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

```
> 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*] (4.1)

▼ 4.1. Definíció.

```
> a:=-3; b:=2; for n to 10 do [(a^n-b^n)/(a-b),a^n+b^n]; od;
```

$$a := -3$$

$$b := 2$$

$$[1, -1]$$

$$[-1, 13]$$

$$[7, -19]$$

$$[-13, 97]$$

$$[55, -211]$$

$$[-133, 793]$$

$$[463, -2059]$$

$$[-1261, 6817]$$
$$[4039, -19171]$$
$$[-11605, 60073] \tag{4.1.1}$$

▶ 4.2. Megjegyzés.

▼ 4.3. Példa.

```
> a:=(1+sqrt(5))/2; b:=(1-sqrt(5))/2; for n to 10 do [expand(
  (a^n-b^n)/(a-b)),expand(a^n+b^n)]; od;
```

$$a := \frac{1}{2} + \frac{1}{2}\sqrt{5}$$
$$b := \frac{1}{2} - \frac{1}{2}\sqrt{5}$$
$$[1, 1]$$
$$[1, 3]$$
$$[2, 4]$$
$$[3, 7]$$
$$[5, 11]$$
$$[8, 18]$$
$$[13, 29]$$
$$[21, 47]$$
$$[34, 76]$$
$$[55, 123] \tag{4.3.1}$$

▼ 4.4. Rekurzió számoláshoz.

```
> #
  # This procedure calculate the Lucas sequence terms
  # [U[n],V[n],U[n+1],V[n+1]]. If the parameter N also given
  # then the calculations are done mod N. Here U[n]=a^n-b^n)/(a
  -b)
  # and V[n]=a^n+b^n, where a,b are the roots of x^2-Px+Q=0.
  #

  lucasUV:=proc(n,P,Q,N) local L,B,i,Qk;
  B:=convert(n,base,2);
  Qk:=1;
  if nargs=3 then
```

```
    L:=[0,2,1,P];
    for i from nops(B) to 1 by -1 do
      if B[i]=0 then
        L:=[L[1]*L[2],L[2]^2-2*Qk,L[3]*L[2]-Qk,L[4]*L[2]-P*Qk];
        Qk:=Qk^2;
      else
        L:=[L[3]*L[2]-Qk,L[4]*L[2]-P*Qk,L[3]*L[4],L[4]^2-2*Q*
  Qk];
        Qk:=Qk^2*Q;
      fi;
    od;
  else
    L:=[0,2 mod N,1,P mod N];
    for i from nops(B) to 1 by -1 do
      if B[i]=0 then
        L:=[L[1]*L[2] mod N,L[2]&^2-2*Qk mod N,\
        L[3]*L[2]-Qk mod N,L[4]*L[2]-P*Qk mod N];
        Qk:=Qk&^2 mod N;
      else
        L:=[L[3]*L[2]-Qk mod N,L[4]*L[2]-P*Qk mod N,\
        L[3]*L[4] mod N,L[4]&^2-2*Q*Qk mod N];
        Qk:=Qk&^2*Q mod N;
      fi;
    od;
  fi; L end;
```

$$lucasUV := \mathbf{proc}(n, P, Q, N) \tag{4.4.1}$$

local L, B, i, Qk;

$B := convert(n, base, 2)$;

$Qk := 1$;

if $nargs = 3$ **then**

$L := [0, 2, 1, P]$;

for i **from** $nops(B)$ **by** -1 **to** 1 **do**

if $B[i] = 0$ **then**

$L := [L[1] * L[2], L[2]\text{^}2 - 2 * Qk, L[3] * L[2] - Qk, L[4] * L[2]$
$- P * Qk]$;

$Qk := Qk\text{^}2$

else

$L := [L[3] * L[2] - Qk, L[4] * L[2] - P * Qk, L[3] * L[4], L[4]\text{^}2$
$- 2 * Q * Qk]$;

$Qk := Qk\text{^}2 * Q$

end if

```
        end do
    else
        L := [0, mod(2, N), 1, mod(P, N)];
        for i from nops(B) by -1 to 1 do
            if B[i] = 0 then
                L := [mod(L[1]*L[2], N), mod(L[2] &^ 2 - 2*Qk,
                N), mod(L[3]*L[2] - Qk, N), mod(L[4]*L[2] - P*Qk, N)];
                Qk := mod(Qk &^ 2, N)
            else
                L := [mod(L[3]*L[2] - Qk, N), mod(L[4]*L[2] - P*Qk,
                N), mod(L[3]*L[4], N), mod(L[4] &^ 2 - 2*Q*Qk, N)];
                Qk := mod(Qk &^ 2*Q, N)
            end if
        end do
    end if;
    L
end proc
```

```
> #
  # This procedure calculate the Lucas sequence terms
  # [V[n],V[n+1]]. If the parameter N also given
  # then the calculations are done mod N. Here V[n]=a^n+b^n,
  # where a,b are the roots of x^2-Px+Q=0.
  #

  lucasV:=proc(n,P,Q,N) local L,B,i,Qk;
  B:=convert(n,base,2);
  Qk:=1;
  if nargs=3 then L:=[2,P];
    for i from nops(B) to 1 by -1 do
      if B[i]=0 then
        L:=[L[1]^2-2*Qk,L[2]*L[1]-P*Qk]; Qk:=Qk^2;
      else
        L:=[L[2]*L[1]-P*Qk,L[2]^2-2*Q*Qk]; Qk:=Qk^2*Q;
      fi;
    od;
  else L:=[2 mod N,P mod N];
    for i from nops(B) to 1 by -1 do
      if B[i]=0 then
        L:=[L[1]&^2-2*Qk mod N,L[2]*L[1]-P*Qk mod N]; Qk:=Qk&^2
  mod N;
```

```
      else
        L:=[L[2]*L[1]-P*Qk mod N,L[2]&^2-2*Q*Qk mod N]; Qk:=
   Qk&^2*Q mod N;
      fi;
    od;
  fi; L end;
```

$$lucasV := \mathbf{proc}(n, P, Q, N) \tag{4.4.2}$$

```
local L, B, i, Qk;
B := convert(n, base, 2);
Qk := 1;
if nargs = 3 then
    L := [2, P];
    for i from nops(B) by -1 to 1 do
        if B[i] = 0 then
            L := [L[1]^2 - 2 * Qk, L[1] * L[2] - P * Qk];
            Qk := Qk^2
        else
            L := [L[1] * L[2] - P * Qk, L[2]^2
            - 2 * Q * Qk];
            Qk := Qk^2 * Q
        end if
    end do
else
    L := [mod(2, N), mod(P, N)];
    for i from nops(B) by -1 to 1 do
        if B[i] = 0 then
            L := [mod(L[1] &^ 2 - 2 * Qk,
            N), mod(L[1] * L[2] - P * Qk, N)];
            Qk := mod(Qk &^ 2, N)
        else
            L := [mod(L[1] * L[2] - P * Qk,
            N), mod(L[2] &^ 2 - 2 * Q * Qk, N)];
            Qk := mod(Qk &^ 2 * Q, N)
        end if
```

```
        end do
    end if;
    L
end proc
```

▶ 4.5. Feladat.

▶ 4.6. Megjegyzés.

▶ 4.7. Definíció.

▶ 4.8. Tétel.

▼ 4.9. Feladat.

```
> #
  # This procedure do primality test for the odd number n
  # using the set S of all factors of n+1 and the Lucas
  # sequence corresponding to the roots of x^2-Px+1.
  #

  naiveplustest:=proc(n::posint,S::list(posint),P::integer)
  local D,q,L;
  D:=P^2-4; modp(D,4); if %=2 or %=3 then RETURN(FAIL) fi;
  igcd(2*D,n);
  if %<>1 then
    if %<n then RETURN(false) else RETURN(FAIL) fi;
  fi;
  if jacobi(D,n)<>-1 then RETURN(FAIL) fi;
  L:=lucasUV(n+1,P,1,n);
  if L[1]<>0 or L[2]<>2 then RETURN(false) fi;
  for q in S do
    L:=lucasUV((n+1)/q,P,1,n);
    if L[1]=0 and L[2]=2 then RETURN(FAIL) fi;
  od; true end;
```

```
naiveplustest := proc(n::posint, S::(list(posint)), P::integer)                (4.9.1)
    local D, q, L;
    D := P^2 - 4;
    modp(D, 4);
    if % = 2 or % = 3 then
        RETURN(FAIL)
```

```
    end if;
    igcd(2*D, n);
    if % <> 1 then
        if % < n then
            RETURN(false)
        else
            RETURN(FAIL)
        end if
    end if;
    if numtheory:-jacobi(D, n) <> -1 then
        RETURN(FAIL)
    end if;
    L := lucasUV(n+1, P, 1, n);
    if L[1] <> 0 or L[2] <> 2 then
        RETURN(false)
    end if;
    for q in S do
        L := lucasUV((n+1)/q, P, 1, n);
        if L[1] = 0 and L[2] = 2 then
            RETURN(FAIL)
        end if
    end do;
    true
end proc
```

```
> naiveplustest(7,[2],3);
```

$$true \tag{4.9.2}$$

```
> #
  # This procedure do primality test for the odd number n
  # using the set S of all factors of n+1.  Try all Lucas
  # sequences  with Q=1 and P in interval II.
  #

  naiveplustests:=proc(n::posint,S::list(posint),II::range
  (integer))
  local r,P;
  for P from op(1,II) to op(2,II) do
```

```
    r:=naiveplustest(n,S,P);
    if r<>FAIL then RETURN(r) fi;
  od; FAIL end;
```

$$naiveplustests := \mathbf{proc}(n::posint,\ S::(list(posint)),\ II::(range(integer))) \tag{4.9.3}$$

```
  local r, P,
  for P from op(1, II) to op(2, II) do
      r := naiveplustest(n, S, P);
      if r <> FAIL then
          RETURN(r)
      end if
  end do;
  FAIL
end proc
```

```
> naiveplustests(7,[2],1..2); naiveplustests(7,[2],1..3);
```

$$FAIL$$

$$true \tag{4.9.4}$$

▶ 4.10. Lucas-típusú teszt.

▼ 4.11. Feladat.

```
> #
  # This procedure do primality test for the odd number n
  # using a large enough set S of factors of n+1 and the Lucas
  # sequence corresponding to the roots of x^2-Px+1.
  #

  plustest:=proc(n::posint,S::list(posint),P::integer)
  local D,q,L; if n=1 then RETURN(false) fi;
  D:=P^2-4; modp(D,4); if %=2 or %=3 then RETURN(FAIL) fi;
  igcd(2*D,n);
  if %<>1 then
    if %<n then RETURN(false) else RETURN(FAIL) fi;
  fi;
  if jacobi(D,n)<>-1 then RETURN(FAIL) fi;
  L:=lucasUV(n+1,P,1,n);
  if L[1]<>0 or L[2]<>2 then RETURN(false) fi;
  for q in S do
    L:=lucasUV((n+1)/q,P,1,n);
    igcd(L[2]-2,L[1],n);
    if %<>1 then
```

```
      if %<n then RETURN(false) else RETURN(FAIL) fi;
    fi;
  od; true end;
```

```
plustest := proc(n::posint, S::(list(posint)), P::integer)                    (4.11.1)
    local D, q, L;
    if n = 1 then
        RETURN(false)
    end if;
    D := P^2 - 4;
    modp(D, 4);
    if % = 2 or % = 3 then
        RETURN(FAIL)
    end if;
    igcd(2*D, n);
    if % <> 1 then
        if % < n then
            RETURN(false)
        else
            RETURN(FAIL)
        end if
    end if;
    if numtheory:-jacobi(D, n) <> -1 then
        RETURN(FAIL)
    end if;
    L := lucasUV(n + 1, P, 1, n);
    if L[1] <> 0 or L[2] <> 2 then
        RETURN(false)
    end if;
    for q in S do
        L := lucasUV((n + 1) / q, P, 1, n);
        igcd(L[2] - 2, L[1], n);
        if % <> 1 then
            if % < n then
```

```
            RETURN(false)
        else
            RETURN(FAIL)
        end if
    end if
  end do;
  true
end proc
```

```
> plustest(7,[2],3);
```

$$true \tag{4.11.2}$$

```
> #
  # This procedure do primality test for the odd number n
  # using a large enough set S of factors of n+1. Try all Lucas
  # sequences with Q=1 and P in interval II.
  #

  plustests:=proc(n::posint,S::list(posint),II::range(integer))
  local r,P;
  for P from op(1,II) to op(2,II) do
    r:=plustest(n,S,P);
    if r<>FAIL then RETURN(r) fi;
  od; FAIL end;
```

$$plustests := \mathbf{proc}(n::posint,\ S::(list(posint)),\ II::(range(integer))) \tag{4.11.3}$$

```
  local r, P,
  for P from op(1, II) to op(2, II) do
      r := plustest(n, S, P);
      if r <> FAIL then
          RETURN(r)
      end if
  end do;
  FAIL
end proc
```

```
> plustests(7,[2],1..2); plustests(7,[2],1..3);
```

$$FAIL$$

$$true \tag{4.11.4}$$

4.12. Feladat.

```
> #
  # This procedure do primality test for the number h*2^n-1
  # with n>1 and odd h<2^n using the Lucas sequence
  corresponding
  # to the roots of x^2-Px+1.
  #

  specplustest:=proc(h::posint,n::posint,P::integer) local D,L,
  N;
  if h>=2^n then error "first parameter is too large",h fi;
  if type(h,even) then error "first parameter have to be odd",h
  fi;
  if n<2 then error "second parameter is too small",h fi;
  N:=h*2^n-1;
  D:=P^2-4; modp(D,4); if %=2 or %=3 then RETURN(FAIL) fi;
  igcd(2*D,N);
  if %<>1 then
    if %<n then RETURN(false) else RETURN(FAIL) fi;
  fi;
  if jacobi(D,N)<>-1 then RETURN(FAIL) fi;
  L:=lucasUV(N+1,P,1,N);
  if L[1]<>0 or L[2]<>2 then RETURN(false) fi;
  L:=lucasUV((N+1)/2,P,1,N);
  if L[1]<>0 then RETURN(false) fi;
  if L[2]<>N-2 then
      if L[2]=2 then RETURN(FAIL) else RETURN(false) fi;
  fi; true end;
```

$$specplustest := \mathbf{proc}(h::posint,\ n::posint,\ P::integer) \tag{4.12.1}$$

local D, L, N;

if $2\hat{}n <= h$ **then**

error "first parameter is too large", h

end if;

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

error "first parameter have to be odd", h

end if;

if $n < 2$ **then**

error "second parameter is too small", h

end if;

$N := h*2\hat{}n - 1$;

D := $P\hat{}2 - 4$;

$modp(\mathrm{D}, 4)$;

```
    if % = 2 or % = 3 then
        RETURN(FAIL)
    end if;
    igcd(2*D, N);
    if % <> 1 then
        if % < n then
            RETURN(false)
        else
            RETURN(FAIL)
        end if
    end if;
    if numtheory:-jacobi(D, N) <> -1 then
        RETURN(FAIL)
    end if;
    L := lucasUV(N+1, P, 1, N);
    if L[1] <> 0 or L[2] <> 2 then
        RETURN(false)
    end if;
    L := lucasUV(1 / 2*N+1 / 2, P, 1, N);
    if L[1] <> 0 then
        RETURN(false)
    end if;
    if L[2] <> N - 2 then
        if L[2] = 2 then
            RETURN(FAIL)
        else
            RETURN(false)
        end if
    end if;
    true
end proc
```

```
> specplustest(1,3,3);
```

$$true \tag{4.12.2}$$

```
> #
  # This procedure do primality test for the number h*2^n-1
  # with n>1 and odd h<2^n. Try all Lucas sequences  with Q=1
  # and P in interval II.
  #

  specplustests:=proc(h::posint,n::posint,II::range(integer))
  local r,P;
  if h>=2^n then error "first parameter is too large",h fi;
  if type(h,even) then error "first parameter have to be odd",h
  fi;
  if n<2 then error "second parameter is too small",h fi;
  for P from op(1,II) to op(2,II) do
    r:=specplustest(h,n,P);
    if r<>FAIL then RETURN(r) fi;
  od; FAIL end;
```

$$specplustests := \mathbf{proc}(h::posint,\ n::posint,\ II::(range(integer)))$$ (4.12.3)

local r, P;

if $2\hat{}n <= h$ **then**

error "first parameter is too large", h

end if;

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

error "first parameter have to be odd", h

end if;

if $n < 2$ **then**

error "second parameter is too small", h

end if;

for P **from** $op(1, II)$ **to** $op(2, II)$ **do**

$r := specplustest(h, n, P)$;

if $r <> FAIL$ **then**

$RETURN(r)$

end if

end do;

$FAIL$

end proc

```
> specplustests(1,3,1..2); specplustests(1,3,1..3);
```

$$FAIL$$

$$true$$ (4.12.4)

▶ 4.13. Számtestek.

▶ 4.14. Tétel.

▶ 4.15. Riesel–teszt.

▶ 4.16. Következmény.

▼ 4.17. Feladat.

```
> #
  # This procedure calculate a^h+a^(-h) for a unit a=r+s*sqrt
  (D)
  # with norm 1. Here r,s rational numbers, D must be a square
  # free integer. All computations are done mod N.
  #

  rieselsetup:=proc(r,s,D,h,N) local P,a;
  a:=r+s*sqrt(D);
  P:=r+s*sqrt(D)+(r-s*sqrt(D))/(r^2-s^2*D);
  if not type(P,integer) then ERROR(a+1/a, `not an integer`)
  fi;
  lucasV(h,P,1,N)[1]; end;
```

$$rieselsetup := \mathbf{proc}(r, s, \mathrm{D}, h, N) \tag{4.17.1}$$

$$\mathbf{local}\ P, a;$$

$$a := r + s * \mathrm{sqrt}(\mathrm{D});$$

$$P := r + s * \mathrm{sqrt}(\mathrm{D}) + (r - s * \mathrm{sqrt}(\mathrm{D})) / (r\text{^}2 - s\text{^}2 * \mathrm{D});$$

$$\mathbf{if\ not}\ type(P, integer)\ \mathbf{then}$$

$$ERROR(a + 1 / a, not\ an\ integer)$$

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

$$lucasV(h, P, 1, N)[1]$$

$$\mathbf{end\ proc}$$

▼ 4.18. Feladat.

```
> #
  # A procedure above for primality testing of h*2^n-1
  # with n>1 and odd h<2^n for which h not divisible by 3.
  #

  rieselsqrt3:=proc(h::posint,n::posint) local i,v,N;
```

```
   if h>=2^n then error "first parameter is too large",h fi;
   if type(h,even) then error "first parameter have to be odd",h
   fi;
   if modp(h,3)=0 then error "first parameter is a multiple of
   3",h fi;
   if n<2 then error "second parameter is too small",h fi;
   N:=h*2^n-1; if modp(N,3)=0 then return false fi;
   v:=rieselsetup(2,1,3,h,N);
   for i to n-2 do v:=v^2-2 mod N od;
   evalb(v=0); end;
```

$$rieselsqrt3 := \mathbf{proc}(h::posint,\ n::posint) \tag{4.18.1}$$

```
local i, v, N;
if 2^n <= h then
    error "first parameter is too large", h
end if;
if type(h, even) then
    error "first parameter have to be odd", h
end if;
if modp(h, 3) = 0 then
    error "first parameter is a multiple of 3", h
end if;
if n < 2 then
    error "second parameter is too small", h
end if;
N := h*2^n - 1;
if modp(N, 3) = 0 then
    return false
end if;
v := rieselsetup(2, 1, 3, h, N);
for i to n - 2 do
    v := mod(v^2 - 2, N)
end do;
evalb(v = 0)
end proc
```

```
> #
  # Some tests.
```

```
#

test:=proc() local h,n;
for h in {1,5,7} do
  for n from 4 to 20 do
    print(isprime(h*2^n-1),rieselsqrt3(h,n))
  od
od end;
```

$$test := \mathbf{proc}()$$ (4.18.2)

```
local h, n;
for h in {1, 5, 7} do
    for n from 4 to 20 do
        print(isprime(h*2^n - 1), rieselsqrt3(h, n))
    end do
end do
end proc
```

```
> test();
```

false, *false*

true, *true*

false, *false*

true, *true*

false, *false*

false, *false*

false, *false*

false, *false*

false, *false*

true, *true*

false, *false*

false, *false*

false, *false*

true, *true*

false, *false*

true, *true*

false, *false*

true, *true*

false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
true, *true*
false, *false*
true, *true*
false, *false*
true, *true*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
false, *false*
false, *false*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
false, *false*

(4.18.3)

false, *false* (4.18.3)

4.19. Feladat.

```
> #
  # This procedure use iteration v<-v^2-2 starting with v=a^h+
  a^-h
  # where the quadratic unit a=r+s*sqrt(D) is given for the
  # primality testing of h*2^n-1. Here n>1 and h<2^n is odd.
  #

  riesel:=proc(h,n,r,s,D) local N,v,i;
  if h>=2^n then error "first parameter is too large",h fi;
  if type(h,even) then error "first parameter have to be odd",h
  fi;
  if n<2 then error "second parameter is too small",h fi;
  N:=h*2^n-1;
  v:=rieselsetup(r,s,D,h,N);
  for i to n-2 do v:=v^2-2 mod N od;
  evalb(v=0); end;
```

riesel := **proc**(*h*, *n*, *r*, *s*, D) (4.19.1)

local *N*, *v*, *i*;

if 2^*n* <= *h* **then**

error "first parameter is too large", *h*

end if;

if *type*(*h*, *even*) **then**

error "first parameter have to be odd", *h*

end if;

if *n* < 2 **then**

error "second parameter is too small", *h*

end if;

N := *h* * 2^*n* − 1;

v := *rieselsetup*(*r*, *s*, D, *h*, *N*);

for *i* **to** *n* − 2 **do**

v := *mod*(*v*^2 − 2, *N*)

end do;

evalb(*v* = 0)

end proc

```
> #
  # Some tests.
  #

  test:=proc() local h,n;
  for h in {1,5,7} do
    for n from 4 to 20 do
      print(isprime(h*2^n-1),riesel(h,n,2,1,3))
    od
  od end;
```

$$test := \mathbf{proc}()$$ (4.19.2)

```
local h, n;
for h in {1, 5, 7} do
    for n from 4 to 20 do
        print(isprime(h*2^n - 1), riesel(h, n, 2, 1, 3))
    end do
end do
end proc
```

```
> test();
```

false, false
true, true
false, false
true, true
false, false
false, false
false, false
false, false
false, false
true, true
false, false
false, false
false, false
true, true
false, false
true, true
false, false

true, *true*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
true, *true*
false, *false*
true, *true*
false, *false*
true, *true*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*
false, *false*
false, *false*
false, *false*
false, *false*
false, *false*
false, *false*
false, *false*
true, *true*
false, *false*

false, *false*

false, *false* (4.19.3)

4.20. Feladat.

```
> #
  # This procedure find units having the form (k+l*sqrt(D))^2/r
  # with small integers k,l and r=k^2-l^2*D in the
  # quadratic extension of the rationals with sqrt(D).
  # D must be a square free integer. The numbers with |k|,|l|<=
  B
  # are tested. We remark that taking -r instead of r we get an
  # other unit. The list of all [k,l,r] is given back.
  #

  findunit:=proc(D,B) local k,l,r,a,b,n,L;
  L:=[];
  for k from 0 to B do
    for l from -B to B do
      r:=k^2-l^2*D;
      if r<>0 then
        a:=(k^2+l^2*D)/r; b:=2*k*l/r;
        if D mod 4=1 then n:=a^2-a*b-b^2*(D-1)/4
        else n:=a^2-D*b^2 fi;
        if n=1 and igcd(k,l,r)=1 then L:=[op(L),[k,l,r]] fi
      fi
    od
  od; L end;
```

$$findunit := \mathbf{proc}(\mathrm{D}, B) \tag{4.20.1}$$

$$\mathbf{local}\ k, l, r, a, b, n, L;$$

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

$$\mathbf{for}\ k\ \mathbf{from}\ 0\ \mathbf{to}\ B\ \mathbf{do}$$

$$\mathbf{for}\ l\ \mathbf{from}\ -B\ \mathbf{to}\ B\ \mathbf{do}$$

$$r := k\hat{}2 - l\hat{}2 * \mathrm{D};$$

$$\mathbf{if}\ r <> 0\ \mathbf{then}$$

$$a := (k\hat{}2 + l\hat{}2 * \mathrm{D}) / r;$$

$$b := 2 * k * l / r;$$

$$\mathbf{if}\ mod(\mathrm{D}, 4) = 1\ \mathbf{then}$$

$$n := a\hat{}2 - a * b - 1 / 4 * b\hat{}2 * (\mathrm{D} - 1)$$

$$\mathbf{else}$$

$$n := a\hat{}2 - \mathrm{D} * b\hat{}2$$

```
                end if;
                if n = 1 and igcd(k, l, r) = 1 then
                    L := [op(L), [k, l, r]]
                end if
            end if
        end do
    end do;
    L
end proc
```

```
> for d in {2,3,5} do print(findunit(d,5)) od;
```

$$[[0, -1, -2], [0, 1, -2], [1, -5, -49], [1, -4, -31], [1, -3, -17], [1, -2, -7], [1, -1, -1], [1, 0, 1], [1, 1, -1], [1, 2, -7], [1, 3, -17], [1, 4, -31], [1, 5, -49], [2, -5, -46], [2, -3, -14], [2, -1, 2], [2, 1, 2], [2, 3, -14], [2, 5, -46], [3, -5, -41], [3, -4, -23], [3, -2, 1], [3, -1, 7], [3, 1, 7], [3, 2, 1], [3, 4, -23], [3, 5, -41], [4, -5, -34], [4, -3, -2], [4, -1, 14], [4, 1, 14], [4, 3, -2], [4, 5, -34], [5, -4, -7], [5, -3, 7], [5, -2, 17], [5, -1, 23], [5, 1, 23], [5, 2, 17], [5, 3, 7], [5, 4, -7]]$$

$$[[0, -1, -3], [0, 1, -3], [1, -5, -74], [1, -4, -47], [1, -3, -26], [1, -2, -11], [1, -1, -2], [1, 0, 1], [1, 1, -2], [1, 2, -11], [1, 3, -26], [1, 4, -47], [1, 5, -74], [2, -5, -71], [2, -3, -23], [2, -1, 1], [2, 1, 1], [2, 3, -23], [2, 5, -71], [3, -5, -66], [3, -4, -39], [3, -2, -3], [3, -1, 6], [3, 1, 6], [3, 2, -3], [3, 4, -39], [3, 5, -66], [4, -5, -59], [4, -3, -11], [4, -1, 13], [4, 1, 13], [4, 3, -11], [4, 5, -59], [5, -4, -23], [5, -3, -2], [5, -2, 13], [5, -1, 22], [5, 1, 22], [5, 2, 13], [5, 3, -2], [5, 4, -23]]$$

$$[[0, -1, -5], [0, 1, -5], [1, 0, 1]] \qquad (4.20.2)$$

4.21. Lucas–Lehmer–teszt a Mersenne–számokra.

```
> #
  # This procedure do Lucas-Lehmer-test for Mersenne
  # numbers M[n]=2^n-1. The exponents are taken from
  # the list L. The result is the sequence of the
  # for which M[n] is prime.
  #
  lucaslehmer:=proc(L::list(posint)) local n,M,i,vi,nL;
```

```
  nL:=[];
  for n in L do vi:=4; M:=2^n-1;
    for i to n-2 do vi:=modp(vi&^2-2,M) od;
    if vi=0 then nL:=[op(nL),n] fi;
  od; nL end;
```

$$lucaslehmer := \mathbf{proc}(L::(list(posint))) \tag{4.21.1}$$

local n, M, i, vi, nL;

$nL := [\,]$;

for n **in** L **do**

$vi := 4$;

$M := 2\,\hat{}\,n - 1$;

for i **to** $n - 2$ **do**

$vi := modp(vi\,\&\hat{}\,2 - 2, M)$

end do;

if $vi = 0$ **then**

$nL := [op(nL), n]$

end if

end do;

nL

end proc

```
> Mersenne:=[2,3,5,7,13,17,19,31,67,127,257]; lucaslehmer
  (Mersenne);
  L:=[i$i=2..257]; lucaslehmer(L);
```

$$Mersenne := [2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257]$$

$$[3, 5, 7, 13, 17, 19, 31, 127]$$

$$L := [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,$$
$$23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,$$
$$42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,$$
$$61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,$$
$$80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,$$
$$99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,$$
$$113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126,$$
$$127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,$$
$$141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,$$
$$155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,$$

169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257]

[3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127] (4.21.2)

4.22. Feladat.

4.23. Valószínűségi teszt.

>

4.24. Feladat.

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

1 (4.24.1)

```
> print(`isprime`);
```

proc(n) ... **end proc** (4.24.2)

```
#
# This is a commented version of the Maple
# function isprime. The details of the procedure are
# given.
#
# The input is the positive integer n to test.
#
# After typecheking the number is tested whether contained in a
list
# of primes below 100 called isprime/w. If yes, it is a prime.
# The second step to check whether it has a gcd>1 with the
# product of these primes. If yes, it is composite. Now any
input
# less then 10201 is a prime. The next step to try whether our
# number has a gcd>1 with the product of primes larger then 100
but
# less then 1000. If yes, the number is composite. After this,
any number
# below 1009^2 is a prime.
#
# The next step is some kind of probabilistic test. First,
# the gcd with (2*3*5*7)^25 is removed from n-1 to
```

```
# obtain a residue r. Then btor;=2^r mod n calculated and
# then the `cyclotest` is called four times with
# bases 2,3,5 and 7, modulus n, with btor and the residue r.
# If the cyclotest proves a composite number then false
returned.
#
# The last step is some kind of Lucas test testing that
# for some D=p^2-4*q where q=1 for which (D|n)=-1 we have
# [V(k+1),V(k)] mod n=[2,p].
#
>
> isprime=proc(n)
> local btor,nr,p,r;
> options remember,system,`Copyright 1993 by Waterloo Maple
Software`;
>     if not type(n,integer) then
>         if type(n,numeric) then ERROR(`argument must be an
integer`)
>         else RETURN('isprime(n)')
>         fi
>     fi;
>     if n < 2 then false
>     elif has(`isprime/w`,n) then true
>     elif igcd(230556796394551842475310214733175607O,n) <> 1
then false
>     elif n < 10201 then true
>     elif igcd
(84969694892334181105323399091873499659260625866489327366115454263
342203893270769390909069477309509137509786917118668028861499333825
097682386722983737962963066757674131126736578936440788157186969893
730633113066478620448624949257324022627395437363639038752608166758
661255956834630697220447512298848222228550062683786342519960225996
301315945644470064720696621750477244528915927867113,n) <>
1 then
>         false
>     elif n < 1018081 then true
>     else
>         nr := igcd(408410100000,n-1);
>         nr := igcd(nr^5,n-1);
>         r := iquo(n-1,nr);
>         btor := modp(power(2,r),n);
>         if `isprime/cyclotest`(n,btor,2,r) = false or
>             irem(nr,3) = 0 and `isprime/cyclotest`(n,btor,3,r)
= false or
>             irem(nr,5) = 0 and `isprime/cyclotest`(n,btor,5,r)
= false or
>             irem(nr,7) = 0 and `isprime/cyclotest`(n,btor,7,r)
= false then
```

```
>             RETURN(false)
>         fi;
>         for p from 3 while numtheory[jacobi](p^2-4,n) <> -1 do
  od;
>         evalb(`isprime/TraceModQF`(p,n+1,n) = [2,p])
>     fi
> end;
```

$$isprime = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This is the initial list of primes.
#
>
> isprime/w=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97];
```

$$\frac{isprime}{w} = ([2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97])$$

```
>
>
#
# This is the probabilistic primality test some
# with refinements.
#
# The input parameters are the number n to test,
# a residue r of n-1 from which the gcd with
# (2*3*5*7)^25 is removed, btor=2^r mod n and
# one of the numbers 2, 3, 5 and 7.
#
# The test run only with the base 2. At most
# The last 25 step of the test is done, but
# for i=3,5,7 also possible to check something
# similar to the probabilistic test: if x^3 mod n=1
# and n is a prime then in the case x mod n<>1
# the congruence x^2+x+1 mod n=0 have to be satisfied.
# This gives some additional information if 3|n-1.
# Similar trick works for 5|n-1, 7|n-1, etc.
#
>
> isprime/cyclotest=proc(n,btor,i,r)
> local nr,k,t,s,x;
> options `Copyright 1993 by Waterloo Maple Software`;
>     nr := iquo(n-1,r);
>     for k from 0 while irem(nr,i,'t') = 0 do  nr := t od;
>     x := modp(power(btor,nr),n);
>     if x = 1 then RETURN(FAIL) fi;
```

```
>     to k do
>         s := `isprime/sumxtor`(x,i,n);
>         if s[1] = 0 then RETURN(FAIL) elif s[2] = 1 then
RETURN(false) fi;
>         x := s[2]
>     od;
>     false
> end;
```

$$\frac{isprime}{cyclotest} = \mathbf{proc}(n, btor, i, r) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This procedure calculates the pair
# [(1+x+..+x^(r-1) mod n, x^r mod n].
#
# For r<6 direct calculation used.
# Otherwise r is written as a*r1+b
# with r1~sqrt(r) and the pair is
# recursively calculated.
#
>
> isprime/sumxtor=proc(x,r,n)
> local i1,i2,i3,r1,r2,r3,s,xj;
> options `Copyright 1993 by Waterloo Maple Software`;
>     if r < 1 then ERROR(`r is too small`)
>     elif r = 1 then [1,x]
>     elif r < 6 then
>         s := x+1;
>         xj := modp(x^2,n);
>         to r-2 do  s := s+xj; xj := modp(xj*x,n) od;
>         [modp(s,n),xj]
>     else
>         i1 := isqrt(r);
>         i2 := iquo(r,i1,'i3');
>         if i3 = 0 then
>             r2 := procname(x,i2,n);
>             r1 := procname(r2[2],i1,n);
>             [modp(r2[1]*r1[1],n),r1[2]]
>         else
>             r3 := procname(x,i3,n);
>             r2 := procname(x,i2,n);
>             r1 := procname(r2[2],i1,n);
>             [modp(r3[1]+r3[2]*modp(r2[1]*r1[1],n),n),modp(r3
[2]*r1[2],n)]
>         fi
>     fi
> end;
```

$$\frac{isprime}{sumxtor} = \mathbf{proc}(x, r, n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This subroutine calculate the Lucas sequence
# terms [V(k),V(k-1)] modulo n.
#
# The input parameters are the modulus n,
# p (and q=1 supposed) and k.
#
# First, the binary form of k-1 is calculated.
# Then starting from [V(1),V(0)], the result
# is calculated by the recursion of the V's.
#
>
> isprime/TraceModQF=proc(p,k,n)
> local i,kk,trc,v;
> options `Copyright 1993 by Waterloo Maple Software`;
>     kk := k;
>     for i while 1 < kk do  kk := iquo(kk+1,2,v[i]) od;
>     trc := [p,2];
>     for i from i-1 by -1 to 1 do
>         if v[i] = 1 then trc := modp([trc[1]^2-2,trc[1]*trc[2]
-p],n)
>         else trc := modp([trc[1]*trc[2]-p,trc[2]^2-2],n)
>         fi
>     od;
>     trc
> end;
```

$$\frac{isprime}{TraceModQF} = \mathbf{proc}(p, k, n) \ldots \mathbf{end\ proc}$$

```
>
>
>
```

▶ 4.25. Megjegyzés.

▶ 4.26. Williams p+1 módszere.

▶ 4.27. Lemma.

▼ 4.28. Feladat.

```
> #
```

```
# This procedure is Williams' p+1 method for factorization of
n.
# P is the parameter for the Lucas sequence. Calculation of
V's
# goes up to V_k!.  The result is the factor found;
# gcd is calculated after m steps.
#

williams:=proc(n::posint,P::integer,k::posint,m::posint)
local g,x,i,j; x:=P;
for i from 2 by m to k do
  for j from i to i+m-1 while j<=k do
    x:=lucasV(j,x,1,n)[1];
  od;
  g:=igcd(x-2,n);
  if g>1 then RETURN(g) fi;
od; igcd(x-2,n) end;
```

$$williams := \mathbf{proc}(n::posint,\ P::integer,\ k::posint,\ m::posint) \tag{4.28.1}$$

local $g,\ x,\ i,\ j$;

$x := P$;

for i **from** 2 **by** m **to** k **do**

for j **from** i **to** $i + m - 1$ **while** $j <= k$ **do**

$x := lucasV(j,\ x,\ 1,\ n)[1]$

end do;

$g := igcd(x - 2,\ n)$;

if $1 < g$ **then**

$RETURN(g)$

end if

end do;

$igcd(x - 2,\ n)$

end proc

```
> williams(25852,3,10,1); williams(999863*999883*999907,3,1000,
  1);
```

$$4$$

$$999907 \tag{4.28.2}$$

▶ 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
▶ 12. Polinomfaktorizálás
▶ 13. Az AKS–teszt
▶ 14. A szita módszerek alapjai
▶ 15. Számtest szita
▶ 16. Vegyes problémák