Számítógépes számelmélet

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

▼ 10. Faktorizálás elliptikus görbékkel

```
> restart;
> #
  # 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;
```

$$ellrand := \mathbf{proc}(n) \tag{10.1}$$

```
    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 := mod(y^2 - x^3 - a*x, n);
        d := mod(4*a^3 + 27*b^2, n);
        gcd(d, n)
    end do;
    if % < n and 1 < % then
        return %
    end if;
    [x, y, a, b]
end proc
```

```
> #
   # 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;
```

$$elladd := \mathbf{proc}(P, Q, a, b, n) \tag{10.2}$$

```
    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^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 := proc(P, a, b, n)                                    (10.3)
    local l, d;
    if P[3] = 0 then
        return P
    end if;
    if P[2] = 0 then
        return [0, 1, 0]
    end if;
```

```
    d := igcdex(2*P[2], n, 'l');
    if 1 < d and d < n then
        return d
    end if;
    l := mod((3*P[1]^2 + a)*l, n);
    mod(l^2 - 2*P[1], n);
    [%, mod(l*(P[1] - %) - P[2], n), 1]
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 := proc(P, k, a, b, n)                                    (10.4)
    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 program compute k*P on an elliptic curve
  # with parameters a,b mod n, where k=k(s,B) is the product of
  # all prime-powers p^ep for which p<=s and p^ep<=B.
  # B is roughly the bound for the prime factor we
  # try to found. Usually s is some hundred and B is some million.
  # If a divisor d of n found then d is given back.
  # We suppose that gcd(n,6)=1.
  #

  elltry:=proc(P,s,B,a,b,n) local lB,p,ep,Q;
  Q:=P; lB:=evalf(ln(B));
  for p from 2 to s do
    if isprime(p) then
      ep:=floor(lB/evalf(ln(p)));
      Q:=ellmul(Q,p^ep,a,b,n);
      if type(Q,integer) then return Q fi;
    fi;
  od; Q; end;
```

```
elltry := proc(P, s, B, a, b, n)                                    (10.5)
    local lB, p, ep, Q;
    Q := P;
    lB := evalf(ln(B));
    for p from 2 to s do
        if isprime(p) then
            ep := floor(lB / evalf(ln(p)));
            Q := ellmul(Q, p^ep, a, b, n);
            if type(Q, integer) then
                return Q
            end if
        end if
    end do;
    Q
```

```
end proc
> #
  # This program try to split an integer n with the
  # elliptic curve method. The parameters are chosen to be
  # optimal to find prime factors below
  # P with probability 1-p. Better to remove first
  # small prime divisors. A factor found is given back.
  #

  ellsplit:=proc(n,P,p) local d,B,s,K,i,a,b,Q;
  if isprime(n) then RETURN([n]) fi;
  B:=ceil(P+2.*sqrt(P)+1);
  s:=evalf(exp(sqrt(ln(P)*ln(ln(P))/2)));
  s:=ceil(max(3,s));
  K:=ceil(max(3,-s*ln(p)));
  for i to K do;
    d:=ellrand(n);
    if type(d,integer) then break fi;
    Q:=[d[1],d[2],1]; a:=d[3]; b:=d[4];
    d:=elltry(Q,s,B,a,b,n);
    if type(d,integer) then break fi;
  od;
  if type(d,integer) then d else 1 fi; end;
```

```
ellsplit := proc(n, P, p)                                    (10.6)
  local d, B, s, K, i, a, b, Q;
  if isprime(n) then
    RETURN([n])
  end if;
  B := ceil(P + 2. * sqrt(P) + 1);
  s := evalf(exp(sqrt(1 / 2 * ln(P) * ln(ln(P)))));
  s := ceil(max(3, s));
  K := ceil(max(3, -s * ln(p)));
  for i to K do
    d := ellrand(n);
    if type(d, integer) then
      break
    end if;
    Q := [d[1], d[2], 1];
    a := d[3];
    b := d[4];
    d := elltry(Q, s, B, a, b, n);
    if type(d, integer) then
```

```
            break
        end if
    end do;
    if type(d, integer) then
        d
    else
        1
    end if
end proc
```

```
> ellsplit(1001,20,0.001);
```

13 (10.7)

10.1. Kötegelt végrehajtás.

```
> #
  # Batch modular inverse modulo n for all L[i] or a divisor of
  n.
  #

  batchmodinv:=proc(L,n) local i,l,d,LL,LLL;
  if nops(L)=1 then
    d:=igcdex(L[1],n,'l');
    if 1<d then return d fi;
    [l];
  else
    LL:=[]; i:=1;
    while i<=nops(L) do
      if i=nops(L) then
        LL:=[op(LL),L[i] mod n]
      else
        LL:=[op(LL),L[i]*L[i+1] mod n]
      fi; i:=i+2;
    od;
    LL:=batchmodinv(LL,n);
    if type(LL,integer) then return LL fi;
    while i<=nops(L) do
      if i=nops(L) then
        LL:=[op(LL),L[i] mod n]
      else
        LL:=[op(LL),L[i]*L[i+1] mod n]
      fi; i:=i+2;
    od;
    LLL:=[]; i:=1;
    while i<=nops(L) do
      if i=nops(L) then
```

```
        LLL:=[op(LLL),LL[(i-1)/2]]
      else
        LLL:=[op(LLL),L[i+1]*LL[(i-1)/2] mod n,L[i]*LL[(i-1)/2]
  mod n]
      fi; i:=i+2;
    od;
    LLL;
  fi; end;
```

$$batchmodinv := \mathbf{proc}(L, n) \tag{10.1.1}$$

```
  local i, l, d, LL, LLL;
  if nops(L) = 1 then
      d := igcdex(L[1], n, 'l');
      if 1 < d then
          return d
      end if;
      [l]
  else
      LL := [ ];
      i := 1;
      while i <= nops(L) do
          if i = nops(L) then
              LL := [op(LL), mod(L[i], n)]
          else
              LL := [op(LL), mod(L[i]*L[i+1], n)]
          end if;
          i := i + 2
      end do;
      LL := modinvbatch(LL, n);
      if type(LL, integer) then
          return LL
      end if;
      while i <= nops(L) do
          if i = nops(L) then
              LL := [op(LL), mod(L[i], n)]
          else
              LL := [op(LL), mod(L[i]*L[i+1], n)]
          end if;
          i := i + 2
      end do;
```

```
        LLL := [ ];
        i := 1;
        while i <= nops(L) do
            if i = nops(L) then
                LLL := [op(LLL), LL[1 / 2 * i - 1 / 2]]
            else
                LLL := [op(LLL), mod(L[i + 1] * LL[1 / 2 * i - 1 / 2],
                n), mod(L[i] * LL[1 / 2 * i - 1 / 2], n)]
            end if;
            i := i + 2
        end do;
        LLL
    end if
end proc
```

10.2. Szorzás egészekkel.

```
> #
  # Doubling on an elliptic "curve" modulo n, where gcd(n,6)=1.
  # x is the first coordinate of the point to double and a, b
  # are the parameters. The return value is either a list, the
  first
  # coordinate of the double of the point or a divisor d of n.
  #

  ellxdou:=proc(x,a,b,n) local l,d;
  d:=igcdex(4*(x^3+a*x+b),n,'l');
  if 1<d then d else [l*((x^2-a)^2-8*b*x) mod n] fi;
  end;
```

```
ellxdou := proc(x, a, b, n)                                    (10.2.1)
    local l, d;
    d := igcdex(4 * x^3 + 4 * a * x + 4 * b, n, 'l');
    if 1 < d then
        d
    else
        [mod(l * ((x^2 - a)^2 - 8 * b * x), n)]
    end if
end proc
```

```
> #
  # Calculation of the triple of first coordinates
  # [x,x_2i,x_2i+1] if t=0 or [x,x_2i+1,x_2i+2] if t=1
```

```
# from the triple of first coordinates [x,x_i,x_i+1]
# on an elliptic "curve" modulo n, where gcd(n,6)=1, and a, b
# are the parameters. The return value is this list
# or a divisor d of n.
#

ellnextx:=proc(L,t,a,b,n) local l,d,x,xi,xip,xii,xiip;
x:=L[1]; xi:=L[2]; xip:=L[3];
if t=0 then
  d:=igcdex(4*(xi^3+a*xi+b),n,'l');
  if 1<d then return d fi;
  xii:=l*((xi^2-a)^2-8*b*xi) mod n;
  d:=igcdex(x*(xi-xip)^2,n,'l');
  if 1<d then return d fi;
  xiip:=l*((a-xi*xip)^2-4*b*(xi+xip)) mod n;
else
  d:=igcdex(x*(xi-xip)^2,n,'l');
  if 1<d then return d fi;
  xii:=l*((a-xi*xip)^2-4*b*(xi+xip)) mod n;
  d:=igcdex(4*(xip^3+a*xip+b),n,'l');
  if 1<d then return d fi;
  xiip:=l*((xip^2-a)^2-8*b*xip) mod n;
fi; [x,xii,xiip] end;
```

$ellnextx := \mathbf{proc}(L, t, a, b, n)$ (10.2.2)

$\mathbf{local}\ l, d, x, \xi, xip, xii, xiip;$

$x := L[1];$

$\xi := L[2];$

$xip := L[3];$

$\mathbf{if}\ t = 0\ \mathbf{then}$

$d := igcdex(4 * \xi\text{^}3 + 4 * a * \xi + 4 * b, n, 'l');$

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

$\mathbf{return}\ d$

$\mathbf{end\ if};$

$xii := mod(l * ((\xi\text{^}2 - a)\text{^}2 - 8 * b * \xi), n);$

$d := igcdex(x * (\xi - xip)\text{^}2, n, 'l');$

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

$\mathbf{return}\ d$

$\mathbf{end\ if};$

$xiip := mod(l * ((a - \xi * xip)\text{^}2 - 4 * b * (\xi + xip)), n)$

$\mathbf{else}$

$d := igcdex(x * (\xi - xip)\text{^}2, n, 'l');$

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

$\mathbf{return}\ d$

```
        end if;
        xii := mod(l*((a - ξ*xip)^2 - 4*b*(ξ + xip)), n);
        d := igcdex(4*xip^3 + 4*a*xip + 4*b, n, 'l');
        if 1 < d then
            return d
        end if;
        xiip := mod(l*((xip^2 - a)^2 - 8*b*xip), n)
    end if;
    [x, xii, xiip]
end proc
```

10.3. Projektív reprezentáció.

```
> #
  # Doubling on an elliptic "curve" modulo n, where gcd(n,6)=1.
  # x, z are the coordinates of the point to double and a, b
  # are the parameters. The return value is a list, the
  # coordinates of the double of the point.
  #

  ellxzdou:=proc(x,z,a,b,n)
  [(x^2-a*z^2)^2-8*b*x*z^3 mod n,4*z*(x^3+a*x*z^2+b*z^3) mod n]
  end;
```

```
ellxzdou := proc(x, z, a, b, n)                                  (10.3.1)
    [mod((x^2 - a*z^2)^2 - 8*b*x*z^3, n), mod(4*z*(x^3
    + a*x*z^2 + b*z^3), n)]
end proc
```

```
> #
  # Calculation of the triple of coordinates
  # [[x,z],[x_2i,z_2i],[x_2i+1,z_2i+1]] if t=0
  # or [[x,z],[x_2i+1,z_2i+1],[x_2i+2,z_2i+2]] if t=1
  # from the triple of coordinates [[x,z],[x_i,z_i],[x_i+1,
  z_i+1]]
  # on an elliptic "curve" modulo n, where igcd(n,6)=1, and a,
  b
  # are the parameters. The return value is this list.
  #

  ellnextxz:=proc(L,t,a,b,n) local l,d,x,z,xi,zi,xip,zip,xii,
  zii,xiip,ziip;
  x:=L[1][1]; z:=L[1][2];
  xi:=L[2][1]; zi:=L[2][2]; xip:=L[3][1]; zip:=L[3][2];
  if t=0 then
    xii:=(xi^2-a*zi^2)^2-8*b*xi*zi^3 mod n;
```

```
    zii:=4*zi*(xi^3+a*xi*zi^2+b*zi^3) mod n;
    xiip:=z*((xi*xip-a*zi*zip)^2-4*b*zi*zip*(xi*zip+xip*zi))
  mod n;
    ziip:=x*(xip*zi-xi*zip)^2 mod n;
  else
    xii:=z*((xi*xip-a*zi*zip)^2-4*b*zi*zip*(xi*zip+xip*zi)) mod
  n;
    zii:=x*(xip*zi-xi*zip)^2 mod n;
    xiip:=(xip^2-a*zip^2)^2-8*b*xip*zip^3 mod n;
    ziip:=4*zip*(xip^3+a*xip*zip^2+b*zip^3) mod n;
  fi; [[x,z],[xii,zii],[xiip,ziip]] end;
```

$$ellnextxz := \mathbf{proc}(L, t, a, b, n) \tag{10.3.2}$$

local $l, d, x, z, \xi, zi, xip, zip, xii, zii, xiip, ziip$;
$x := L[1][1]$;
$z := L[1][2]$;
$\xi := L[2][1]$;
$zi := L[2][2]$;
$xip := L[3][1]$;
$zip := L[3][2]$;
if $t = 0$ **then**
$xii := mod((\xi\text{^}2 - a*zi\text{^}2)\text{^}2 - 8*b*\xi*zi\text{^}3, n)$;
$zii := mod(4*zi*(\xi\text{^}3 + a*\xi*zi\text{^}2 + b*zi\text{^}3), n)$;
$xiip := mod(z*((\xi*xip - a*zi*zip)\text{^}2 - 4*b*zi*zip*(\xi*zip + xip*zi)), n)$;
$ziip := mod(x*(xip*zi - \xi*zip)\text{^}2, n)$
else
$xii := mod(z*((\xi*xip - a*zi*zip)\text{^}2 - 4*b*zi*zip*(\xi*zip + xip*zi)), n)$;
$zii := mod(x*(xip*zi - \xi*zip)\text{^}2, n)$;
$xiip := mod((xip\text{^}2 - a*zip\text{^}2)\text{^}2 - 8*b*xip*zip\text{^}3, n)$;
$ziip := mod(4*zip*(xip\text{^}3 + a*xip*zip\text{^}2 + b*zip\text{^}3), n)$
end if;
$[[x, z], [xii, zii], [xiip, ziip]]$
end proc

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

  batchellmuls:=proc(L,k,n) local LL,LLL,LLLL,a,b,i,j,P,B,d,l;
```

```
if k=0 then return L fi; LL:=[];
for i to nops(L) do
  a:=L[i][2]; b:=L[i][3];
  P:=ellxzdou(L[i][1],1,a,b,n);
  P:=[[L[i][1],1],[L[i][1],1],P,L[i][2],L[i][3]];
  LL:=[op(LL),P];
od;
B:=convert(k,base,2);
for j from nops(B)-1 to 1 by -1 do
  LLL:=LL; LL:=[];
  for i to nops(LLL) do
    P:=LLL[i][1..3]; a:=LLL[i][4]; b:=LLL[i][5];
    P:=ellnextxz(P,B[j],a,b,n);
    LL:=[op(LL),[op(P),a,b]];
  od;
od; LLL:=[];
for i to nops(LL) do LLL:=[op(LLL),LL[i][2][2]] od;
LLL:=batchmodinv(LLL,n);
if type(LLL,integer) then
  if LLL<n then return LLL fi; LL:=[];
  for i to nops(L) do
    a:=L[i][2]; b:=L[i][3];
    P:=ellxzdou(L[i][1],1,a,b,n);
    d:=igcdex(P[2],n,'l');
    if 1<d then
      if d<n then return d else next fi;
    else
      P:=[[L[i][1],1],[L[i][1],1],[l*P[1] mod n,1]];
    fi;
    for j from nops(B)-1 to 1 by -1 do
      P:=ellnextxz(P,B[j],a,b,n);
      d:=igcdex(P[2][2],n,'l');
      if 1<d then
        if d<n then return d else break fi;
      else
        P[2][1]:=l*P[2][1]*l mod n; P[2][2]:=1;
      fi;
      d:=igcdex(P[2][2],n,'l');
      if 1<d then
        if d<n then return d else break fi;
      else
        P[3][1]:=l*P[3][1] mod n; P[3][2]:=1;
      fi;
    od;
    if d=n then next fi;
    LL:=[op(LL),[P[2][1],a,b]];
  od;
  LLLL:=LL;
```

```
    else
      LLLL:=[];
      for i to nops(LLL) do
        P:=LL[i][2][1]*LLL[i] mod n;
        a:=LL[i][4]; b:=LL[i][5];
        LLLL:=[op(LLLL),[P,a,b]];
      od;
    fi; LLLL; end;
```

```
batchellmuls := proc(L, k, n)                                   (10.3.3)
  local LL, LLL, LLLL, a, b, i, j, P, B, d, l;
  if k = 0 then
    return L
  end if;
  LL := [ ];
  for i to nops(L) do
    a := L[i][2];
    b := L[i][3];
    P := ellxzdou(L[i][1], 1, a, b, n);
    P := [[L[i][1], 1], [L[i][1], 1], P, L[i][2], L[i][3]];
    LL := [op(LL), P]
  end do;
  B := convert(k, base, 2);
  for j from nops(B) - 1 by -1 to 1 do
    LLL := LL;
    LL := [ ];
    for i to nops(LLL) do
      P := LLL[i][1..3];
      a := LLL[i][4];
      b := LLL[i][5];
      P := ellnextxz(P, B[j], a, b, n);
      LL := [op(LL), [op(P), a, b]]
    end do
  end do;
  LLL := [ ];
  for i to nops(LL) do
    LLL := [op(LLL), LL[i][2][2]]
  end do;
  LLL := batchmodinv(LLL, n);
  if type(LLL, integer) then
```

```
        if LLL < n then
            return LLL
        end if;
        LL := [ ];
        for i to nops(L) do
            a := L[i][2];
            b := L[i][3];
            P := ellxzdou(L[i][1], 1, a, b, n);
            d := igcdex(P[2], n, 'l');
            if 1 < d then
                if d < n then
                    return d
                else
                    next
                end if
            else
                P := [[L[i][1], 1], [L[i][1], 1], [mod(l*P[1], n), 1]]
            end if;
            for j from nops(B) - 1 by -1 to 1 do
                P := ellnextxz(P, B[j], a, b, n);
                d := igcdex(P[2][2], n, 'l');
                if 1 < d then
                    if d < n then
                        return d
                    else
                        break
                    end if
                else
                    P[2][1] := mod(l*P[2][1]*l, n);
                    P[2][2] := 1
                end if;
                d := igcdex(P[2][2], n, 'l');
                if 1 < d then
                    if d < n then
                        return d
                    else
                        break
                    end if
```

```
                else
                    P[3][1] := mod(l*P[3][1], n);
                    P[3][2] := 1
                end if
            end do;
            if d = n then
                next
            end if;
            LL := [op(LL), [P[2][1], a, b]]
        end do;
        LLLL := LL
    else
        LLLL := [ ];
        for i to nops(LLL) do
            P := mod(LL[i][2][1]*LLL[i], n);
            a := LL[i][4];
            b := LL[i][5];
            LLLL := [op(LLLL), [P, a, b]]
        end do
    end if;
    LLLL
end proc
```

```
> #
  # This procedure is elliptic curve method with projective
  # representation for factorization of n. It use K "curves"
  # parallel and only after all step of a multiplication is
  # done an igcd operation.  If no success,
  # backtracking is used for each curves separately.
  Unsuccessful
  # curves are "killed" from the list of curves.
  # The prime powers up to P are considered so that they
  # are not less then the bound B. The result is the list
  # of triples [x,a,b], where x is the first coordinate of
  # the multiple and a,b are the parameters of the curve.
  #

  batchellsplit:=proc(n,K,P,B) local e,d,p,L,i;
  L:=[];
  for i to K do
    p:=ellrand(n); if type(p,integer) then return p fi;
    L:=[op(L),[p[1],p[3],p[4]]];
  od;
```

```
   p:=2; e:=1; while 2^e<B do e:=e+1 od;
   while p<=P do
     while p^e>p*B and e>1 do e:=e-1 od;
     L:=batchellmuls(L,p^e,n);
     if type(L,integer) then return L fi;
     p:=nextprime(p);
   od; L; end;
```

$$batchellsplit := \mathbf{proc}(n, K, P, B) \tag{10.3.4}$$

```
 local e, d, p, L, i;
 L := [ ];
 for i to K do
     p := ellrand(n);
     if type(p, integer) then
         return p
     end if;
     L := [op(L), [p[1], p[3], p[4]]]
 end do;
 p := 2;
 e := 1;
 while 2^e < B do
     e := e + 1
 end do;
 while p <= P do
     while p*B < p^e and 1 < e do
         e := e - 1
     end do;
     L := batchellmuls(L, p^e, n);
     if type(L, integer) then
         return L
     end if;
     p := nextprime(p)
 end do;
 L
end proc
```

>

10.4. Második lépcső.

This is a calculation of the distribution function F of the size of the largest

prime factor of a random number.

```
> F2:=x->1+ln(x);
```

$$F2 := x \to 1 + \ln(x) \tag{10.4.1}$$

```
> F3:=x->F2(1/2)-int(F2(t/(1-t))/t,t=x..1/2);
```

$$F3 := x \to F2\left(\frac{1}{2}\right) - \int_x^{\frac{1}{2}} \frac{F2\left(\frac{t}{1-t}\right)}{t}\, dt \tag{10.4.2}$$

```
> F4:=x->F3(1/3)-int(F3(t/(1-t))/t,t=x..1/3);
```

$$F4 := x \to F3\left(\frac{1}{3}\right) - \int_x^{\frac{1}{3}} \frac{F3\left(\frac{t}{1-t}\right)}{t}\, dt \tag{10.4.3}$$

The next line would be the definition between $1/5$ and $1/4$, but too slow; instead we will use approximation.

```
> F5:=x->F4(1/4)-int(F4(t/(1-t))/t,t=x..1/4);
```

$$F5 := x \to F4\left(\frac{1}{4}\right) - \int_x^{\frac{1}{4}} \frac{F4\left(\frac{t}{1-t}\right)}{t}\, dt \tag{10.4.4}$$

```
> evalf(F4(0.25));
```

$$0.0049109254 \tag{10.4.5}$$

```
> Order:=10; series(F4(x),x=1/4);
```

$$Order := 10$$

$$1 - \ln(3) - \frac{1}{2}\ln(2)^2 - \mathrm{dilog}\left(\frac{3}{2}\right) \tag{10.4.6}$$

$$
\begin{aligned}
&+\ln(2)\ln(3)-\frac{1}{12}\pi^2-\int_{\frac{1}{4}}^{\frac{1}{3}}\frac{1-\ln(2)-\int_{\frac{t}{1-t}}^{\frac{1}{2}}\frac{1+\ln\left(\frac{t}{1-t}\right)}{t}\,dt}{t}\,dt\\
&+\left(4-4\ln(3)-2\ln(2)^2-4\,\mathrm{dilog}\left(\frac{3}{2}\right)\right.\\
&\left.+4\ln(2)\ln(3)-\frac{1}{3}\pi^2\right)\left(x-\frac{1}{4}\right)+\left(\frac{8}{3}-\frac{32}{3}\ln(2)+8\ln(3)\right.\\
&\left.+4\ln(2)^2+8\,\mathrm{dilog}\left(\frac{3}{2}\right)-8\ln(2)\ln(3)+\frac{2}{3}\pi^2\right)\left(x-\frac{1}{4}\right)^2+\left(\frac{320}{27}\right.\\
&\left.+\frac{1024}{27}\ln(2)-\frac{64}{3}\ln(3)-\frac{32}{3}\ln(2)^2-\frac{64}{3}\,\mathrm{dilog}\left(\frac{3}{2}\right)\right.\\
&\left.+\frac{64}{3}\ln(2)\ln(3)-\frac{16}{9}\pi^2\right)\left(x-\frac{1}{4}\right)^3+\left(-\frac{4160}{81}-\frac{11008}{81}\ln(2)\right.\\
&\left.+64\ln(3)+32\ln(2)^2+64\,\mathrm{dilog}\left(\frac{3}{2}\right)-64\ln(2)\ln(3)\right.\\
&\left.+\frac{16}{3}\pi^2\right)\left(x-\frac{1}{4}\right)^4+\left(\frac{65536}{135}\ln(2)\right.\\
&\left.+\frac{7168}{27}-\frac{1024}{5}\ln(3)-\frac{512}{5}\ln(2)^2-\frac{1024}{5}\,\mathrm{dilog}\left(\frac{3}{2}\right)\right.\\
&\left.+\frac{1024}{5}\ln(2)\ln(3)-\frac{256}{15}\pi^2\right)\left(x-\frac{1}{4}\right)^5\\
&+\left(-\frac{800768}{729}-\frac{6397952}{3645}\ln(2)+\frac{2048}{3}\ln(3)+\frac{1024}{3}\ln(2)^2\right.\\
&\left.+\frac{2048}{3}\,\mathrm{dilog}\left(\frac{3}{2}\right)-\frac{2048}{3}\ln(2)\ln(3)+\frac{512}{9}\pi^2\right)\left(x-\frac{1}{4}\right)^6\\
&+\left(\frac{354992128}{76545}+\frac{490471424}{76545}\ln(2)-\frac{16384}{7}\ln(3)-\frac{8192}{7}\ln(2)^2\right.\\
&\left.-\frac{16384}{7}\,\mathrm{dilog}\left(\frac{3}{2}\right)+\frac{16384}{7}\ln(2)\ln(3)-\frac{4096}{21}\pi^2\right)\left(x-\frac{1}{4}\right)^7
\end{aligned}
$$

$$+\left(-\frac{1422598144}{76545}-\frac{1806270464}{76545}\ln(2)+8192\ln(3)+4096\ln(2)^2\right.$$
$$\left.+8192\,\mathrm{dilog}\left(\frac{3}{2}\right)-8192\ln(2)\ln(3)+\frac{2048}{3}\pi^2\right)\left(x-\frac{1}{4}\right)^8$$
$$+\left(\frac{155127119872}{2066715}\right.$$
$$+\frac{180925497344}{2066715}\ln(2)-\frac{262144}{9}\ln(3)-\frac{131072}{9}\ln(2)^2$$
$$\left.-\frac{262144}{9}\,\mathrm{dilog}\left(\frac{3}{2}\right)+\frac{262144}{9}\ln(2)\ln(3)-\frac{65536}{27}\pi^2\right)\left(x-\frac{1}{4}\right)^9$$
$$+\mathrm{O}\left(\left(x-\frac{1}{4}\right)^{10}\right)$$

```
> F4s:=evalf(%);
```

$$F4s := 0.00491092536+0.194433553\,(x-0.2500000000)+2.884229635\,(x-0.2500000000)^2+17.84374609\,(x-0.2500000000)^3-84.66851915\,(x-0.2500000000)^4+407.1260619\,(x-0.2500000000)^5-1665.622940\,(x-0.2500000000)^6+6852.317002\,(x-0.2500000000)^7-27147.86069\,(x-0.2500000000)^8+1.080283354\,10^5\,(x-0.2500000000)^9+\mathrm{O}\left((x-0.2500000000)^{10}\right) \tag{10.4.7}$$

```
> F4p:=convert(F4s,polynom); F4as:=subs(x=t/(1-t),F4p)/t;
```

$$F4p := -0.04369746289+0.194433553\,x+2.884229635\,(x-0.2500000000)^2+17.84374609\,(x-0.2500000000)^3-84.66851915\,(x-0.2500000000)^4+407.1260619\,(x-0.2500000000)^5-1665.622940\,(x-0.2500000000)^6+6852.317002\,(x-0.2500000000)^7-27147.86069\,(x-0.2500000000)^8$$

$$+1.080283354\,10^{5}\,(x-0.2500000000)^{9}$$

$$F4as := \frac{1}{t}\left(-0.04369746289+\frac{0.194433553\,t}{1-t}\right. \tag{10.4.8}$$
$$+2.884229635\left(\frac{t}{1-t}-0.2500000000\right)^{2}$$
$$+17.84374609\left(\frac{t}{1-t}-0.2500000000\right)^{3}$$
$$-84.66851915\left(\frac{t}{1-t}-0.2500000000\right)^{4}$$
$$+407.1260619\left(\frac{t}{1-t}-0.2500000000\right)^{5}$$
$$-1665.622940\left(\frac{t}{1-t}-0.2500000000\right)^{6}$$
$$+6852.317002\left(\frac{t}{1-t}-0.2500000000\right)^{7}$$
$$-27147.86069\left(\frac{t}{1-t}-0.2500000000\right)^{8}$$
$$\left.+1.080283354\,10^{5}\left(\frac{t}{1-t}-0.2500000000\right)^{9}\right)$$

```
> F5a:=y->F4(1/4)-int(F4as,t=y..1/4);
```

$$F5a := y \to F4\left(\frac{1}{4}\right)-\int_{y}^{\frac{1}{4}} F4as\,\mathrm{d}t \tag{10.4.9}$$

```
> evalf(F5a(0.20));
```

$$0.00035463226 \tag{10.4.10}$$

```
> F:=proc(x) if x>1. then 1. elif x>=1/2. then F2(x) elif x>=
  1/3.
  then F3(x) elif x>=1/4. then F4(x) elif x>=1/5. then F5a(x)
  else 0. fi; end;
```

$F := \mathbf{proc}(x)$ (10.4.11)

```
if 1. < x then
    1.
elif 1 / 2. <= x then
    F2(x)
elif 1 / 3. <= x then
    F3(x)
```

```
    elif 1 / 4. <= x then
        F4(x)
    elif 1 / 5. <= x then
        F5a(x)
    else
        0.
    end if
end proc
```

```
> F(0.22);
```

$$0.9963042122 - \ln(3) - \frac{1}{2}\ln(2)^2 - \mathrm{dilog}\left(\frac{3}{2}\right) + \ln(2)\ln(3) - \frac{1}{12}\pi^2 - \int_{\frac{1}{4}}^{\frac{1}{3}} \frac{1 - \ln(2) - \int_{\frac{t}{1-t}}^{\frac{1}{2}} \frac{1 + \ln\left(\frac{t}{1-t}\right)}{t}\,dt}{t}\,dt \tag{10.4.12}$$

```
> evalf(%);
```

$$0.00121513756 \tag{10.4.13}$$

Now the definition of F is finished. We start to count probability ENI
and ENII to does not succed with working N elliptic curves, and using
step I or step II also. We have to give the following bounds:
B0 is the bound for prime powers; B1 is the bound for primes in step I;
B2 is the bound for primes in step II; finally, B is the bound for factors;
the conditions B0>=B1 and B2>>B1 are supposed.
The probabilities to does not succeed with one elliptic curve is also given,
these are EI and EII, respectively. Two additional probabilities also given:
E0 is an upper bound for probability that error is caused by too small B0,
and E1 is an upper bound the probability that error is cause by too small B0
with respect to B1. These shoud have to keep below 1-EI or 1-EII, depending
on whether we plane to use step I only or step II, too.
Upper bounds for work also given back: these are calculated for one curve
as the number of additions on the curve, and are the following:
W0 for part of step I depending only on B0;
W1 for part of step I depending mainly on B1;
W2 for step II and depending mainly on B2.

```
> 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*] (10.4.14)

```
> B0:=10.^5; B1:=10.^4; B2:=10.^6; B:=10.^20; N:=32;
```

$$B0 := 1.00000\,10^{5}$$

$$B1 := 10000.$$

$$B2 := 1.000000\,10^{6}$$

$$B := 1.000000000\,10^{20}$$

$$N := 32 \tag{10.4.15}$$

```
> E0:=proc() evalf(pi(floor(sqrt(B0)))/B0); end; E0();
```

$$E0 := \mathbf{proc}()\ evalf(numtheory:-\pi(\mathrm{floor}(\mathrm{sqrt}(B0)))\ /\ B0)\ \mathbf{end\ proc}$$

$$0.0006500000000 \tag{10.4.16}$$

```
> W0:=proc() 2*floor(log[2.](B0))*pi(floor(sqrt(B0)))*N; end;
  W0();
```

$$W0 := \mathbf{proc}()$$
$$2 * \mathrm{floor}(\log[2.](B0)) * numtheory:-\pi(\mathrm{floor}(\mathrm{sqrt}(B0))) *$$
$$N$$
$$\mathbf{end\ proc}$$

$$66560 \tag{10.4.17}$$

```
> E1:=proc() local s,p; p:=nextprime(floor(sqrt(B0))); s:=0.;
  while p<B1 do s:=s+1/p^2; p:=nextprime(p); od; s; end;
```

$$E1 := \mathbf{proc}() \tag{10.4.18}$$
$$\mathbf{local}\ s, p;$$
$$p := nextprime(\mathrm{floor}(\mathrm{sqrt}(B0)));$$
$$s := 0.;$$
$$\mathbf{while}\ p < B1\ \mathbf{do}$$
$$s := s + 1\ /\ p\hat{}2;$$
$$p := nextprime(p)$$
$$\mathbf{end\ do};$$
$$s$$
$$\mathbf{end\ proc}$$

```
> E1();
```

$$0.0004499193528 \tag{10.4.19}$$

```
> W1:=proc() local s,p; p:=nextprime(floor(sqrt(B0))); s:=0;
  while p<B1 do s:=s+floor(log[2.](p)); p:=nextprime(p); od; 2*
  N*s; end;
```

$$W1 := \mathbf{proc}()\ \mathbf{local}\ s, p;\ p := nextprime(\mathrm{floor}(\mathrm{sqrt}(B0)));\ s := 0;\ \mathbf{while}\ p < B1\ \mathbf{do}\ s := s + \mathrm{floor}(\log[2.](p));\ p := nextprime(p)\ \mathbf{end\ do};\ 2 * N * s\ \mathbf{end\ proc} \tag{10.4.20}$$

```
> W1();
```

$$850368 \tag{10.4.21}$$

```
> EI:=proc() evalf(1.-F(log[B](B1))); end;
  EI(); 1-EI(); ENI:=proc() EI()^N; end; ENI();
```

$$EI := \mathbf{proc}()\ evalf(1. - F(\log[B](B1)))\ \mathbf{end\ proc}$$

$$0.9996453676$$

$$0.0003546324$$

$$ENI := \mathbf{proc}()\ EI()\hat{}N\ \mathbf{end\ proc}$$

$$0.9887139216 \tag{10.4.22}$$

```
> EII:=proc() evalf(1.-exp(gamma)*log[B](B1)*F(log[B](B2)));
  end;
  EII(); 1-EII(); ENII:=proc() EII()^N; end; ENII();
```

$$EII := \mathbf{proc}()\ evalf(1. - \exp(\gamma) * \log[B](B1) * F(\log[B](B2)))\ \mathbf{end\ proc}$$

$$0.9915754783$$

$$0.0084245217$$

$$ENII := \mathbf{proc}()\ EII()\hat{}N\ \mathbf{end\ proc}$$

$$0.7628249680 \tag{10.4.23}$$

```
> W2:=proc() ceil(pi(floor(B2))-pi(floor(B1)))*N; end; W2();
```

$$W2 := \mathbf{proc}()\ \mathrm{ceil}(numtheory\!:\!-\pi(\mathrm{floor}(B2)) - numtheory\!:\!-\pi(\mathrm{floor}(B1))) * N\ \mathbf{end\ proc}$$

$$2472608 \tag{10.4.24}$$

```
> N:=64; B0:=10.^6; B1:=10.^5; B2:=10.^7; B:=10.^25;
  E0(); E1(); EI(); 1-EI(); ENI(); EII(); 1-EII(); ENII();
  W0(); W1(); W2();
```

$$N := 64$$

$$B0 := 1.000000\,10^{6}$$
$$B1 := 1.00000\,10^{5}$$
$$B2 := 1.0000000\,10^{7}$$
$$B := 1.000000000\,10^{25}$$
$$0.0001680000000$$
$$0.0001261875417$$
$$0.9996453676$$
$$0.0003546324$$
$$0.9775552187$$
$$0.9950978649$$
$$0.0049021351$$
$$0.7301479030$$
$$408576$$
$$17655424$$
$$41919168 \qquad (10.4.25)$$

```
> #
  # This procedure is the second step of elliptic curve method
  for
  # factorization. The list of "curves" is L, the table of
  doubles
  # has size N and primes greater then m but not greater then M
  # are considered. The result is the last list or a factor of
  n.
  #

  batchellsplit2:=proc(L,n::posint,N::posint,m::posint,
  M::posint)
  local y,i,j,E,P,PP,p,pp,d,LL,LLL,a,b;
  LL:=[];
  for i to nops(L) do
    p:=L[i];
    y:=msqrt(L[i][1],n);
    if y=FAIL then next fi;
    LL:=[op(LL),[L[i][1],y,1,L[i][2],L[i][3]]];
  od;
  E:=Array(1..nops(LL),1..N);
  for i to nops(LL) do
    P:=LL[i][1..3];
    a:=LL[i][4]; b:=LL[i][5];
    PP:=elldou(P,a,b,n);
    if type(PP,integer) then return PP fi;
    E[i,1]:=PP;
    for j from 2 to N do
```

```
      PP:=elladd(E[i,j-1],PP,a,b,n);
      if type(PP,integer) then return PP fi;
      E[i,1]:=PP;
    od;
  od;
  p:=nextprime(m);
  for i to nops(LL) do
    P:=LL[i][1..3];
    a:=LL[i][4]; b:=LL[i][5];
    PP:=ellmul(P,p,a,b,n);
    if type(PP,integer) then return PP fi;
    LLL[i]:=[op(PP),a,b];
  od;
  pp:=nextprime(p);
  while pp<M do
  d:=pp-p; p:=pp;
    if d<=2*N then
      for i to nops(LL) do
        PP:=LLL[i][1..3];
        a:=LL[i][4]; b:=LL[i][5];
        PP:=elladd(PP,E[i,d/2],a,b,n);
        if type(PP,integer) then return PP fi;
        LLL[i]:=[op(PP),a,b];
      od;
    else
      for i to nops(LL) do
        P:=LL[i][1..3];
        a:=LL[i][4]; b:=LL[i][5];
        PP:=ellmul(P,d,a,b,n);
        if type(PP,integer) then return PP fi;
        PP:=elladd(PP,LLL[i][1..3],a,b,n);
        if type(PP,integer) then return PP fi;
        LLL[i]:=[op(PP),a,b];
      od;
    fi;
    pp:=nextprime(p);
  od; LLL; end;
```

$$batchellsplit2 := \mathbf{proc}(L, n::posint, N::posint, m::posint, M::posint) \tag{10.4.26}$$

```
  local y, i, j, E, P, PP, p, pp, d, LL, LLL, a, b;
  LL := [ ];
  for i to nops(L) do
    p := L[i];
    y := msqrt(L[i][1], n);
    if y = FAIL then
      next
    end if;
```

```
        LL := [op(LL), [L[i][1], y, 1, L[i][2], L[i][3]]]
    end do;
    E := Array(1 ..nops(LL), 1 ..N);
    for i to nops(LL) do
        P := LL[i][1 ..3];
        a := LL[i][4];
        b := LL[i][5];
        PP := elldou(P, a, b, n);
        if type(PP, integer) then
            return PP
        end if;
        E[i, 1] := PP;
        for j from 2 to N do
            PP := elladd(E[i, j - 1], PP, a, b, n);
            if type(PP, integer) then
                return PP
            end if;
            E[i, 1] := PP
        end do
    end do;
    p := nextprime(m);
    for i to nops(LL) do
        P := LL[i][1 ..3];
        a := LL[i][4];
        b := LL[i][5];
        PP := ellmul(P, p, a, b, n);
        if type(PP, integer) then
            return PP
        end if;
        LLL[i] := [op(PP), a, b]
    end do;
    pp := nextprime(p);
    while pp < M do
        d := pp - p;
        p := pp;
        if d <= 2 * N then
            for i to nops(LL) do
                PP := LLL[i][1 ..3];
```

```
                a := LL[i][4];
                b := LL[i][5];
                PP := elladd(PP, E[i, 1 / 2 * d], a, b, n);
                if type(PP, integer) then
                    return PP
                end if;
                LLL[i] := [op(PP), a, b]
            end do
        else
            for i to nops(LL) do
                P := LL[i][1..3];
                a := LL[i][4];
                b := LL[i][5];
                PP := ellmul(P, d, a, b, n);
                if type(PP, integer) then
                    return PP
                end if;
                PP := elladd(PP, LLL[i][1..3], a, b, n);
                if type(PP, integer) then
                    return PP
                end if;
                LLL[i] := [op(PP), a, b]
            end do
        end if;
        pp := nextprime(p)
    end do;
    LLL
end proc
```

▶ 11. Prímteszt elliptikus görbékkel

▶ 12. Polinomfaktorizálás

▶ 13. Az AKS-teszt

▶ 14. A szita módszerek alapjai