0$ **then**
return Q
end if;
if $Q[3] = 0$ **then**
return P
end if;
if $P = Q$ **then**
return $elldou(P, a, b, n)$
end if;
if $P[1] = Q[1]$ **then**
return $[0, 1, 0]$
end if;
$d := igcdex(P[1] - Q[1], n, 'l')$;
if $1 < d$ **and** $d < n$ **then**
return d
end if;
$l := mod((P[2] - Q[2]) * l, n)$;
$mod(l\hat{}2 - P[1] - Q[1], n)$;
$[\%, mod(l * (P[1] - \%) - P[2], n), 1]$
end proc

```
> #
   # Doubling on an elliptic "curve" modulo n, where gcd(n,6)=1.
```

```
   # P is the point to double and a, b are the parameters.
   # The return value is the double of the point P or
   # a divisor d of n.
   #

   elldou:=proc(P,a,b,n) local l,d;
   if P[3]=0 then return P fi;
   if P[2]=0 then return [0,1,0] fi;
   d:=igcdex(2*P[2],n,'l');
   if 1<d and d<n then return d fi;
   l:=(3*P[1]^2+a)*l mod n;
   l^2-2*P[1] mod n;
   [%,l*(P[1]-%)-P[2] mod n,1];
   end;
```

$$elldou := \mathbf{proc}(P, a, b, n) \tag{11.1.3}$$

$$\mathbf{local}\ l, d;$$

$$\mathbf{if}\ P[3] = 0\ \mathbf{then}$$

$$\mathbf{return}\ P$$

$$\mathbf{end\ if};$$

$$\mathbf{if}\ P[2] = 0\ \mathbf{then}$$

$$\mathbf{return}\ [0, 1, 0]$$

$$\mathbf{end\ if};$$

$$d := igcdex(2 * P[2], n, 'l');$$

$$\mathbf{if}\ 1 < d\ \mathbf{and}\ d < n\ \mathbf{then}$$

$$\mathbf{return}\ d$$

$$\mathbf{end\ if};$$

$$l := mod((3 * P[1]\hat{}2 + a) * l, n);$$

$$mod(l\hat{}2 - 2 * P[1], n);$$

$$[\%, mod(l * (P[1] - \%) - P[2], n), 1]$$

$$\mathbf{end\ proc}$$

```
> #
   # This program compute k*P, k>=0 or a divisor of n
   # on an elliptic "curve" modulo n, where gcd(n,6)=1.
   # It use the left-to-right binary method.
   #

   ellmul:=proc(P,k,a,b,n) local L,i,Q;
   if k=0 then return [0,1,0] fi;
   if P[3]=0 then return P fi;
   L:=convert(k,base,2);
```

```
  Q:=P;
  for i from nops(L)-1 to 1 by -1 do
    Q:=elldou(Q,a,b,n);
    if type(Q,integer) then return Q fi;
    if L[i]=1 then
      Q:=elladd(P,Q,a,b,n);
      if type(Q,integer) then return Q fi;
    fi;
  od; Q; end;
```

$$ellmul := \mathbf{proc}(P, k, a, b, n) \tag{11.1.4}$$

```
local L, i, Q;
if k = 0 then
    return [0, 1, 0]
end if;
if P[3] = 0 then
    return P
end if;
L := convert(k, base, 2);
Q := P;
for i from nops(L) - 1 by -1 to 1 do
    Q := elldou(Q, a, b, n);
    if type(Q, integer) then
        return Q
    end if;
    if L[i] = 1 then
        Q := elladd(P, Q, a, b, n);
        if type(Q, integer) then
            return Q
        end if
    end if
end do;
Q
end proc
```

```
> #
  # This brute force procedure calculate the number of points
  # on an elliptic curve modulo p>3, a prime. The curve
```

```
   # parameters are a, b.
   #

   ellcount:=proc(a,b,p) local x,c;
   c:=1;
   for x from 0 to p-1 do c:=c+numtheory[jacobi](x^3+a*x+b,p)+1;
   od;
   c; end;
```

$$ellcount := \mathbf{proc}(a, b, p) \tag{11.1.5}$$

$$\mathbf{local}\ x, c;$$

$$c := 1;$$

$$\mathbf{for}\ x\ \mathbf{from}\ 0\ \mathbf{to}\ p-1\ \mathbf{do}$$

$$c := c + numtheory[numtheory:-jacobi](x\text{^}3 + a*x + b, p) + 1$$

$$\mathbf{end\ do};$$

$$c$$

$$\mathbf{end\ proc}$$

```
> n:=nextprime(1008); E:=ellrand(n); m:=ellcount(E[3],E[4],n);
```

$$n := 1009$$

$$E := [453, 76, 621, 422]$$

$$m := 1047 \tag{11.1.6}$$

```
> n:=1009; E := [889, 300, 991, 649]; m := 974;
```

$$n := 1009$$

$$E := [889, 300, 991, 649]$$

$$m := 974 \tag{11.1.7}$$

```
> ifactor(974); ellmul([E[1],E[2],1],m,E[3],E[4],n);
  elldou([E[1],E[2],1],E[3],E[4],n);
```

$$(2)\,(487)$$

$$[0, 1, 0]$$

$$[758, 824, 1] \tag{11.1.8}$$

11.2. Tétel.

```
> #
  # This brute force procedure calculate the high of
  # an elliptic curve modulo p>3, a prime. The curve
  # parameters are a, b.
  #

  ellhigh:=proc(a,b,p) local x,y,h,i,P;
```

```
  h:=1;
  for x from 0 to p-1 do
    y:=msqrt(x^3+a*x+b,p);
    if y=FAIL then next fi;
    P:=[x,y,1];
    for i from 2 do
      i;
      P:=elladd(P,[x,y,1],a,b,p);
      if P=[0,1,0] then
        if i>h then h:=i fi; break;
      fi;
    od; if 3*h>2*p then break fi;
  od; h; end;
```

$$ellhigh := \mathbf{proc}(a, b, p) \tag{11.2.1}$$

$\mathbf{local}\ x, y, h, i, P;$

$h := 1;$

$\mathbf{for}\ x\ \mathbf{from}\ 0\ \mathbf{to}\ p - 1\ \mathbf{do}$

$y := numtheory:-msqrt(x\hat{}3 + a*x + b, p);$

$\mathbf{if}\ y = FAIL\ \mathbf{then}$

$\mathbf{next}$

$\mathbf{end\ if};$

$P := [x, y, 1];$

$\mathbf{for}\ i\ \mathbf{from}\ 2\ \mathbf{do}$

$i;$

$P := elladd(P, [x, y, 1], a, b, p);$

$\mathbf{if}\ P = ([0, 1, 0])\ \mathbf{then}$

$\mathbf{if}\ h < i\ \mathbf{then}$

$h := i$

$\mathbf{end\ if};$

$\mathbf{break}$

$\mathbf{end\ if}$

$\mathbf{end\ do};$

$\mathbf{if}\ 2*p < 3*h\ \mathbf{then}$

$\mathbf{break}$

$\mathbf{end\ if}$

$\mathbf{end\ do};$

h

end proc

```
> ellrand(97);
```

$$[58, 15, 0, 83] \tag{11.2.2}$$

```
> ellhigh(5,57,97); ellhigh(95,78,97);
  ellhigh(33,96,97); ellhigh(55,51,97);
  ellhigh(24,13,97);ellhigh(59,39,97);
  ellhigh(49,54,97);ellhigh(45,68,97);
  ellhigh(76,30,97);ellhigh(0,83,97);
```

$$103$$
$$80$$
$$116$$
$$44$$
$$86$$
$$88$$
$$110$$
$$98$$
$$102$$
$$103 \tag{11.2.3}$$

11.3. A prímtesztek vázlata.

```
> n:=1009; E:=ellrand(n); m:=ellcount(E[3],E[4],n);
```

$$n := 1009$$
$$E := [38, 603, 582, 65]$$
$$m := 960 \tag{11.3.1}$$

```
> E:=[889,300,991,649]; m:=974;
```

$$E := [889, 300, 991, 649]$$
$$m := 974 \tag{11.3.2}$$

```
> ifactor(m); ellmul([E[1],E[2],1],m,E[3],E[4],n);
  elldou([E[1],E[2],1],E[3],E[4],n);
```

$$(2)\,(487)$$
$$[0, 1, 0]$$
$$[758, 824, 1] \tag{11.3.3}$$

```
> n1:=487; E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$n1 := 487$$
$$E1 := [154, 481, 71, 52]$$

(11.3.4)

$$m1 := 465 \tag{11.3.4}$$

```
> E1:=[201,473,457,19]; m1:=530;
```

$$E1 := [201, 473, 457, 19]$$

$$m1 := 530 \tag{11.3.5}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  ellmul([E1[1],E1[2],1],10,E1[3],E1[4],n1);
```

$$(2)\,(5)\,(53)$$

$$[0, 1, 0]$$

$$[400, 454, 1] \tag{11.3.6}$$

```
> n2:=53;
```

$$n2 := 53 \tag{11.3.7}$$

11.4. A Goldwasser–Kilian–teszt.

```
> n:=1009; E:=ellrand(n); m:=ellcount(E[3],E[4],n);
```

$$n := 1009$$

$$E := [627, 750, 129, 78]$$

$$m := 1012 \tag{11.4.1}$$

```
> E:=[889,300,991,649]; m:=974;
```

$$E := [889, 300, 991, 649]$$

$$m := 974 \tag{11.4.2}$$

```
> ifactor(m); ellmul([E[1],E[2],1],m,E[3],E[4],n);
  elldou([E[1],E[2],1],E[3],E[4],n);
```

$$(2)\,(487)$$

$$[0, 1, 0]$$

$$[758, 824, 1] \tag{11.4.3}$$

```
> n1:=487; E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$n1 := 487$$

$$E1 := [72, 340, 296, 91]$$

$$m1 := 520 \tag{11.4.4}$$

```
> E1:=[201,473,457,19]; m1:=530;
```

$$E1 := [201, 473, 457, 19]$$

$$m1 := 530 \tag{11.4.5}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  elldou([E1[1],E1[2],1],E1[3],E1[4],n1);
```

$$(2)\,(5)\,(53)$$

$$[0, 1, 0]$$

$$[68, 198, 1] \tag{11.4.6}$$

```
> n2:=265; E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$n2 := 265$$

$$E2 := [230, 259, 171, 136]$$

$$m2 := 266 \tag{11.4.7}$$

```
> E2:=[230,259,171,136]; m2:=266;
```

$$E2 := [230, 259, 171, 136]$$

$$m2 := 266 \tag{11.4.8}$$

```
> ifactor(m2); ellmul([E2[1],E2[2],1],m2,E2[3],E2[4],n2);
  elldou([E2[1],E2[2],1],E2[3],E2[4],n2);
```

$$(2)\,(7)\,(19)$$

$$[208, 89, 1]$$

$$[169, 198, 1] \tag{11.4.9}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [268, 146, 265, 300]$$

$$m1 := 451 \tag{11.4.10}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [142, 191, 8, 66]$$

$$m1 := 472 \tag{11.4.11}$$

```
> E1:=[142,191,8,66]; m1:=472;
```

$$E1 := [142, 191, 8, 66]$$

$$m1 := 472 \tag{11.4.12}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  elldou([E1[1],E1[2],1],E1[3],E1[4],n1);
```

$$(2)^3\,(59)$$

$$[0, 1, 0]$$

$$[176, 105, 1] \tag{11.4.13}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [249, 267, 252, 400]$$

$$m1 := 475 \tag{11.4.14}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [307, 255, 408, 332]$$

$$m1 := 522 \tag{11.4.15}$$

```
> E1 := [307, 255, 408, 332]; m1 := 522;
```

$$E1 := [307, 255, 408, 332]$$

$$m1 := 522 \tag{11.4.16}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  elldou([E1[1],E1[2],1],E1[3],E1[4],n1);
```

$$(2)\,(3)^2\,(29)$$

$$[0, 1, 0]$$

$$[161, 49, 1] \tag{11.4.17}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [246, 327, 360, 30]$$

$$m1 := 468 \tag{11.4.18}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [112, 89, 273, 301]$$

$$m1 := 520 \tag{11.4.19}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [170, 371, 182, 391]$$

$$m1 := 506 \tag{11.4.20}$$

```
> E1 := [170, 371, 182, 391]; m1 := 506;
```

$$E1 := [170, 371, 182, 391]$$

$$m1 := 506 \tag{11.4.21}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  elldou([E1[1],E1[2],1],E1[3],E1[4],n1);
```

$$(2)\,(11)\,(23)$$

$$[0, 1, 0]$$

$$[72, 50, 1] \tag{11.4.22}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [108, 39, 16, 437]$$

$$m1 := 474 \tag{11.4.23}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [453, 307, 336, 338]$$

$$m1 := 465 \tag{11.4.24}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [214, 417, 212, 484]$$

$$m1 := 528 \tag{11.4.25}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [136, 360, 195, 222]$$

(11.4.26)

$$m1 := 492 \tag{11.4.26}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [338, 124, 446, 259]$$

$$m1 := 469 \tag{11.4.27}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [355, 453, 291, 475]$$

$$m1 := 525 \tag{11.4.28}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [163, 265, 295, 369]$$

$$m1 := 518 \tag{11.4.29}$$

```
> E1 := [163, 265, 295, 369]; m1 := 518;
```

$$E1 := [163, 265, 295, 369]$$

$$m1 := 518 \tag{11.4.30}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  elldou([E1[1],E1[2],1],E1[3],E1[4],n1);
```

$$(2)\,(7)\,(37)$$

$$[0, 1, 0]$$

$$[383, 148, 1] \tag{11.4.31}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [321, 86, 143, 358]$$

$$m1 := 480 \tag{11.4.32}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [353, 426, 308, 26]$$

$$m1 := 468 \tag{11.4.33}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [60, 46, 283, 461]$$

$$m1 := 520 \tag{11.4.34}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [458, 2, 388, 94]$$

$$m1 := 503 \tag{11.4.35}$$

```
> E1:=ellrand(n1); m1:=ellcount(E1[3],E1[4],n1);
```

$$E1 := [365, 230, 106, 399]$$

$$m1 := 502 \tag{11.4.36}$$

```
> E1 := [365, 230, 106, 399]; m1 := 502;
```

$$E1 := [365, 230, 106, 399]$$

(11.4.37)

$$m1 := 502 \tag{11.4.37}$$

```
> ifactor(m1); ellmul([E1[1],E1[2],1],m1,E1[3],E1[4],n1);
  elldou([E1[1],E1[2],1],E1[3],E1[4],n1);
```

$$(2)\,(251)$$

$$[0, 1, 0]$$

$$[132, 185, 1] \tag{11.4.38}$$

```
> n2:=251; E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$n2 := 251$$

$$E2 := [10, 164, 111, 188]$$

$$m2 := 261 \tag{11.4.39}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [74, 3, 218, 82]$$

$$m2 := 264 \tag{11.4.40}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [149, 39, 57, 36]$$

$$m2 := 252 \tag{11.4.41}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [220, 225, 106, 119]$$

$$m2 := 257 \tag{11.4.42}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [10, 29, 133, 17]$$

$$m2 := 275 \tag{11.4.43}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [31, 39, 115, 42]$$

$$m2 := 232 \tag{11.4.44}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [151, 79, 95, 195]$$

$$m2 := 237 \tag{11.4.45}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [10, 22, 168, 63]$$

$$m2 := 234 \tag{11.4.46}$$

```
> E2 := [10, 22, 168, 63]; m2 := 234;
```

$$E2 := [10, 22, 168, 63]$$

$$m2 := 234 \tag{11.4.47}$$

```
> ifactor(m2); ellmul([E2[1],E2[2],1],m2,E2[3],E2[4],n2);
```

```
elldou([E2[1],E2[2],1],E2[3],E2[4],n2);
```

$$(2)\,(3)^2\,(13)$$

$$[0,\ 1,\ 0]$$

$$[201,\ 137,\ 1] \tag{11.4.48}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [179,\ 127,\ 57,\ 164]$$

$$m2 := 281 \tag{11.4.49}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [66,\ 241,\ 120,\ 111]$$

$$m2 := 229 \tag{11.4.50}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [208,\ 64,\ 196,\ 165]$$

$$m2 := 264 \tag{11.4.51}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [129,\ 35,\ 188,\ 179]$$

$$m2 := 240 \tag{11.4.52}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [83,\ 223,\ 221,\ 2]$$

$$m2 := 264 \tag{11.4.53}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [158,\ 208,\ 30,\ 23]$$

$$m2 := 265 \tag{11.4.54}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [53,\ 66,\ 240,\ 136]$$

$$m2 := 234 \tag{11.4.55}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [186,\ 230,\ 65,\ 179]$$

$$m2 := 267 \tag{11.4.56}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [139,\ 176,\ 100,\ 89]$$

$$m2 := 256 \tag{11.4.57}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [89,\ 34,\ 14,\ 0]$$

$$m2 := 252 \tag{11.4.58}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [153, 99, 117, 125]$$

$$m2 := 246 \tag{11.4.59}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [151, 135, 190, 93]$$

$$m2 := 240 \tag{11.4.60}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [77, 150, 237, 20]$$

$$m2 := 276 \tag{11.4.61}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [72, 149, 240, 141]$$

$$m2 := 280 \tag{11.4.62}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [159, 225, 8, 243]$$

$$m2 := 249 \tag{11.4.63}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [52, 2, 97, 183]$$

$$m2 := 272 \tag{11.4.64}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [68, 124, 130, 81]$$

$$m2 := 270 \tag{11.4.65}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [72, 215, 164, 19]$$

$$m2 := 259 \tag{11.4.66}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [187, 109, 93, 112]$$

$$m2 := 241 \tag{11.4.67}$$

```
> E2:=ellrand(n2); m2:=ellcount(E2[3],E2[4],n2);
```

$$E2 := [179, 15, 156, 173]$$

$$m2 := 262 \tag{11.4.68}$$

```
> E2 := [179, 15, 156, 173]; m2 := 262;
```

$$E2 := [179, 15, 156, 173]$$

$$m2 := 262 \tag{11.4.69}$$

```
> ifactor(m2); ellmul([E2[1],E2[2],1],m2,E2[3],E2[4],n2);
  elldou([E2[1],E2[2],1],E2[3],E2[4],n2);
```

$$(2)\,(131)$$

$$[0, 1, 0]$$

$$[219, 125, 1] \tag{11.4.70}$$

```
> n3:=131; E3:=ellrand(n3); m3:=ellcount(E3[3],E3[4],n3);
```

$$n3 := 131$$

$$E3 := [50, 130, 70, 12]$$

$$m3 := 127 \tag{11.4.71}$$

```
> E3:=ellrand(n3); m3:=ellcount(E3[3],E3[4],n3);
```

$$E3 := [104, 73, 48, 108]$$

$$m3 := 113 \tag{11.4.72}$$

```
> E3:=ellrand(n3); m3:=ellcount(E3[3],E3[4],n3);
```

$$E3 := [32, 128, 104, 69]$$

$$m3 := 146 \tag{11.4.73}$$

```
> E3 := [32, 128, 104, 69]; m3 := 146;
```

$$E3 := [32, 128, 104, 69]$$

$$m3 := 146 \tag{11.4.74}$$

```
> ifactor(m3); ellmul([E3[1],E3[2],1],m3,E3[3],E3[4],n3);
  elldou([E3[1],E3[2],1],E3[3],E3[4],n3);
```

$$(2)\,(73)$$

$$[0, 1, 0]$$

$$[110, 26, 1] \tag{11.4.75}$$

```
> n4:=73;
```

11.5. Atkin tesztje.

```
> interface(echo=2);
```

$$1 \tag{11.5.1}$$

```
> ;
> #
  # For a given negative discriminant -D
  # we calculate the ideal class number.
  #

  idealclassnumber:=proc(D) local a,b,c,n,h; h:=0;
  for a while 3*a^2<=D do
    b:=D mod 2; n:=(b^2+D)/4;
    while b<=a do
      if n mod a=0 then c:=n/a;
        if c>=a and igcd(a,b,c)=1 then
          if a=c or b=0 or b=a then h:=h+1 else h:=h+2 fi;
```

```
        fi;
      fi; n:=n+b+1; b:=b+2;
    od;
  od; h; end;
```

$$idealclassnumber := \mathbf{proc}(\mathrm{D}) \tag{11.5.2}$$

```
  local a, b, c, n, h;
  h := 0;
  for a while 3 * a^2 <= D do
      b := mod(D, 2);
      n := 1 / 4 * b^2 + 1 / 4 * D;
      while b <= a do
          if mod(n, a) = 0 then
              c := n / a;
              if a <= c and igcd(a, b, c) = 1 then
                  if a = c or b = 0 or b = a then
                      h := h + 1
                  else
                      h := h + 2
                  end if
              end if
          end if;
          n := n + b + 1;
          b := b + 2
      end do
  end do;
  h
end proc
```

```
> #
  # For a given negative discriminant -D
  # this calculates the ideal class number.
  #
  # This versions is slower then the other!
  #

  idealclassnumber2:=proc(D) local a,b,c,n,h,X,x,DD; h:=1;
  if type(D,even) then
    DD:=D/4;
```

```
    for a from 2 while 3*a^2<=D do
      X:=Roots(x^2+DD) mod a;
      for b in X do b:=2*b[1];
        if b>a then b:=b-2*a; fi;
        c:=(b^2+D)/4/a;
        if c>=a and igcd(a,b,c)=1 then
          if a=c and b<0 then next fi;
          h:=h+1;
        fi;
      od;
    od;
  else
    DD:=(D+1)/4;
    for a from 2 while 3*a^2<=D do
      X:=Roots(x^2+x+DD) mod a;
      for b in X do
        b:=2*b[1]+1;
        if b>a then b:=b-2*a; fi;
        c:=(b^2+D)/4/a;
        if c>=a and igcd(a,b,c)=1 then
          if a=c and b<0 then next fi;
          h:=h+1;
        fi;
      od;
    od;
  fi; h; end;
```

```
idealclassnumber2 := proc(D)                                    (11.5.3)
    local a, b, c, n, h, X, x, DD;
    h := 1;
    if type(D, even) then
        DD := 1 / 4 * D;
        for a from 2 while 3 * a^2 <= D do
            X := mod(Roots(x^2 + DD), a);
            for b in X do
                b := 2 * b[1];
                if a < b then
                    b := b - 2 * a
                end if;
                c := 1 / 4 * (b^2 + D) / a;
                if a <= c and igcd(a, b, c) = 1 then
                    if a = c and b < 0 then
```

```
                    next
                end if;
                h := h + 1
            end if
        end do
    end do
else
    DD := 1 / 4 * D + 1 / 4;
    for a from 2 while 3 * a^2 <= D do
        X := mod(Roots(x^2 + x + DD), a);
        for b in X do
            b := 2 * b[1] + 1;
            if a < b then
                b := b - 2 * a
            end if;
            c := 1 / 4 * (b^2 + D) / a;
            if a <= c and igcd(a, b, c) = 1 then
                if a = c and b < 0 then
                    next
                end if;
                h := h + 1
            end if
        end do
    end do
end if;
h
end proc
> #
  # For a given negative discriminant -D
  # this calculates the ideal class number.
  #
  # This versions is faster as others for
  # large discriminants
  #
  # NOT YET FINISHED!!!!
```

```
   #

   idealclassnumbershank:=proc(D,f,b) local P,p,Q,d,DD;
   DD:=2^16; # have to adjust for an appropriate limit
   if D<DD then return(idealclassnumber(D)) fi;
   P:=ceil(D^(1/4.));
   Q:=1.*sqrt(D)/Pi/(1.-numtheory[jacobi](-D,2)/2.);
   Q:=Q/(1-numtheory[jacobi](-D,3)/3.);
   p:=5; d:=2;
   while p<=P do
     if isprime(p) then Q:=Q/(1-numtheory[jacobi](-D,p)/p); fi;
     p:=p+d; d:=6-d;
   od; end;
```

$$idealclassnumbershank := \mathbf{proc}(\mathrm{D}, f, b) \tag{11.5.4}$$

```
local P, p, Q, d, DD;
DD := 65536;
if D < DD then
    return idealclassnumber(D)
end if;
P := ceil(D^(1 / 4.));
Q := 1.*sqrt(D) / (π*(1. - numtheory[jacobi](-D, 2) / 2.));
Q := Q / (1 - numtheory[jacobi](-D, 3) / 3.);
p := 5;
d := 2;
while p <= P do
    if isprime(p) then
        Q := Q / (1 - numtheory[jacobi](-D, p) / p)
    end if;
    p := p + d;
    d := 6 - d
end do
end proc
```

```
> #
  # This procedure gives back the next foundamental
  # discriminant larger then d. A discriminant have
  # to be larger then 6, congruent to 3 modulo 4 or congruent
  # to 4 or 8 modulo 16 and a square of an odd prime
  # may not divide it. The result is in form
  # [d,h,q1,q2,..], where d is the discriminant,
```

```
# h is the ideal class number in Q(sgrt(-d)),
# and the q's are odd primes except maybe q1, which may 4 or
8.
#

nextdiscriminant:=proc(d) local dd,dp,L,n,p,i;
dd:=max(d+1,7);
while true do
  if dd mod 16=4 or dd mod 16=8 or dd mod 4=3 then
    L:=[]; n:=dd;
    if type(n,even) then
      for i from 0 while type(n,even) do n:=n/2; od;
      L:=[2^i];
    fi;
    if n>=9 then
      if n mod 3=0 then n:=n/3;
        if n mod 3=0 then dd:=dd+1; next; fi;
        L:=[op(L),3];
      fi;
    fi;
    dp:=2; p:=5;
    while n>=p^2 do
      if n mod p=0 then n:=n/p;
        if n mod p=0 then L:=false; break; fi;
        L:=[op(L),p];
      fi;
      p:=p+dp; dp:=6-dp;
    od;
    if L=false then dd:=dd+1; next; fi;
    L:=[dd,idealclassnumber(dd),op(L)];
    if n>1 then L:=[op(L),n] fi;
    break;
  else
    dd:=dd+1;
  fi;
od; L; end;
```

$$nextdiscriminant := \mathbf{proc}(d) \tag{11.5.5}$$

$$\mathbf{local}\ dd,\ dp,\ L,\ n,\ p,\ i;$$

$$dd := \max(d+1,\ 7);$$

$$\mathbf{do}$$

$$\mathbf{if}\ mod(dd,\ 16) = 4\ \mathbf{or}\ mod(dd,\ 16) = 8\ \mathbf{or}\ mod(dd,\ 4) = 3\ \mathbf{then}$$

$$L := [\,];$$

$$n := dd;$$

$$\mathbf{if}\ type(n,\ even)\ \mathbf{then}$$

```
    for i from 0 while type(n, even) do
        n := 1 / 2 * n
    end do;
    L := [2^i]
end if;
if 9 <= n then
    if mod(n, 3) = 0 then
        n := 1 / 3 * n;
        if mod(n, 3) = 0 then
            dd := dd + 1;
            next
        end if;
        L := [op(L), 3]
    end if
end if;
dp := 2;
p := 5;
while p^2 <= n do
    if mod(n, p) = 0 then
        n := n / p;
        if mod(n, p) = 0 then
            L := false;
            break
        end if;
        L := [op(L), p]
    end if;
    p := p + dp;
    dp := 6 - dp
end do;
if L = false then
    dd := dd + 1;
    next
```

```
            end if;
            L := [dd, idealclassnumber(dd), op(L)];
            if 1 < n then
                L := [op(L), n]
            end if;
            break
        else
            dd := dd + 1
        end if
    end do;
    L
end proc
```

```
> #
  # This procedure write to a file all the foundamental
  discriminants
  # in the interval II. The ideal class number h also
  calculated,
  # and written. First h is written on two bytes, then the
  number
  # of the factors on one byte. The last factor is written on 4
  # bytes, the second and third last factors are written on 2
  bytes,
  # all other factors are written on one byte.
  #

  discriminants:=proc(II,filename) local x,L,B,D,id;
  x:=op(1,II)-1;
  if x<7 then
    id:=fopen(filename,WRITE,BINARY)
  else
    id:=fopen(filename,APPEND,BINARY)
  fi;
  B:=op(2,II); x:=nextdiscriminant(x); D:=x[1];
  while D<=B do
    writediscriminant(x,id); x:=nextdiscriminant(D); D:=x[1];
  od;
  fclose(id);
  end;
```

```
discriminants := proc(II, filename)                    (11.5.6)
    local x, L, B, D, id;
    x := op(1, II) - 1;
```

```
    if x < 7 then
        id := fopen(filename, WRITE, BINARY)
    else
        id := fopen(filename, APPEND, BINARY)
    end if;
    B := op(2, II);
    x := nextdiscriminant(x);
    D := x[1];
    while D <= B do
        writediscriminant(x, id);
        x := nextdiscriminant(D);
        D := x[1]
    end do;
    fclose(id)
end proc
```

```
> #
  # This procedure write to a file a discriminant having size
  # up to ds bytes; if ds is not given, ds=4 is the default
  value.
  # First h(-d) is written on hs bytes; if hs is not given,
  # we presuppose hs=ceil(ds/2+1/2), which is large enough
  # to contain the ideal class number h(-d)<sqrt(d)ln(sqrt(d))
  /Pi
  # whenever ds<=18. Then the number of the factors is written
  # on one byte, and the factors starting with 4 or 8, if
  # there is such a factor and continuing with odd factors in
  # increasing order. The last factor is written on ds bytes,
  # the second last factor on ceil(ds/2) bytes, the third last
  # factors on ceil(ds/3) bytes, etc. If ps is also given, all
  # factors are written on at most ps bytes. Discriminants
  which
  # cannot included in this format couse error.
  #

  writediscriminant:=proc(L,id) local i,j,k,l,B,ds,hs,ps;
  if nargs<2 then ERROR(`not enough args`) fi;
  if nargs=2 then ds:=4 else ds:=args[3] fi;
  if nargs<=3 then hs:=ceil((ds/2+1/2)) else hs:=args[4] fi;
  if nargs<=4 then ps:=ds else ps:=args[5] fi;
  B:=convert(L[2],base,256);
  if nops(B)>hs then ERROR(`too large ideal class number`,L[2])
```

```
  fi;
  if nops(B)<hs then B:=[op(B),0$j=nops(B)+1..hs] fi;
  writebytes(id,B);
  i:=3; k:=nops(L)-2;
  writebytes(id,[k]);
  while k>=ds do writebytes(id,[L[i]]); i:=i+1; k:=k-1; od;
  while k>0 do l:=ceil(ds/k); if l>ps then l:=ps fi;
    B:=convert(L[i],base,256);
    if nops(B)>l then ERROR(`too large prime factor`,L[i]) fi;
    if nops(B)<l then B:=[op(B),0$j=nops(B)+1..l] fi;
    writebytes(id,B); i:=i+1; k:=k-1;
  od; end;
```

$writediscriminant := \mathbf{proc}(L, id)$ (11.5.7)

local $i, j, k, l, B, ds, hs, ps$;

if $nargs < 2$ **then**

$ERROR(not\ enough\ args)$

end if;

if $nargs = 2$ **then**

$ds := 4$

else

$ds := args[3]$

end if;

if $nargs <= 3$ **then**

$hs := \mathrm{ceil}(1/2 * ds + 1/2)$

else

$hs := args[4]$

end if;

if $nargs <= 4$ **then**

$ps := ds$

else

$ps := args[5]$

end if;

$B := convert(L[2], base, 256)$;

if $hs < nops(B)$ **then**

$ERROR(too\ large\ ideal\ class\ number, L[2])$

end if;

```
    if nops(B) < hs then
        B := [op(B), $(0, j = nops(B) + 1 ..hs)]
    end if;
    writebytes(id, B);
    i := 3;
    k := nops(L) - 2;
    writebytes(id, [k]);
    while ds <= k do
        writebytes(id, [L[i]]);
        i := i + 1;
        k := k - 1
    end do;
    while 0 < k do
        l := ceil(ds / k);
        if ps < l then
            l := ps
        end if;
        B := convert(L[i], base, 256);
        if l < nops(B) then
            ERROR(too large prime factor, L[i])
        end if;
        if nops(B) < l then
            B := [op(B), $(0, j = nops(B) + 1 ..l)]
        end if;
        writebytes(id, B);
        i := i + 1;
        k := k - 1
    end do
end proc
> #
  # This procedure read from a file a discriminant having size
  # up to ds bytes; if ds is not given, ds=4 is the default
  value.
  # First h(-d) is read on hs bytes; if hs is not given,
```

```
# we presuppose hs=ceil(ds/2+1/2), which is large enough
# to contain the ideal class number h(-d)<sqrt(d)ln(sqrt(d))
/Pi
# whenever ds<19. Then the number of the factors on one byte
# is read, and the factors starting with 4 or 8, if there is
# such a factor and continuing with odd factors in increasing
# order. The last factor is supposed to be written on ds
bytes,
# the second last factor on ceil(ds/2) bytes, the third last
# factor on ceil(ds/3) bytes, etc. If ps is also given, all
# factors are supposed to be written on at most ps bytes.
#

readdiscriminant:=proc(id)
local i,j,k,B,l,d,L,ds,hs,ps;
if nargs<1 then ERROR(`not enough args`) fi;
if nargs=1 then ds:=4 else ds:=args[2] fi;
if nargs<=2 then hs:=ceil((ds/2+1/2)) else hs:=args[3] fi;
if nargs<=3 then ps:=ds else ps:=args[4] fi;
B:=readbytes(id,hs); if B=0 then RETURN(NULL) fi;
L:=[convert([B[j]*256^(j-1)$j=1..nops(B)],`+`)];
B:=readbytes(id,1);k:=B[1]; d:=1;
while k>=ds do
  B:=readbytes(id,1); L:=[op(L),B[1]];
  d:=d*B[1]; k:=k-1;
od;
while k>0 do l:=ceil(ds/k); if l>ps then l:=ps fi;
  B:=readbytes(id,l);
  B:=convert([B[j]*256^(j-1)$j=1..nops(B)],`+`);
  d:=d*B; L:=[op(L),B]; k:=k-1;
od; [d,op(L)]; end;
```

$$readdiscriminant := \mathbf{proc}(id) \tag{11.5.8}$$

local $i, j, k, B, l, d, L, ds, hs, ps$;

if $nargs < 1$ **then**

$ERROR(not\ enough\ args)$

end if;

if $nargs = 1$ **then**

$ds := 4$

else

$ds := args[2]$

end if;

if $nargs <= 2$ **then**

$hs := \mathrm{ceil}(1/2 * ds + 1/2)$

```
else
    hs := args[3]
end if;
if nargs <= 3 then
    ps := ds
else
    ps := args[4]
end if;
B := readbytes(id, hs);
if B = 0 then
    RETURN(NULL)
end if;
L := [convert([$(B[j]*256^(j-1), j = 1..nops(B))], +)];
B := readbytes(id, 1);
k := B[1];
d := 1;
while ds <= k do
    B := readbytes(id, 1);
    L := [op(L), B[1]];
    d := d*B[1];
    k := k-1
end do;
while 0 < k do
    l := ceil(ds/k);
    if ps < l then
        l := ps
    end if;
    B := readbytes(id, l);
    B := convert([$(B[j]*256^(j-1), j = 1..nops(B))], +);
    d := d*B;
    L := [op(L), B];
    k := k-1
```

```
    end do;
    [d, op(L)]
end proc
```

```
> #
  # This procedure sieve out discriminants from a file for
  which
  # the function f gives value true. The result is [n,e] where
  n
  # is the number if good discriminants and e is the sum of 1/h
  (D)'s.
  #

  sievediscriminants:=proc(filename,f) local e,n,x,id;
  id:=fopen(filename,READ,BINARY); e:=0.0; n:=0;
  do
    x:=readdiscriminant(id);
    if x=NULL then break fi;
    if f(x) then n:=n+1; e:=e+1/x[2]; fi;
  od; fclose(id); [n,e]; end;
```

$$sievediscriminants := \mathbf{proc}(filename, f) \tag{11.5.9}$$

```
    local e, n, x, id;
    id := fopen(filename, READ, BINARY);
    e := 0.;
    n := 0;
    do
        x := readdiscriminant(id);
        if x = NULL then
            break
        end if;
        if f(x) then
            n := n + 1;
            e := e + 1 / x[2]
        end if
    end do;
    fclose(id);
    [n, e]
end proc
```

```
> badness:=x->x[2]/2^(nops(x)-3);
```

$$badness := x \rightarrow \frac{x_2}{2^{nops(x)-3}} \tag{11.5.10}$$

2. Find good discriminants

```
> #
  # This procedure calculates square root of the number a
  # modulo a "probable prime" p. If p is not a prime, an error
  # message may be caused.
  #

  modsqrt:=proc(a::integer,p::posint) local k,s,r,n,ra,j,i,z,c,
  x,t,b;
  if p=1 or type(p,even) then ERROR(`not a prime in modsqrt`,p)
  fi;
  if numtheory[jacobi](a,p)<>1 then ERROR(`not a square`,a) fi;
  k:=p-1; s:=0;
  while type(k,even) do s:=s+1; k:=k/2; od;
  k:=(k-1)/2; r:=a&^(k+1) mod p;n:=a&^(2*k+1) mod p;
  ra:=rand(p); j:=1;
  for i to 100 while j<>-1 do
    z:=ra(); j:=numtheory[jacobi](z,p);
  od;
  if i=100 then ERROR(`probably not a prime in modsqrt`,p) fi;
  c:=z&^(2*k+1) mod p;
  while n<>1 do x:=n;
    for t from s to 0 by -1 while x<>1 do x:=x^2 mod p; od;
    if t=0 then ERROR(`not a prime in modsqrt`,p) fi;
    b:=c&^(2^(t-1)) mod p; r:=r*b mod p; c:=b^2 mod p;
    n:=n*c mod p; s:=s-t;
  od; r; end;
```

$$\begin{aligned}
&modsqrt := \mathbf{proc}(a::integer,\ p::posint) \\
&\quad \mathbf{local}\ k,\ s,\ r,\ n,\ ra,\ j,\ i,\ z,\ c,\ x,\ t,\ b; \\
&\quad \mathbf{if}\ p = 1\ \mathbf{or}\ type(p,\ even)\ \mathbf{then} \\
&\qquad ERROR(not\ a\ prime\ in\ modsqrt,\ p) \\
&\quad \mathbf{end\ if}; \\
&\quad \mathbf{if}\ numtheory[jacobi](a,\ p) <> 1\ \mathbf{then} \\
&\qquad ERROR(not\ a\ square,\ a) \\
&\quad \mathbf{end\ if}; \\
&\quad k := p - 1; \\
&\quad s := 0; \\
&\quad \mathbf{while}\ type(k,\ even)\ \mathbf{do}
\end{aligned} \tag{11.5.11}$$

```
        s := s + 1;
        k := 1 / 2 * k
    end do;
    k := 1 / 2 * k - 1 / 2;
    r := mod(a &^ (k + 1), p);
    n := mod(a &^ (2 * k + 1), p);
    ra := rand(p);
    j := 1;
    for i to 100 while j <> -1 do
        z := ra();
        j := numtheory[jacobi](z, p)
    end do;
    if i = 100 then
        ERROR(probably not a prime in modsqrt, p)
    end if;
    c := mod(z &^ (2 * k + 1), p);
    while n <> 1 do
        x := n;
        for t from s by -1 to 0 while x <> 1 do
            x := mod(x^2, p)
        end do;
        if t = 0 then
            ERROR(not a prime in modsqrt, p)
        end if;
        b := mod(c &^ (2^(t - 1)), p);
        r := mod(r * b, p);
        c := mod(b^2, p);
        n := mod(n * c, p);
        s := s - t
    end do;
    r
end proc
```

```
> #
  # This procedure put all the primes p below P to the
  # table goodprimes, for which (n|p)=1, where
  # n is a given probable prime. Differences between
  # odd primes are read from a file.
  #

  goodprimes:=proc(filename,P,n,goodprimes) local p,id,d;
  if numtheory[jacobi](-1,n)=1 then goodprimes[-1]:=-1
    else goodprimes[-1]:=0; fi;
  if numtheory[jacobi](-2,n)=1 then goodprimes[-2]:=-2
    else goodprimes[-2]:=0; fi;
  if numtheory[jacobi](2,n)=1 then goodprimes[2]:=2
    else goodprimes[2]:=0 fi;
  id:=fopen(filename,READ,BINARY); p:=3;
  while p<P do
    if numtheory[jacobi](n,p)=1 then
      goodprimes[p]:=1;
    else
      goodprimes[p]:=0;
    fi;
    d:=readdiff(id);
    if d=NULL then ERROR(`not enough prime differences`) fi;
    p:=p+d;
  od; fclose(id); end;
```

$$goodprimes := \mathbf{proc}(filename, P, n, goodprimes) \tag{11.5.12}$$

local p, id, d;

if $numtheory[jacobi](-1, n) = 1$ **then**

$goodprimes[-1] := -1$

else

$goodprimes[-1] := 0$

end if;

if $numtheory[jacobi](-2, n) = 1$ **then**

$goodprimes[-2] := -2$

else

$goodprimes[-2] := 0$

end if;

if $numtheory[jacobi](2, n) = 1$ **then**

$goodprimes[2] := 2$

else

$goodprimes[2] := 0$

```
    end if;
    id := fopen(filename, READ, BINARY);
    p := 3;
    while p < P do
        if numtheory[jacobi](n, p) = 1 then
            goodprimes[p] := 1
        else
            goodprimes[p] := 0
        end if;
        d := readdiff(id);
        if d = NULL then
            ERROR(not enough prime differences)
        end if;
        p := p + d
    end do;
    fclose(id)
end proc
```

```
> #
  # This procedure put the square root of p or -p
  # for all the odd primes p below P to the
  # table roots, for which (n|p)=1, where
  # n is a given probable prime. Differences between
  # odd primes are read from a file. Good primes came
  # from table goodprimes.
  #

  goodprimeroots:=proc(filename,P,n,goodprimes,roots) local p,
  id,d;
  if goodprimes[-1]<>0 then roots[-1]:=modsqrt(n-1,n) fi;
  if goodprimes[-2]<>0 then roots[-2]:=modsqrt(n-2,n) fi;
  if goodprimes[2]<>0 then roots[2]:=modsqrt(2,n) fi;
  id:=fopen(filename,READ,BINARY); p:=3;
  if n mod 4=1 then
    while p<P do
      if goodprimes[p]<>0 then roots[p]:=modsqrt(p,n); fi;
      d:=readdiff(id);
      if d=NULL then ERROR(`not enough prime differences`) fi;
      p:=p+d;
    od;
  else
```

```
    while p<P do
      if goodprimes[p]<>0 then
        if p mod 4=1 then
          roots[p]:=modsqrt(p,n);
        else
          roots[p]:=modsqrt(n-p,n);
        fi;
      fi;
      d:=readdiff(id);
      if d=NULL then ERROR(`not enough prime differences`) fi;
      p:=p+d;
    od;
  fi; fclose(id); end;
```

$$goodprimeroots := \mathbf{proc}(filename, P, n, goodprimes, roots) \tag{11.5.13}$$

local p, id, d;

if $goodprimes[-1] <> 0$ **then**

$roots[-1] := modsqrt(n-1, n)$

end if;

if $goodprimes[-2] <> 0$ **then**

$roots[-2] := modsqrt(n-2, n)$

end if;

if $goodprimes[2] <> 0$ **then**

$roots[2] := modsqrt(2, n)$

end if;

$id := fopen(filename, READ, BINARY)$;

$p := 3$;

if $mod(n, 4) = 1$ **then**

while $p < P$ **do**

if $goodprimes[p] <> 0$ **then**

$roots[p] := modsqrt(p, n)$

end if;

$d := readdiff(id)$;

if $d = NULL$ **then**

$ERROR(not\ enough\ prime\ differences)$

end if;

$p := p + d$

end do

```
    else
        while p < P do
            if goodprimes[p] <> 0 then
                if mod(p, 4) = 1 then
                    roots[p] := modsqrt(p, n)
                else
                    roots[p] := modsqrt(n - p, n)
                end if
            end if;
            d := readdiff(id);
            if d = NULL then
                ERROR(not enough prime differences)
            end if;
            p := p + d
        end do
    end if;
    fclose(id)
end proc
```

```
> #
  # This procedure gives the next good discriminant for
  # the probably prime number n. The function value have to be
  # true for the discriminant. The condition (-d|n)=1 is
  checked.
  # Good prime roots came from table roots.
  #

  nextgooddiscriminantfor:=proc(id,n,f,roots) local x,i,j,s,q,
  ff;
  do
    x:=readdiscriminant(id);
    if x=NULL then return(NULL) fi;
    if f(x) then
      i:=3; q:=x[3]; s:=1; ff:=true;
      if q=4 or q=8 then i:=i+1 else q:=1 fi;
      for j from i to nops(x) do
        if not type(roots[x[j]],integer) then ff:=false; break;
  fi;
        if x[j] mod 4=3 then s:=-s; fi;
      od;
```

```
        if not ff then next; fi;
        i:=n mod 8;
        if i=1 or i=5 then s:=1 else s:=-s; fi;
        if q=8 and (i=3 or i=5) then s:=-s; fi;
        if s=1 then break; fi;
      fi;
    od; x; end;
```

$$nextgooddiscriminantfor := \mathbf{proc}(id, n, f, roots) \tag{11.5.14}$$

```
  local x, i, j, s, q, ff;
  do
    x := readdiscriminant(id);
    if x = NULL then
      return NULL
    end if;
    if f(x) then
      i := 3;
      q := x[3];
      s := 1;
      ff := true;
      if q = 4 or q = 8 then
        i := i + 1
      else
        q := 1
      end if;
      for j from i to nops(x) do
        if not type(roots[x[j]], integer) then
          ff := false;
          break
        end if;
        if mod(x[j], 4) = 3 then
          s := -s
        end if
      end do;
      if not ff then
        next
```

```
            end if;
            i := mod(n, 8);
            if i = 1 or i = 5 then
                s := 1
            else
                s := -s
            end if;
            if q = 8 and (i = 3 or i = 5) then
                s := -s
            end if;
            if s = 1 then
                break
            end if
        end if
    end do;
    x
end proc
```

```
> #
  # This procedure sieve out discriminants from a file for
  which
  # the function f gives value true, and conditions (-d|n)=1,
  (n|p)=1
  # are satisfied. The result is [m,e] where m is the number if
  good
  # discriminants and e is the expected number of elliptic
  curves.
  #

  gooddiscriminantsfor:=proc(filename,n,f) local e,m,x,id;
  id:=fopen(filename,READ,BINARY);
  e:=0.0; m:=0;
  do
    x:=nextgooddiscriminantfor(id,n,f);
    if x=NULL then break fi;
    m:=m+1; e:=e+2^(nops(x)-2)/x[2];
  od; fclose(id); [m,e]; end;
```

$$gooddiscriminantsfor := \mathbf{proc}(filename, n, f) \tag{11.5.15}$$

```
    local e, m, x, id;
    id := fopen(filename, READ, BINARY);
```

```
    e := 0.;
    m := 0;
    do
        x := nextgooddiscriminantfor(id, n, f);
        if x = NULL then
            break
        end if;
        m := m + 1;
        e := e + 2^(nops(x) - 2) / x[2]
    end do;
    fclose(id);
    [m, e]
end proc
```

```
> #
  # This procedure gives the next good discriminant and its
  square
  # root for the probably prime number n. The function value
  have
  # to be true for the discriminant. The condition (-d|n)=1 is
  # checked. Prime roots came from table roots.
  #

  nextgooddiscriminantroot:=proc(id,n,f,roots) local r,x,i,j,s,
  q,ff;
  do
    x:=readdiscriminant(id);
    if x=NULL then return(NULL) fi;
    if f(x) then
      i:=3; q:=x[3]; s:=1; ff:=true;
      if q=4 or q=8 then i:=i+1 else q:=1 fi;
      for j from i to nops(x) do
        if roots[x[j]]=0 then ff:=false; break; fi;
        if x[j] mod 4=3 then s:=-s; fi;
      od;
      if not ff then next; fi;
      j:=n mod 8;
      if j=1 or j=5 then s:=1 else s:=-s; fi;
      if q=8 and (j=3 or j=5) then s:=-s; fi;
      if s=1 then r:=1;
        for j from i to nops(x) do r:=r*roots[x[j]] mod n; od;
        if q=4 then r:=r*2*roots[-1] mod n; fi;
        j:=n mod 8;
```

```
        if q=8 then
          if (j=3 or j=5) then
            r:=r*2*roots[2] mod n
          else
            r:=r*2*roots[-2] mod n
          fi;
        fi;
        break;
      fi;
    fi;
  od; [x[1],r] end;
```

```
nextgooddiscriminantroot := proc(id, n, f, roots)                    (11.5.16)
  local r, x, i, j, s, q, ff;
  do
    x := readdiscriminant(id);
    if x = NULL then
      return NULL
    end if;
    if f(x) then
      i := 3;
      q := x[3];
      s := 1;
      ff := true;
      if q = 4 or q = 8 then
        i := i + 1
      else
        q := 1
      end if;
      for j from i to nops(x) do
        if roots[x[j]] = 0 then
          ff := false;
          break
        end if;
        if mod(x[j], 4) = 3 then
          s := -s
        end if
```

```
        end do;
        if not ff then
            next
        end if;
        j := mod(n, 8);
        if j = 1 or j = 5 then
            s := 1
        else
            s := -s
        end if;
        if q = 8 and (j = 3 or j = 5) then
            s := -s
        end if;
        if s = 1 then
            r := 1;
            for j from i to nops(x) do
                r := mod(r * roots[x[j]], n)
            end do;
            if q = 4 then
                r := mod(2 * r * roots[-1], n)
            end if;
            j := mod(n, 8);
            if q = 8 then
                if j = 3 or j = 5 then
                    r := mod(2 * r * roots[2], n)
                else
                    r := mod(2 * r * roots[-2], n)
                end if
            end if;
            break
        end if
    end if
```

```
    end do;
    [x[1], r]
end proc
```

3. Reduced form we look for

```
> DD:=7; a:=1; b:=-DD; c:=(-DD)*(-DD-1)/4;
```

$$DD := 7$$

$$a := 1$$

$$b := -7$$

$$c := 14 \qquad (11.5.17)$$

```
> DD:=8; a:=1; b:=-DD; c:=(-DD)*(-DD-1)/4;
```

$$DD := 8$$

$$a := 1$$

$$b := -8$$

$$c := 18 \qquad (11.5.18)$$

4. Reduction of forms

```
> #
  # This procedure calculates for a quadratic form a*x*x+b*x*y+
  c*y*y
  # the corresponding reduced quadratic form. The values x, y
  for
  # which the reduced form takes the same value as the original
  # form  for x=1, y=0 also calculated. The new coefficients
  and
  # x, y are given back.
  #

  reduceform:=proc(a,b,c) local aa,bb,cc,x,y,q;
  aa:=a; bb:=b; cc:=c;
  x:=1; y:=0;
  while cc<aa or bb<=-aa or bb>aa or (cc=aa and bb<0) do
    q:=bb+aa-1; q:=(q-(q mod (2*aa)))/2/aa;
    bb:=bb-2*aa*q; cc:=cc-q*(bb+q*aa); x:=x+q*y;
    if cc<aa then q:=x; x:=y; y:=q; q:=aa; aa:=cc; cc:=q; fi;
    if cc=aa and bb<0 then bb:=-bb; x:=-x; fi;
  od; [aa,bb,cc,x,y]; end;
```

$$reduceform := \mathbf{proc}(a, b, c) \qquad (11.5.19)$$

$$\mathbf{local}\ aa, bb, cc, x, y, q;$$

```
    aa := a;
    bb := b;
    cc := c;
    x := 1;
    y := 0;
    while cc < aa or bb <= -aa or aa < bb or cc = aa and bb < 0 do
        q := bb + aa - 1;
        q := 1 / 2 * (q - (mod(q, 2 * aa))) / aa;
        bb := bb - 2 * aa * q;
        cc := cc - q * (bb + aa * q);
        x := x + q * y;
        if cc < aa then
            q := x;
            x := y;
            y := q;
            q := aa;
            aa := cc;
            cc := q
        end if;
        if cc = aa and bb < 0 then
            bb := -bb;
            x := -x
        end if
    end do;
    [aa, bb, cc, x, y]
end proc
```

```
> DD:=7; a:=1; b:=-DD; c:=(-DD)*(-DD-1)/4; reduceform(a,b,c);
```

$$DD := 7$$

$$a := 1$$

$$b := -7$$

$$c := 14$$

$$[1, 1, 2, 1, 0] \qquad (11.5.20)$$

```
> DD:=8; a:=1; b:=-DD; c:=(-DD)*(-DD-1)/4; reduceform(a,b,c);
```

$$DD := 8$$

$$a := 1$$

$$b := -8$$

$$c := 18$$

$$[1, 0, 2, 1, 0] \tag{11.5.21}$$

5. Find the algebraic integer

```
> #
  # This procedure gives the next good discriminant and
  # the trace of the corresponding algebraic integer for
  # the probably prime number n. The function value
  # have to be true for the discriminant.
  #

  nextgoodorder:=proc(id,n,f,roots) local x,t;
  do
    x:=nextgooddiscriminantroot(id,n,f,roots);
    if x=NULL then return(NULL) fi;
    t:=testorder(x[1],n);
    if t<>NULL then break fi;
  od; [x[1],t] end;
```

$$nextgoodorder := \mathbf{proc}(id, n, f, roots) \tag{11.5.22}$$

```
local x, t;
do
    x := nextgooddiscriminantroot(id, n, f, roots);
    if x = NULL then
        return NULL
    end if;
    t := testorder(x[1], n);
    if t <> NULL then
        break
    end if
end do;
[x[1], t]
end proc
> #
```

```
   # This procedure count good orders for
   # the probably prime number n. The function value
   # have to be true for the discriminant.
   #

   countgoodorders:=proc(discr,n,f,roots) local x,i,id; i:=0;
   id:=fopen(discr,READ,BINARY);
   do
     x:=nextgoodorder(id,n,f,roots);
     if x=NULL then break fi; i:=i+2;
   od; fclose(discr); i; end;
```

$$countgoodorders := \mathbf{proc}(discr, n, f, roots) \tag{11.5.23}$$

$$\begin{aligned}
&\mathbf{local}\ x, i, id;\\
&i := 0;\\
&id := fopen(discr, READ, BINARY);\\
&\mathbf{do}\\
&\quad x := nextgoodorder(id, n, f, roots);\\
&\quad \mathbf{if}\ x = NULL\ \mathbf{then}\\
&\quad\quad \mathbf{break}\\
&\quad \mathbf{end\ if};\\
&\quad i := i + 2\\
&\mathbf{end\ do};\\
&fclose(discr);\\
&i\\
&\mathbf{end\ proc}
\end{aligned}$$

```
> #
   # This procedure gives back the list of good orders for
   # the probably prime number n. The function value
   # have to be true for the discriminant.
   #

   goodorders:=proc(discr,n,f) local x,i,id,L; i:=0; L:=[];
   id:=fopen(discr,READ,BINARY);
   do
     x:=nextgoodorder(id,n,f,roots);
     L:=[op(L),x];
     if x=NULL then break fi; i:=i+2;
   od; fclose(discr); L; end;
```

$$goodorders := \mathbf{proc}(discr, n, f) \tag{11.5.24}$$

$$\begin{aligned}
&\mathbf{local}\ x, i, id, L;
\end{aligned}$$

```
    i := 0;
    L := [ ];
    id := fopen(discr, READ, BINARY);
    do
        x := nextgoodorder(id, n, f, roots);
        L := [op(L), x];
        if x = NULL then
            break
        end if;
        i := i + 2
    end do;
    fclose(discr);
    L
end proc
```

```
> #
  # This procedure test whether in the quadratic order with
  # discriminant -D there is an element with norm equal to n,
  # a probable prime. If such an element a+b*sqrt(-D) is found,
  # the trace of it is given back, else the result is NULL.
  #

  testform:=proc(D,n) local a,b,r,al;
  if numtheory[jacobi](-D,n)<>1 then RETURN() fi;
  b:=modsqrt(-D mod n,n);
  if type(b+D,odd) then b:=b+n; fi;
  r:=reduceform(n,b,(b^2+D)/4/n);
  if D mod 4=0 then
    if r[1]<>1 or r[2]<>0 then RETURN() fi;
    2*r[4];
  else
    if r[1]<>1 or r[2]<>1 then RETURN() fi;
    2*r[4]+r[5];
  fi; end;
>
```

6. Factorization of the orders of curves

```
> ;
> ;
>
```

```
>
>
>
> ;
> ;
>
>
>
>
```

7. Set up the Hilbert polynomial

```
> #
  # For a given negative discriminant -D this procedure
  calculates
  # a list of all pairs of numbers [a,b] representing a reduced
  # ideal and gives back the list of these.
  #

  allredideals:=proc(D) local a,b,c,L; L:=[];
  for a while 3*a^2<=D do
    for b from -a to a do
      if b^2+D mod 4*a <> 0 then next fi;
      c:=(b^2+D)/(4*a);
      if c<a then next fi;
      if igcd(a,b,c)>1 then next fi;
      if a=c and b<0 then next fi;
      if b=-a then next fi;
      L:=[op(L),[a,b]];
    od;
  od; L; end;
```

$$allredideals := \mathbf{proc}(\mathrm{D}) \tag{11.5.25}$$

```
local a, b, c, L;
L := [ ];
for a while 3 * a^2 <= D do
    for b from -a to a do
        if mod(b^2 + D, 4 * a) <> 0 then
            next
        end if;
        c := 1 / 4 * (b^2 + D) / a;
        if c < a then
            next
```

```
            end if;
            if 1 < igcd(a, b, c) then
                next
            end if;
            if a = c and b < 0 then
                next
            end if;
            if b = -a then
                next
            end if;
            L := [op(L), [a, b]]
        end do
    end do;
    L
end proc
```

```
> #
  # This procedure calculate directly the parameters g2(L) and
  g3(L)
  # of the elliptic curve over the complex numbers
  corresponding
  # to a lattice k+l*al with a complex number al. The summing
  is
  # done for all k,l with absolute value less then K. You may
  change
  # Digits and increase K to obtain better approximation.
  #

  directparam:=proc(al,K) local k,l,s4,s6; s6:=0;
  for k while k<K do s4:=s4+2/k^4; s6:=s6+2/k^6; od;
  for l while l<K do s4:=s4+2/(l*al)^4; s6:=s6+2/(l*al)^6; od;
  for l while l<K do
    for k while k<K do
      s4:=s4+2/(k+l*al)^4+2/(k-l*al)^4;
      s6:=s6+2/(k+l*al)^6+2/(k-l*al)^6;
    od;
  od; [60*s4,140*s6]; end;
```

```
directparam := proc(al, K)                                    (11.5.26)
    local k, l, s4, s6;
    s6 := 0;
```

```
    for k while k < K do
        s4 := s4 + 2 / k^4;
        s6 := s6 + 2 / k^6
    end do;
    for l while l < K do
        s4 := s4 + 2 / (l^4 * al^4);
        s6 := s6 + 2 / (l^6 * al^6)
    end do;
    for l while l < K do
        for k while k < K do
            s4 := s4 + 2 / (k + l * al)^4 + 2 / (k - l * al)^4;
            s6 := s6 + 2 / (k + l * al)^6 + 2 / (k - l * al)^6
        end do
    end do;
    [60 * s4, 140 * s6]
end proc
```

```
> #
  # This procedure calculate a crude approximation of the j-
  invariant
  # corresponding to a reduced ideal with parameters a,b,-D.
  # Simple summing is done for all k,l with absolute value less
  # then K. You may change Digits and increase K to obtain
  better
  # approximation.
  #

  directjinvariant:=proc(a,b,D,K) local al,p;
  al:=evalf((b+sqrt(-D))/2./a);
  p:=directparam(al,K);
  2^6*3^3*p[1]^3/(p[1]^3-27*p[2]^2);
  end;
```

$$directjinvariant := \mathbf{proc}(a, b, \mathrm{D}, K) \qquad (11.5.27)$$

```
    local al, p;
    al := evalf((b + sqrt(-D)) / (2. * a));
    p := directparam(al, K);
    1728 * p[1]^3 / (p[1]^3 - 27 * p[2]^2)
end proc
```

```
> #
  # This procedure calculates a crude approximation of the
  # Hilbert polynomial for a given discriminant -D. To get the
  # j-invariants, simple summing up for all k,l with absolute
  # value less then K is used. You may change Digits and
  increase K
  # to obtain better approximation.
  #

  directhilbert:=proc(D,K) local p,j,P,L; global X;
  L:=allredideals(D);P:=1;
  for p in L do
    j:=directjinvariant(p[1],p[2],D,K);
    P:=P*(X-j);
  od; P; end;
```

$$directhilbert := \mathbf{proc}(\mathrm{D}, K) \tag{11.5.28}$$

$$\mathbf{local}\ p, j, P, L;$$

$$\mathbf{global}\ X;$$

$$L := allredideals(\mathrm{D});$$

$$P := 1;$$

$$\mathbf{for}\ p\ \mathbf{in}\ L\ \mathbf{do}$$

$$j := directjinvariant(p[1], p[2], \mathrm{D}, K);$$

$$P := P^{*}(X - j)$$

$$\mathbf{end\ do};$$

$$P$$

$$\mathbf{end\ proc}$$

```
> #
  # This procedure calculate the coefficients of the product of
  # two pover series in order below n. The coefficients are
  given
  # as lists starting with the constant term and ending with
  the
  # term with order n-1.
  #

  cauchyprod:=proc(A,B,n) local c,i,j,C; C:=[];
  for i from 0 to n-1 do c:=0;
    for j from 0 to i do c:=c+A[j+1]*B[i-j+1]; od;
    C:=[op(C),c];
  od; C; end;
```

$$cauchyprod := \mathbf{proc}(A, B, n) \tag{11.5.29}$$

$$\mathbf{local}\ c, i, j, C;$$

```
    C := [ ];
    for i from 0 to n - 1 do
        c := 0;
        for j from 0 to i do
            c := c + A[j + 1] * B[i - j + 1]
        end do;
        C := [op(C), c]
    end do;
    C
end proc
```

```
> #
  # This procedure calculate the coefficients of the quotient
  of
  # two pover series in order below n. The coefficients are
  given
  # as lists starting with the constant term and ending with
  the
  # term with order n-1.
  #

  cauchyquo:=proc(A,B,n) local c,i,j,C; C:=[A[1]/B[1]];
  for i to n-1 do c:=0;
    for j to i do c:=c+C[j]*B[i-j+2]; od;
    C:=[op(C),(A[i+1]-c)/B[1]];
  od; C; end;
```

```
cauchyquo := proc(A, B, n)                                  (11.5.30)
    local c, i, j, C;
    C := [A[1] / B[1]];
    for i to n - 1 do
        c := 0;
        for j to i do
            c := c + C[j] * B[i - j + 2]
        end do;
        C := [op(C), (A[i + 1] - c) / B[1]]
    end do;
    C
end proc
```

```
> #
  # This procedure calculate the coefficients of the q-expansion
  # with order below n. The 1/q term will not be given back.
  #

  qexpansion:=proc(n) local c,i,j,C,A,B;
  A:=[1,240*sigma[3](i)$i=1..n+1];
  B:=[1,-504*sigma[5](i)$i=1..n+1];
  C:=cauchyprod(A,A,n+2);
  A:=cauchyprod(A,C,n+2);
  B:=cauchyprod(B,B,n+2);
  C:=[];
  for i to n+1 do C:=[op(C),A[i+1]-B[i+1]]; od;
  C:=cauchyquo(A,C,n+1);
  map(i->1728*i,[C[2..n+1]]);
  end;
```

$$qexpansion := \mathbf{proc}(n) \tag{11.5.31}$$

local c, i, j, C, A, B;

$A := [1, \$(240 * numtheory:-\sigma[3](i), i = 1..n+1)]$;

$B := [1, \$(-504 * numtheory:-\sigma[5](i), i = 1..n+1)]$;

$C := cauchyprod(A, A, n+2)$;

$A := cauchyprod(A, C, n+2)$;

$B := cauchyprod(B, B, n+2)$;

$C := [\,]$;

for i **to** $n+1$ **do**

$C := [op(C), A[i+1] - B[i+1]]$

end do;

$C := cauchyquo(A, C, n+1)$;

$map(\mathbf{proc}(i)$

option *operator*, *arrow*;

$1728 * i$

end proc, $[C[2..n+1]])$

end proc

```
> #
  # This procedure calculate a j-invariant corresponding to a
  # reduced ideal given by a, b, and D. The prescribed
  precision
  # is Dig decimal digits. The coefficients of the q-expansion
  # have to be given by the global list ccoeff. The length of
  this
```

```
   # vector is doubled, if necessary.
   #

   jinvariant:=proc(a,b,D,Dig) local olddig,q,r,qq,eps,i; global
   ccoeff;
   olddig:=Digits; Digits:=Dig;
   q:=evalf(exp(2*Pi*I*(b+sqrt(-D))/2./a));r:=1/q;
   qq:=1.; eps:=10.^(-Dig);
   for i while abs(ccoeff[i]*qq)>abs(r)*eps do
     r:=r+ccoeff[i]*qq; qq:=qq*q;
     if i=nops(ccoeff) then ccoeff:=qexpansion(2*i); fi;
   od; Digits:=olddig; r; end;
```

$$jinvariant := \mathbf{proc}(a, b, \mathrm{D}, Dig) \tag{11.5.32}$$

local *olddig*, *q*, *r*, *qq*, *eps*, *i*;

global *ccoeff*;

$olddig := Digits$;

$Digits := Dig$;

$q := evalf(\exp(2 * \pi * \mathrm{I} * (b + \mathrm{sqrt}(-\mathrm{D})) / (2. * a)))$;

$r := 1 / q$;

$qq := 1.$;

$eps := 10.\wedge(-Dig)$;

for *i* **while** $\mathrm{abs}(r) * eps < \mathrm{abs}(ccoeff[i] * qq)$ **do**

$r := r + ccoeff[i] * qq$;

$qq := qq * q$;

if $i = nops(ccoeff)$ **then**

$ccoeff := qexpansion(2 * i)$

end if

end do;

$Digits := olddig$;

r

end proc

```
> #
   # This procedure calculate the Hilbert polynomial
   corresponding
   # to a discriminant -D. After all reduced ideals are
   calculated
   # a preliminary calculation is done to obtain a bound for the
   # l_1-norm of the coefficient vector of the result and log
   [10]
```

```
# of this + Dig decimal digits will be the precision used.
The
# resulting polynomial of the global variable X and an
estimate
# of the error are given back. The coefficients of the q-
expansion
# have to be given by the global list ccoeff. The length of
this
# list is doubled, if necessary.
#

hilbert:=proc(D,Dig) local olddig,DD,ll,L,i,H,HH,eps,c;
global ccoeff,X;
olddig:=Digits; Digits:=Dig;
DD:=evalf(sqrt(D)); ll:=0; L:=allredideals(D);
for i in L do ll:=ll+log[10.](abs(exp(Pi*DD/i[1]))+2101.);
od;
Digits:=Dig+ceil(ll); H:=1.;
for i in L do H:=expand(H*(X-jinvariant(i[1],i[2],D,Digits)))
; od;
HH:=0; eps:=0.;
for i from 0 to nops(L) do
  c:=coeff(H,X,i); HH:=HH+round(c)*X^i;
  if abs(c-round(c))>eps then eps:=abs(c-round(c)); fi;
od; Digits:=olddig; HH,eps; end;
```

$$hilbert := \mathbf{proc}(\mathrm{D},\ Dig) \tag{11.5.33}$$

local *olddig*, *DD*, *ll*, *L*, *i*, *H*, *HH*, *eps*, *c*;

global *ccoeff*, *X*;

$olddig := Digits$;

$Digits := Dig$;

$DD := evalf(\mathrm{sqrt}(\mathrm{D}))$;

$ll := 0$;

$L := allredideals(\mathrm{D})$;

for i **in** L **do**

$ll := ll + \log[10.](\mathrm{abs}(\exp(\pi * DD / i[1]))) + 2101.$

end do;

$Digits := Dig + \mathrm{ceil}(ll)$;

$H := 1.$;

for i **in** L **do**

$H := expand(H * (X - jinvariant(i[1], i[2], \mathrm{D}, Digits)))$

end do;

```
    HH := 0;
    eps := 0.;
    for i from 0 to nops(L) do
        c := coeff(H, X, i);
        HH := HH + round(c) * X^i;
        if eps < abs(c - round(c)) then
            eps := abs(c - round(c))
        end if
    end do;
    Digits := olddig;
    HH, eps
end proc
```

8. Factoring the Hilbert polynomial

```
> #
  # This procedure find the factorization of the polynomial P
  of
  # the variable X modulo the prime number p in the form of
  product
  # P_i^i, where each P_i is square free. The list of P_i's
  # is given back.
  #

  squarefreefactor:=proc(P,p) local Pm,D,PP,L,LL,Q,R,i,j;
  global X;
  Pm:=P mod p;
  if degree(Pm)<=1 then RETURN([Pm]); fi;
  PP:=diff(Pm,X) mod p;
  if PP=0 then
    for i from 0 while p*i<=degree(Pm) do PP:=PP+coeff(Pm,X,i*
  p)*X^i; od;
    LL:=squarefreefactor(PP,p); L:=[];
    for i to nops(LL) do L:=[op(L),0$j=1..p-1,LL[i]]; od;
    RETURN(L);
  fi;
  D:=Gcd(Pm,PP) mod p; if degree(D)=0 then RETURN([Pm]); fi;
  Q:=Quo(Pm,D,X) mod p; PP:=Gcd(Q,D) mod p; Divide(Q,PP,'R')
  mod p;
  [R,op(squarefreefactor(D,p))];
  end;
```

squarefreefactor := **proc**(*P*, *p*) (11.5.34)

```
    local Pm, D, PP, L, LL, Q, R, i, j;
    global X;
    Pm := mod(P, p);
    if degree(Pm) <= 1 then
        RETURN([Pm])
    end if;
    PP := mod(diff(Pm, X), p);
    if PP = 0 then
        for i from 0 while p*i <= degree(Pm) do
            PP := PP + coeff(Pm, X, p*i) * X^i
        end do;
        LL := squarefreefactor(PP, p);
        L := [];
        for i to nops(LL) do
            L := [op(L), $(0, j = 1..p - 1), LL[i]]
        end do;
        RETURN(L)
    end if;
    D := mod(Gcd(Pm, PP), p);
    if degree(D) = 0 then
        RETURN([Pm])
    end if;
    Q := mod(Quo(Pm, D, X), p);
    PP := mod(Gcd(Q, D), p);
    mod(Divide(Q, PP, 'R'), p);
    [R, op(squarefreefactor(D, p))]
end proc
```

```
> #
  # This procedure try to find a root of a polynomial P of X
  # modulo p. The polynomial is known to be product of degree
  # 1 irreducible polynomials. The parameter K is the limit
  # for the random tries. The root found or FAIL is given back.
  #

  modpolyroot:=proc(P,p,K) local L,PP,PPP,i,r,T,D;
```

```
PP:=1; r:=rand(p); L:=squarefreefactor(P,p);
for i to nops(L) do
  PPP:=L[i];
  if degree(PPP)>0 then
    if degree(PP)=0 or degree(PP)>degree(PPP) then PP:=PPP;
fi;
  fi;
od;
if degree(PP)=0 then RETURN(false) fi;
while degree(PP)>1 do
  for i to K do
    T:=0; T:=X+r(); T:=Powmod(T,(p-1)/2,PP,X) mod p; D:=Gcd
(P,T+1) mod p;
    if degree(D)>0 and degree(D)<degree(PP) then
      if degree(D)<=degree(PP)/2 then PP:=D; else PP:=Quo(PP,
D,X) mod p; fi;
      break;
    fi;
  od;
  if i>K then RETURN(FAIL) fi;
od; -coeff(PP,X,0)/coeff(PP,X,1) mod p; end;
```

$$modpolyroot := \mathbf{proc}(P, p, K) \tag{11.5.35}$$

local L, PP, PPP, i, r, T, D;

$PP := 1$;

$r := rand(p)$;

$L := squarefreefactor(P, p)$;

for i **to** $nops(L)$ **do**

$PPP := L[i]$;

if $0 < degree(PPP)$ **then**

if $degree(PP) = 0$ **or** $degree(PPP) < degree(PP)$ **then**

$PP := PPP$

end if

end if

end do;

if $degree(PP) = 0$ **then**

$RETURN(false)$

end if;

while $1 < degree(PP)$ **do**

for i **to** K **do**

```
            T := 0;
            T := X + r( );
            T := mod(Powmod(T, 1 / 2 * p - 1 / 2, PP, X), p);
            D := mod(Gcd(P, T + 1), p);
            if 0 < degree(D) and degree(D) < degree(PP) then
                if degree(D) <= 1 / 2 * degree(PP) then
                    PP := D
                else
                    PP := mod(Quo(PP, D, X), p)
                end if;
                break
            end if
        end do;
        if K < i then
            RETURN(FAIL)
        end if
    end do;
    mod(-coeff(PP, X, 0) / coeff(PP, X, 1), p)
end proc
```

9. Set up the curves and the points

```
> #
  # This procedure calculate a random point on a given elliptic
  curve
  # modulo n with parameters a,b.
  #

  ellrandpoint:=proc(a,b,n) local i,j,x,y,r;
  r:=rand(n); j:=-1;
  for i to 100 while j<>1 do
    x:=r(); j:=numtheory[jacobi](x^3+a*x+b,n);
  od;
  if i=100 then ERROR(`probably not a prime`,n) fi;
  y:=modsqrt(x^3+a*x+b mod n,n); j:=r();
  if j>n/2 then [x,y,1]; else [x,-y mod n,1]; fi;
  end;
```

$$ellrandpoint := \mathbf{proc}(a, b, n) \tag{11.5.36}$$

```
    local i, j, x, y, r;
    r := rand(n);
    j := -1;
    for i to 100 while j <> 1 do
        x := r();
        j := numtheory[jacobi](x^3 + a*x + b, n)
    end do;
    if i = 100 then
        ERROR(probably not a prime, n)
    end if;
    y := modsqrt(mod(x^3 + a*x + b, n), n);
    j := r();
    if 1 / 2*n < j then
        [x, y, 1]
    else
        [x, mod(-y, n), 1]
    end if
end proc
>
>
```

10. Putting all together

```
> interface(echo=3);
```

$$2 \tag{11.5.37}$$

```
> ;
> #
  # This simple Maple routine calculates the function
  # L(x,a,b)=exp(b*(ln(x))^a*(ln(ln(x)))^(1-a))
  #

  L:=proc(x,a,b) evalf(exp(b*(ln(x))^a*(ln(ln(x)))^(1-a))) end;
```

$$L := \mathbf{proc}(x, a, b) \tag{11.5.38}$$

```
    evalf(exp(b*ln(x)^a*ln(ln(x))^(1 - a)))
end proc
> #
```

```
# Calculation of running time estimates
#
> D2:=n->m(ln(n))*d(n)/ln(d(n))*rep(n);
```

$$D2 := n \rightarrow \frac{m(\ln(n))\, d(n)\, rep(n)}{\ln(d(n))} \tag{11.5.39}$$

```
> D3:=n->m(ln(n))*ln(ln(n))*d(n)/ln(d(n))*rep(n);
```

$$D3 := n \rightarrow \frac{m(\ln(n))\, \ln(\ln(n))\, d(n)\, rep(n)}{\ln(d(n))} \tag{11.5.40}$$

```
> aa:=n->e(n)*b(n)*ln(n)*rep(n);
```

$$aa := n \rightarrow e(n)\, b(n)\, \ln(n)\, rep(n) \tag{11.5.41}$$

```
> bb:=n->ln(e(n)*ln(n))*b(n)/e(n)/ln(n)*m(e(n)*ln(n))*rep(n);
```

$$bb := n \rightarrow \frac{\ln(e(n)\ln(n))\, b(n)\, m(e(n)\ln(n))\, rep(n)}{e(n)\ln(n)} \tag{11.5.42}$$

```
> cc:=n->0;
```

$$cc := n \rightarrow 0 \tag{11.5.43}$$

```
> dd:=n->e(n)*b(n)^(1/2)*m(ln(n))*rep(n);
```

$$dd := n \rightarrow e(n)\, \sqrt{b(n)}\, m(\ln(n))\, rep(n) \tag{11.5.44}$$

```
> ee:=n->e(n)*m(ln(n))*L(b(n),sqrt(2))*rep(n);
```

$$ee := n \rightarrow e(n)\, m(\ln(n))\, L\left(b(n), \sqrt{2}\right) rep(n) \tag{11.5.45}$$

```
> D6:=n->e(n)*ln(n)*m(ln(n))*rep(n);
```

$$D6 := n \rightarrow e(n)\, \ln(n)\, m(\ln(n))\, rep(n) \tag{11.5.46}$$

```
> A:=n->m(h(DD)*ln(n))*ln(n)*rep(n);
```

$$A := n \rightarrow m(h(DD)\ln(n))\, \ln(n)\, rep(n) \tag{11.5.47}$$

```
> B:=n->m(g(DD)*ln(n))*ln(n)*rep(n);
```

$$B := n \rightarrow m(g(DD)\ln(n))\, \ln(n)\, rep(n) \tag{11.5.48}$$

```
> m:=x->x*ln(x)*ln(ln(x));
```

$$m := x \rightarrow x \ln(x) \ln(\ln(x)) \tag{11.5.49}$$

```
> L:=(x,beta)->exp(beta*((ln(x)*ln(ln(x)))^(1/2)));
```

$$L := (x, \beta) \rightarrow e^{\beta \sqrt{\ln(x) \ln(\ln(x))}} \tag{11.5.50}$$

```
> rep:=n->ln(n)/ln(b(n));
```

$$rep := n \rightarrow \frac{\ln(n)}{\ln(b(n))} \tag{11.5.51}$$

```
> h:=x->x^(1/2);
```

$$h := x \rightarrow \sqrt{x} \tag{11.5.52}$$

```
> g:=x->x^(1/2)/ln(x);
```

(11.5.53)

$$g := x \to \frac{\sqrt{x}}{\ln(x)} \tag{11.5.53}$$

```
> DD:=d(n);
```

$$DD := \ln(n)^2 \tag{11.5.54}$$

```
> d:=x->(ln(x))^2; e:=x->(d(x))^(1/2); b:=x->ln(x);
```

$$d := x \to \ln(x)^2$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x) \tag{11.5.55}$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
  expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)}$$

$$\frac{\ln(n)^4 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3}{\ln(\ln(n))}$$

$$\sqrt{\ln(n)^2}\, \ln(n)^3 \ln(\ln(\ln(n)))$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln(\ln(n))}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right) \ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2) \ln(\ln(n))} \tag{11.5.56}$$

```
> d:=x->(ln(x))^2; e:=x->(d(x))^(1/2); b:=x->ln(x);
```

$$d := x \to \ln(x)^2$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x) \tag{11.5.57}$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
  expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)}$$

$$\frac{\ln(n)^4 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3}{\ln(\ln(n))}$$

$$\sqrt{\ln(n)^2}\, \ln(n)^3 \ln(\ln(\ln(n)))$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln(\ln(n))}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right) \ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2) \ln(\ln(n))} \qquad (11.5.58)$$

```
> d:=x->(ln(x))^2; e:=x->(d(x))^(1/2); b:=x->(ln(x))^2;
```

$$d := x \to \ln(x)^2$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x)^2 \qquad (11.5.59)$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
  expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)^2}$$

$$\frac{\ln(n)^4 \ln(\ln(n))^2 \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)^2}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^4}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\,\ln(n)^3\ln\left(\frac{\sqrt{\ln(n)^2}\,\ln(n)}{\ln(\ln(n)^2)}\right)\ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\,\ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2)^2} \tag{11.5.60}$$

```
> b:=x->(ln(x))^2; e:=x->(d(x))^(1/2);
  b:=x->ln(x)^(ln(ln(x))*ln(ln(ln(x)))^(-2));
```

$$b := x \to \ln(x)^2$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x)^{\frac{\ln(\ln(x))}{\ln(\ln(\ln(x)))^2}} \tag{11.5.61}$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
  expand(B(n));
```

$$\frac{\ln(n)^4\ln(\ln(n))\ln(\ln(\ln(n)))}{\ln(\ln(n)^2)\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\ln(n)^4\ln(\ln(n))^2\ln(\ln(\ln(n)))}{\ln(\ln(n)^2)\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\,\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\ln(n)^2}{\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\,\ln(n)^3\ln(\ln(n))\ln(\ln(\ln(n)))}{\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\,\ln(n)^3\ln\left(\sqrt{\ln(n)^2}\,\ln(n)\right)\ln\left(\ln\left(\sqrt{\ln(n)^2}\,\ln(n)\right)\right)}{\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\,\ln(n)^3\ln\left(\frac{\sqrt{\ln(n)^2}\,\ln(n)}{\ln(\ln(n)^2)}\right)\ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\,\ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2)\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)} \tag{11.5.62}$$

```
> d:=x->(ln(x))^2/(ln(ln(x)))^2; e:=x->(d(x))^(1/2); b:=x->ln
  (x);
```

$$d := x \to \frac{\ln(x)^2}{\ln(\ln(x))^2}$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x) \tag{11.5.63}$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
  expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n))^2 \ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right)}$$

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n)) \ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^3}{\ln(\ln(n))}$$

$$\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^3 \ln(\ln(\ln(n)))$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln(\ln(n))}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right) \ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2) \ln(\ln(n))} \tag{11.5.64}$$

```
> d:=x->(ln(x))^2/(ln(ln(x)))^2; e:=x->(d(x))^(1/2);
  b:=x->(ln(x))^3/ln(ln(n))^3;
```

$$d := x \to \frac{\ln(x)^2}{\ln(\ln(x))^2}$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \frac{\ln(x)^3}{\ln(\ln(n))^3} \tag{11.5.65}$$

```
> expand(D2(n));
  expand(D3(n));
```

```
expand(aa(n));
expand(D6(n));
expand(A(n));
expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n)) \ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right) \ln\left(\frac{\ln(n)^3}{\ln(\ln(n))^3}\right)}$$

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right) \ln\left(\frac{\ln(n)^3}{\ln(\ln(n))^3}\right)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^5}{\ln(\ln(n))^3 \ln\left(\frac{\ln(n)^3}{\ln(\ln(n))^3}\right)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^3 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln\left(\frac{\ln(n)^3}{\ln(\ln(n))^3}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln\left(\frac{\ln(n)^3}{\ln(\ln(n))^3}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right) \ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2) \ln\left(\frac{\ln(n)^3}{\ln(\ln(n))^3}\right)} \qquad (11.5.66)$$

```
> d:=x->(ln(x))^2/(ln(ln(x)))^2; e:=x->(d(x))^(1/2); b:=x->(ln
  (x))^2;
```

$$d := x \to \frac{\ln(x)^2}{\ln(\ln(x))^2}$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x)^2 \qquad (11.5.67)$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
```

```
expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n)) \ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right) \ln(\ln(n)^2)}$$

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right) \ln(\ln(n)^2)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^4}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^3 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln(\ln(n)^2)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right) \ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2)^2} \qquad (11.5.68)$$

```
> d:=x->(ln(x))^2/(ln(ln(x)))^2; e:=x->(d(x))^(1/2);
  b:=x->ln(x)^(ln(ln(x))*ln(ln(ln(x)))^(-2));
```

$$d := x \to \frac{\ln(x)^2}{\ln(\ln(x))^2}$$

$$e := x \to \sqrt{d(x)}$$

$$b := x \to \ln(x)^{\frac{\ln(\ln(x))}{\ln(\ln(\ln(x)))^2}} \qquad (11.5.69)$$

```
> expand(D2(n));
  expand(D3(n));
  expand(aa(n));
  expand(D6(n));
  expand(A(n));
  expand(B(n));
```

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln(\ln(n)) \ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right) \ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\ln(n)^4 \ln(\ln(\ln(n)))}{\ln\left(\frac{\ln(n)^2}{\ln(\ln(n))^2}\right) \ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}} \ln(n)^2}{\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\frac{\ln(n)^2}{\ln(\ln(n))^2}}\, \ln(n)^3 \ln(\ln(n)) \ln(\ln(\ln(n)))}{\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right) \ln\left(\ln\left(\sqrt{\ln(n)^2}\, \ln(n)\right)\right)}{\ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)}$$

$$\frac{\sqrt{\ln(n)^2}\, \ln(n)^3 \ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right) \ln\left(\ln\left(\frac{\sqrt{\ln(n)^2}\, \ln(n)}{\ln(\ln(n)^2)}\right)\right)}{\ln(\ln(n)^2) \ln\left(\ln(n)^{\frac{\ln(\ln(n))}{\ln(\ln(\ln(n)))^2}}\right)} \qquad (11.5.70)$$

>
>
>
>
>
>
>
>
>
>
>
>
>
>
>
>

11. Testing a proof

```
> #
  # The following procedure has as input a list
  # [n,a,b,x,y,s,n1,a1,b1,x1,y1,s1,..,nn], and checks whether
  # this list represents a "partial proof" of the primality
  # of n, i.e. a proof with elliptic curves that if nn is a
  # prime then n is a prime, too. The steps are to check
  whether
  # n_i is a prime if n_{i+1} is a prime. This consist of
  cheking
  # that a_i, b_i are the parameters of an elliptic curve
  # modulo n_i, the point P=(x_i,y_i,1)  is on this elliptic
  # curve, that Q=(m_i/n_{i+1})*P<>0 and n_{i+1}*Q=0, the zero
  # of the elliptic curve. Here m_i, the cardinality of the
  # elliptic curve is n_i+1+s_i.
  #

  partialproof:=proc(PP) local i,n,a,b,P,Q,x,y,m,np;
  if not type(LL,list) then RETURN(false) fi;
  if nops(PP) mod 6<>1 then RETURN(false) fi;
  for i to nops(LL) do
    if not type(LL[i],nonnegint) then RETURN(false) fi;
  od;
  for i by 6 while i<nops(PP) do
    n:=PP[i]; if gcd(n,6)>1 then RETURN(false); fi;
    a:=PP[i+1]; b:=PP[i+2];
    if gcd(n,4*a^3+27*b^2)>1 then RETURN(false); fi;
    x:=PP[i+3]; y:=PP[i+4];
    if y^2-x^3-a*x-b mod n<>0 then RETURN(false) fi;
    P:=[x,y,1]; m:=n+1+PP[i+5]; np:=PP[i+6];
    P:=ellmul(P,m/np,a,b,n);
    if P[3]<>1 then RETURN(false); fi;
    P:=ellmul(P,np,a,b,n);
    if P[3]<>0 then RETURN(false); fi;
  od; true; end;
```

$$partialproof := \mathbf{proc}(PP) \tag{11.5.71}$$

local *i*, *n*, *a*, *b*, *P*, *Q*, *x*, *y*, *m*, *np*;

if not $type(LL, list)$ **then**

$RETURN(false)$

end if;

if $mod(nops(PP), 6) <> 1$ **then**

$RETURN(false)$

end if;

for *i* **to** $nops(LL)$ **do**

```
    if not type(LL[i], nonnegint) then
        RETURN(false)
    end if
end do;
for i by 6 while i < nops(PP) do
    n := PP[i];
    if 1 < gcd(n, 6) then
        RETURN(false)
    end if;
    a := PP[i+1];
    b := PP[i+2];
    if 1 < gcd(n, 4*a^3 + 27*b^2) then
        RETURN(false)
    end if;
    x := PP[i+3];
    y := PP[i+4];
    if mod(y^2 - x^3 - a*x - b, n) <> 0 then
        RETURN(false)
    end if;
    P := [x, y, 1];
    m := n + 1 + PP[i+5];
    np := PP[i+6];
    P := ellmul(P, m / np, a, b, n);
    if P[3] <> 1 then
        RETURN(false)
    end if;
    P := ellmul(P, np, a, b, n);
    if P[3] <> 0 then
        RETURN(false)
    end if
end do;
true
```

```
end proc
```

```
> #
  # The following procedure has as input a list
  # [n,a,b,x,y,s,n1,a1,b1,x1,y1,s1,..,nn], and checks whether
  # this list represents a proof of the primality of n with
  # elliptic curves. First the primality of nn checked with
  # trial division. Then the procedure partialproof is called
  # to complete the proof.
  #

  proof:=proc(PP) local i,nn;
  if not type(LL,list) then RETURN(false) fi;
  if nops(PP) mod 6<>1 then RETURN(false) fi;
  if not type(LL[nops(LL)],nonnegint) then RETURN(false) fi;
  nn:=PP[nops(PP)];
  if type(nn,even) then RETURN(false) fi;
  for i from 3 by 2 while i*i<=nn do
    if nn mod i=0 then RETURN(false) fi;
  od; partialproof(PP); end;
```

$$proof := \mathbf{proc}(PP) \tag{11.5.72}$$

local i, nn;

if not $type(LL, list)$ **then**

$RETURN(false)$

end if;

if $mod(nops(PP), 6) <> 1$ **then**

$RETURN(false)$

end if;

if not $type(LL[nops(LL)], nonnegint)$ **then**

$RETURN(false)$

end if;

$nn := PP[nops(PP)]$;

if $type(nn, even)$ **then**

$RETURN(false)$

end if;

for i **from** 3 **by** 2 **while** $i * i <= nn$ **do**

if $mod(nn, i) = 0$ **then**

$RETURN(false)$

end if

```
    end do;
    partialproof(PP)
end proc
```

▶ 12. Polinomfaktorizálás

▶ 13. Az AKS–teszt

▶ 14. A szita módszerek alapjai

▶ 15. Számtest szita

▶ 16. Vegyes problémák