Számítógépes számelmélet

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

1. A prímek eloszlása, szitálás

2. Egyszerű faktorizálási módszerek

3. Egyszerű prímtesztelési módszerek

```
> restart; with(numtheory);
[GIgcd, bigomega, cfrac, cfracpol, cyclotomic, divisors, factorEQ, factorset,        (3.1)
    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]
```

3.1. Kérdés.

3.2. Feladat.

```
> #
  # This procedure do Wilson's test. The return value
  # is a pair of lower two digits of (n-1)!+1 in base n.
  #

  wilsontest:=proc(n::posint) local i,r,m;
  r:=1; m:=n^2;
  for i from 2 to n-1 do r:=r*i mod m od;
  r:=r+1 mod m; [irem(r,n),iquo(r,n)] end;
wilsontest := proc(n)                                                     (3.2.1)
    local i, r, m;
    r := 1;
    m := n^2;
    for i from 2 to n - 1 do
```

```
        r := mod(r*i, m)
    end do;
    r := mod(r + 1, m);
    [irem(r, n), iquo(r, n)]
end proc
```

```
> #
  # This procedure do Wilson's test and
  # look for Wilson numbers until a given
  # limit.
  #

  wilson:=proc(n) local i,r,p,w;
  p:=[]; w:=[];
  for i from 2 to n do
    r:=wilsontest(i);
    if r[1]=0 then
      p:=[op(p),i];
      if r[2]=0 then w:=[op(w),i] fi
    fi
  od; [w,p] end;
```

```
wilson := proc(n)                                                    (3.2.2)
    local i, r, p, w;
    p := [ ];
    w := [ ];
    for i from 2 to n do
        r := wilsontest(i);
        if r[1] = 0 then
            p := [op(p), i];
            if r[2] = 0 then
                w := [op(w), i]
            end if
        end if
    end do;
    [w, p]
end proc
```

```
> wilson(1000);
```

```
[[5, 13, 563], [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,     (3.2.3)
    61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211,
223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283,
293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,
383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461,
463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563,
569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643,
647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739,
743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829,
839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937,
941, 947, 953, 967, 971, 977, 983, 991, 997]]
```

3.3. Probléma.

3.4. Fermat-teszt.

```
> n:=(2^28-9)/7; 3&^(n-1) mod n; 2&^(n-1)mod n;
                              n := 38347921

                                  1

                               10225489                         (3.4.1)
```

3.5. Feladat.

```
> #
  # This procedure do Fermat's test for the number n with base
  a.
  # If the result is false, the number is composite, if FAIL,
  # then may be prime.
  #

  fermattest:=proc(n::posint,a::posint)
  if type(n,even) then error "first argument must be odd",n fi;
  if n<3 then error "first argument is too small",n fi;
  if n<=a then error "second argument is too large",a fi;
  if modp(a&^(n-1),n)<>1 then RETURN(false) fi; FAIL end;
fermattest := proc(n::posint, a::posint)                        (3.5.1)
   if type(n, even) then
       error "first argument must be odd", n
   end if;
```

```
    if n < 3 then
        error "first argument is too small", n
    end if;
    if n <= a then
        error "second argument is too large", a
    end if;
    if modp(a &^ (n - 1), n) <> 1 then
        RETURN(false)
    end if;
    FAIL
end proc
```

```
> n:=(2^28-9)/7; fermattest(n,3); fermattest(n,2);
                              n := 38347921
                                  FAIL
                                  false                          (3.5.2)
```

3.6. Feladat.

```
> #
  # This procedure look for pseudoprimes until a given limit n
  # in bases in the list L.
  #

  pseudoprime:=proc(n::posint,L::list(posint)) local pp,i,a,t;
  pp:=[];
  for i from max(3,op(L))+1 to n do
    if type(i,even) then next fi;
    for a in L do
      t:=fermattest(i,a);
      if not t then break fi
    od;
    if t=FAIL then
      if not isprime(i) then pp:=[op(pp),i] fi
    fi
  od; pp end;
```

```
pseudoprime := proc(n::posint, L::(list(posint)))                    (3.6.1)
    local pp, i, a, t;
    pp := [];
    for i from max(3, op(L)) + 1 to n do
```

```
        if type(i, even) then
            next
        end if;
        for a in L do
            t := fermattest(i, a);
            if not t then
                break
            end if
        end do;
        if t = FAIL then
            if not isprime(i) then
                pp := [op(pp), i]
            end if
        end if
    end do;
    pp
end proc
```

```
> pseudoprime(10000,[2]);
[341, 561, 645, 1105, 1387, 1729, 1905, 2047, 2465, 2701, 2821, 3277,      (3.6.2)
    4033, 4369, 4371, 4681, 5461, 6601, 7957, 8321, 8481, 8911]
```

3.7. Feladat.

```
> #
  # This procedure look for Carmichael
  # numbers until a given limit n.
  #

  carmichael:=proc(n) local L,i,a,t;
  L:=[];
  for i from 2 to n do
    if type(i,even) then next fi;
    if isprime(i) then next fi;
    for a from 2 to i-1 do
      if igcd(i,a)=1 then
        t:=fermattest(i,a);
        if not t then break fi
      fi
      od;
      if t=FAIL then L:=[op(L),i] fi
    od; L end;
```

```
carmichael := proc(n)                                                   (3.7.1)
    local L, i, a, t;
    L := [ ];
    for i from 2 to n do
        if type(i, even) then
            next
        end if;
        if isprime(i) then
            next
        end if;
        for a from 2 to i − 1 do
            if igcd(i, a) = 1 then
                t := fermattest(i, a);
                if not t then
                    break
                end if
            end if
        end do;
        if t = FAIL then
            L := [op(L), i]
        end if
    end do;
    L
end proc
```

```
> carmichael(10000);
              [561, 1105, 1729, 2465, 2821, 6601, 8911]                 (3.7.2)
```

```
> #
  # Much larger speed can be obtained using that a number n is
  # Carmichael number if and only if it is squre free, has at
  least
  # three prime factors and p-1 divides n-1 for every prime
  factor
  # p of n.
```

```
#
```

3.8. Lemma.

3.9. Lemma.

3.10. Tétel.

3.11. Definíció.

3.12. Euler tétele.

3.13. Gauss lemmája.

3.14. Gauss kvadratikus reciprocitási törvénye.

3.15. Definíció.

3.16. Tétel.

3.17. Példa.

3.18. A Jacobi-szimbólum kiszámítása.

```
> #
  # This recursive procedure calculates
  # the Jacobi symbol (n|m) for integers
  # m>2 and n, where m is odd.
  #

  jacobisymbol:=proc(n::integer,m::posint) local s;
  if type(m,even) then error "second argument must be odd" fi;
  if n=0 then RETURN(0) fi;
  if n=1 then RETURN(1) fi;
  if n<0 then
    if type((m-1)/2,odd) then
      RETURN(-jacobisymbol(-n,m))
    else
      RETURN(jacobisymbol(-n,m))
    fi
  fi;
  if type(n,even) then
    if type((irem(m,16)^2-1)/8,odd) then
```

```
      RETURN(-jacobisymbol(n/2,m))
    else
      RETURN(jacobisymbol(n/2,m))
    fi
  fi;
  if n>m then RETURN(jacobisymbol(irem(n,m),m)) fi;
  if type(irem((m-1)/2,2)*irem((n-1)/2,2),odd) then
    -jacobisymbol(m,n)
  else
    jacobisymbol(m,n)
  fi
  end;
```

```
jacobisymbol := proc(n::integer, m::posint)                    (3.18.1)
    local s;
    if type(m, even) then
        error "second argument must be odd"
    end if;
    if n = 0 then
        RETURN(0)
    end if;
    if n = 1 then
        RETURN(1)
    end if;
    if n < 0 then
        if type(1 / 2 * m − 1 / 2, odd) then
            RETURN(−jacobisymbol(−n, m))
        else
            RETURN(jacobisymbol(−n, m))
        end if
    end if;
    if type(n, even) then
        if type(1 / 8 * irem(m, 16)^2 − 1 / 8, odd) then
            RETURN(−jacobisymbol(1 / 2 * n, m))
        else
            RETURN(jacobisymbol(1 / 2 * n, m))
        end if
        end if;
        if m < n then
            RETURN(jacobisymbol(irem(n, m), m))
        end if;
        if type(irem(1 / 2 * m - 1 / 2, 2) * irem(1 / 2 * n - 1 / 2, 2), odd) then
            -jacobisymbol(m, n)
        else
            jacobisymbol(m, n)
        end if
    end proc
```

```
> debug(jacobisymbol); jacobisymbol(76,131);
```

```
                          jacobisymbol
{--> enter jacobisymbol, args = 76, 131
{--> enter jacobisymbol, args = 38, 131
{--> enter jacobisymbol, args = 19, 131
{--> enter jacobisymbol, args = 131, 19
{--> enter jacobisymbol, args = 17, 19
{--> enter jacobisymbol, args = 19, 17
{--> enter jacobisymbol, args = 2, 17
{--> enter jacobisymbol, args = 1, 17
<-- exit jacobisymbol (now in jacobisymbol) = 1}
<-- exit jacobisymbol (now in jacobisymbol) = 1}
<-- exit jacobisymbol (now in jacobisymbol) = 1}
                             1
<-- exit jacobisymbol (now in jacobisymbol) = 1}
<-- exit jacobisymbol (now in jacobisymbol) = 1}
                             -1
<-- exit jacobisymbol (now in jacobisymbol) = 1}
<-- exit jacobisymbol (now in jacobisymbol) = 1}
<-- exit jacobisymbol (now at top level) = 1}
                             -1                                  (3.18.2)
```

```
> #
  #  What is this?
  #

  jacobiproba:=proc(a,b,p,n) local i,j,L;
  for i from 0 to n-1 do L:=[];
    for j from 0 to p-1 do L:=[op(L),jacobi(a,b+(i*p+j)*a)];
  od;
    print(L);
  od; end;
```

```
jacobiproba := proc(a, b, p, n)                                                  (3.18.3)
    local i, j, L;
    for i from 0 to n − 1 do
        L := [ ];
        for j from 0 to p − 1 do
            L := [op(L), numtheory:-jacobi(a, b + (i* p + j) * a)]
        end do;
        print(L)
    end do
end proc
```

▶ 3.19. Feladat.

▼ 3.20. Feladat.

```
> interface(verboseproc=2);
                                        1                                        (3.20.1)
```

```
> print(jacobi);
proc(a::(Or(integer, Not(constant))), b::(Or(nonnegint,                          (3.20.2)
    Not(constant))))
     ...
 end proc
```

▼ 3.21. Feladat.

```
> print(legendre);
proc(a::(Or(integer, Not(constant))), p::(Or(prime, Not(constant))))             (3.21.1)
     ...
 end proc
```

▼ 3.22. Soloway-Strassen-teszt.

```
> n:=561; 5&^(n-1) mod n; 5&^((n-1)/2) mod n; jacobi(5,n);
                                      n := 561

                                        1

                                        67
```

```
1                                                        (3.22.1)
```

3.23. Feladat.

```
> #
  # This procedure do the Soloway-Strassen probabilistic test.
  # The first parameter is the integer to test, the second
  # the number of rounds. if the result is false, the number
  # is composite, if FAIL, then probably prime.
  #
  solowaystrassen:=proc(n::posint,m::posint) local i,b,rnd;
  if type(n,even) then error "first argument must be odd",n fi;
  if n<5 then error "first argument is too small",n fi;
  rnd:=rand(2..n-2);
  for i to m do b:=rnd();
    if igcd(b,n)>1 then RETURN(false) fi;
    if modp(b&^((n-1)/2),n)<>modp(jacobi(b,n),n) then RETURN
  (false) fi
  od; FAIL end;
```

```
solowaystrassen := proc(n::posint, m::posint)                    (3.23.1)
    local i, b, rnd;
    if type(n, even) then
        error "first argument must be odd", n
    end if;
    if n < 5 then
        error "first argument is too small", n
    end if;
    rnd := rand(2 ..n − 2);
    for i to m do
        b := rnd( );
        if 1 < igcd(b, n) then
            RETURN(false)
        end if;
        if modp(b &^ (1 / 2 * n − 1 / 2),
       n) <> modp(numtheory:-jacobi(b, n), n) then
            RETURN(false)
        end if
    end do;
    FAIL
end proc
```

```
> debug(solowaystrassen); solowaystrassen(5,2);
```

```
                              solowaystrassen

{--> enter solowaystrassen, args = 5, 2
rnd := proc( ) proc( ) option builtin; 393 end proc(6, 2, 1) + 2 end proc
                                  b := 2
                                  b := 2
                                   FAIL
<-- exit solowaystrassen (now at top level) = FAIL}
                                   FAIL                              (3.23.2)
```

3.24. Miller-Rabin-teszt.

```
> millerrabin:=proc(n::posint,b::posint) local j,e,r,bb;
  if n<9 then ERROR(`first parameter is too small`,n) fi;
  if type(n,even) then ERROR(`first parameter is even`,n) fi;
  if b<2 then ERROR(`second parameter is too small`,b) fi;
  if n<=b then ERROR(`second parameter is too large`,b) fi;
  e:=0; r:=n-1;
  while type(r,even) do e:=e+1; r:=r/2 od;
  bb:=modp(b&^r,n); if bb=1 then RETURN(FAIL) fi;
  for j to e while bb<>n-1 do bb:=modp(bb&^2,n) od;
  if j=e+1 then RETURN(false) fi; FAIL; end;
```

```
millerrabin := proc(n::posint, b::posint)                            (3.24.1)
    local j, e, r, bb;
    if n < 9 then
        ERROR(first parameter is too small, n)
    end if;
    if type(n, even) then
        ERROR(first parameter is even, n)
    end if;
    if b < 2 then
        ERROR(second parameter is too small, b)
    end if;
    if n <= b then
        ERROR(second parameter is too large, b)
    end if;
    e := 0;
    r := n − 1;
    while type(r, even) do
        e := e + 1;
        r := 1 / 2 * r
    end do;
    bb := modp(b &^ r, n);
    if bb = 1 then
        RETURN(FAIL)
    end if;
    for j to e while bb <> n − 1 do
        bb := modp(bb &^ 2, n)
    end do;
    if j = e + 1 then
        RETURN(false)
    end if;
    FAIL
end proc
```

```
> rabin:=proc(n::posint,m::posint) local i,j,b,e,r,rnd;
  if n<9 then ERROR(`first parameter is too small`,n) fi;
  if type(n,even) then ERROR(`first parameter is even`,n) fi;
  rnd:=rand(2..n-1); e:=0; r:=n-1;
  while type(r,even) do e:=e+1; r:=r/2 od;
  for i to m do
    b:=rnd();
    if 1<gcd(b,n) then RETURN(false) fi;
    modp(b&^r,n); if %=1 then next fi;
    for j to e while %<>n-1 do modp(%&^2,n) od;
    if j=e+1 then RETURN(false) fi;
  od; FAIL end;
```

```
rabin := proc(n::posint, m::posint)                                    (3.24.2)
    local i, j, b, e, r, rnd;
    if n < 9 then
        ERROR(first parameter is too small, n)
    end if;
    if type(n, even) then
        ERROR(first parameter is even, n)
    end if;
    rnd := rand(2 ..n − 1 );
    e := 0;
    r := n − 1;
    while type(r, even) do
        e := e + 1;
        r := 1 / 2 * r
    end do;
    for i to m do
        b := rnd( );
        if 1 < gcd(b, n) then
            RETURN(false)
        end if;
        modp(b &^ r, n);
        if % = 1 then
            next
        end if;
        for j to e while % <> n − 1 do
            modp(% &^ 2, n)
        end do;
        if j = e + 1 then
            RETURN(false)
        end if
    end do;
    FAIL
end proc
```

```
> trueisprime:=proc(n::posint) local f;
  if n<2 then RETURN(false) fi;
  if n=2 or n=3 or n=5 or n=7 or n=11 or n=13 then RETURN(true)
  fi;
  if 1<gcd(n,30030) then RETURN(false) fi;
  if n<289 then RETURN(true) fi;
if 1<gcd(n,247110827) then RETURN(false) fi;
f:=millerrabin(n,2); if f=false then RETURN(f) fi;
f:=millerrabin(n,3); if f=false then RETURN(f) fi;
if n<1373653 then RETURN(true) fi;
f:=millerrabin(n,5); if f=false then RETURN(f) fi;
if n< 5326001 then RETURN(true) fi;
f:=millerrabin(n,7); if f=false then RETURN(f) fi;
if n<3215031751 then RETURN(true) fi;
f:=millerrabin(n,11); if f=false then RETURN(f) fi;
if n<2152302898747 then RETURN(true) fi;
f:=millerrabin(n,13); if f=false then RETURN(f) fi;
if n<3474749660383 then RETURN(true) fi;
f:=millerrabin(n,17); if f=false then RETURN(f) fi;
if n<341550071728321 then RETURN(true) fi; FAIL end;
```

```
trueisprime := proc(n::posint)                                        (3.24.3)
    local f;
    if n < 2 then
        RETURN(false)
    end if;
    if n = 2 or n = 3 or n = 5 or n = 7 or n = 11 or n = 13
    then
        RETURN(true)
    end if;
    if 1 < gcd(n, 30030) then
        RETURN(false)
    end if;
    if n < 289 then
        RETURN(true)
    end if;
    if 1 < gcd(n, 247110827) then
        RETURN(false)
    end if;
    f := millerrabin(n, 2);
    if f = false then
        RETURN(f)
    end if;
    f := millerrabin(n, 3);
```

```
if $f = false$ then
    $RETURN(f)$
end if;
if $n < 1373653$ then
    $RETURN(true)$
end if;
$f := millerrabin(n, 5)$;
if $f = false$ then
    $RETURN(f)$
end if;
if $n < 5326001$ then
    $RETURN(true)$
end if;
$f := millerrabin(n, 7)$;
if $f = false$ then
    $RETURN(f)$
end if;
if $n < 3215031751$ then
    $RETURN(true)$
end if;
$f := millerrabin(n, 11)$;
if $f = false$ then
    $RETURN(f)$
end if;
if $n < 2152302898747$ then
    $RETURN(true)$
end if;
$f := millerrabin(n, 13)$;
if $f = false$ then
    $RETURN(f)$
end if;
if $n < 3474749660383$ then
```

```
        RETURN(true)
    end if;
    f := millerrabin(n, 17);
    if f = false then
        RETURN(f)
    end if;
    if n < 341550071728321 then
        RETURN(true)
    end if;
    FAIL
end proc
```

3.25. Feladat.

3.26. Feladat.

3.27. Miller tétele.

3.28. Feladat.

```
> #
  # This procedure do Miller's deterministic test
  # based on GRH. The parameter is the integer to test.
  #

  millertest:=proc(n::posint) local i,b,t;
  if n=1 then RETURN(false) fi;
  if n=2 or n=3 or n=5 or n=7 or n=11 or n=13 then RETURN(true)
  fi;
  if type(n,even) then RETURN(false) fi;
  if n=9 then RETURN(false) fi;
  b:=floor(2.0002*ln(n)^2);
  for i from 2 to b do
    if not millerrabin(n,i) then RETURN(false) fi;
  od; true end;
```

```
millertest := proc(n::posint)                                    (3.28.1)
    local i, b, t;
    if n = 1 then
        RETURN(false)
    end if;
    if n = 2  or  n = 3  or  n = 5  or  n = 7  or  n = 11  or  n = 13
    then
        RETURN(true)
    end if;
    if type(n, even) then
        RETURN(false)
    end if;
    if n = 9 then
        RETURN(false)
    end if;
    b := floor(2.0002*ln(n)^2);
    for i from 2 to b do
        if not  millerrabin(n, i) then
            RETURN(false)
        end if
    end do;
    true
end proc
```

```
> millertest(17);
                                true                                (3.28.2)
```

3.29. Lucas-teszt.

```
> #
  # This procedure do Lucas' test
  # for the number n on the naive way
  # using base a
  #

  naivelucastest:=proc(n,a) local m;
  if igcd(a,n)>1 then RETURN(false) fi;
  if modp(a&^(n-1),n)<>1 then RETURN(false) fi;
  for m from 2 to n-2 do
    if modp(a&^m,n)=1 then RETURN(false) fi
  od; true end;
naivelucastest := proc(n, a)                                        (3.29.1)
```

```
    local m;
    if 1 < igcd(a, n) then
        RETURN(false)
    end if;
    if modp(a &^ (n - 1), n) <> 1 then
        RETURN(false)
    end if;
    for m from 2 to n - 2 do
        if modp(a &^ m, n) = 1 then
            RETURN(false)
        end if
    end do;
    true
end proc
```

```
> #
  # This procedure do Lucas' test
  # for the number n using the set S of
  # factors of n-1 and base a
  #

  lucastest:=proc(n,a,S) local p;
  if n<=a then ERROR(`second parameter is too large`,a) fi;
  if igcd(a,n)>1 then RETURN(false) fi;
  if modp(a&^(n-1),n)<>1 then RETURN(false) fi;
  for p in S do
    if modp(a&^((n-1)/p),n)=1 then RETURN(FAIL) fi
  od; true end;
```

```
lucastest := proc(n, a, S)                                        (3.29.2)
    local p;
    if n <= a then
        ERROR(second parameter is too large, a)
    end if;
    if 1 < igcd(a, n) then
        RETURN(false)
    end if;
    if modp(a &^ (n - 1), n) <> 1 then
        RETURN(false)
    end if;
    for p in S do
        if modp(a &^ ((n − 1) / p), n) = 1 then
            RETURN(FAIL)
        end if
    end do;
    true
end proc
```

```
> #
  # This procedure do Lucas' test
  # for numbers n*k^m+1 with small
  # n,k,m and base a.
  #

  speclucas:=proc(n,k,m,a) local S,N,p;
  S:=factorset(n);
  if m>0 then S:=S union factorset(k) fi;
  N:=n*k^m+1;
  if N<=a then ERROR(`last parameter is too large`,a) fi;
  if igcd(a,N)>1 then RETURN(false) fi;
  if modp(a&^(N-1),N)<>1 then RETURN(false) fi;
  for p in S do
    if modp(a&^((N-1)/p),N)=1 then RETURN(FAIL) fi
  od; true end;
```

```
speclucas := proc(n, k, m, a)                                    (3.29.3)
    local S, N, p;
    S := numtheory:-factorset(n);
    if 0 < m then
        S := union(S, numtheory:-factorset(k))
    end if;
    N := n * k^m + 1;
    if N <= a then
        ERROR(last parameter is too large, a)
    end if;
    if 1 < igcd(a, N) then
        RETURN(false)
    end if;
    if modp(a &^ (N − 1), N) <> 1 then
```

```
            RETURN(false)
        end if;
        for p in S do
            if modp(a &^ ((N − 1) / p), N) = 1 then
                RETURN(FAIL)
            end if
        end do;
        true
    end proc
```

```
> #
  # This procedure try to find a large prime using Lucas' test.
  # The search goes for n,n+1,n+2,... between numbers n*k^m+1.
  #

  largewithlucas:=proc(n::posint,k::posint,m::posint) local i,
  a,aa,f;
  if k<2 then ERROR(`second parameter is too small`,k) fi;
  for i from n do
    f:=FAIL;
    for a from 2 while f=FAIL do
      f:=speclucas(i,k,m,a); aa:=a;
    od;
    print(i*k^m+1,aa,f);
  od; end;
```

```
largewithlucas := proc(n::posint, k::posint, m::posint)                    (3.29.4)
    local i, a, aa, f;
    if k < 2 then
        ERROR(second parameter is too small, k)
    end if;
    for i from n do
        f := FAIL;
        for a from 2 while f = FAIL do
            f := speclucas(i, k, m, a);
            aa := a
        end do;
        print(i* k^m + 1, aa, f)
    end do
end proc
> debug(largewithlucas); debug(speclucas); largewithlucas(1,2,
  16);
                         largewithlucas

                            speclucas

{--> enter largewithlucas, args = 1, 2, 16
                            f:= FAIL

{--> enter speclucas, args = 1, 2, 16, 2
                             S:= {}

                             S:= {2}

                           N:= 65537

                             p:= 2

<-- exit speclucas (now in largewithlucas) = FAIL}
                            f:= FAIL

                            aa:= 2

{--> enter speclucas, args = 1, 2, 16, 3
                             S:= {}

                             S:= {2}

                           N:= 65537

                             p:= 2

                             true

<-- exit speclucas (now in largewithlucas) = true}
                            f:= true

                            aa:= 3

                       65537, 3, true

                            f:= FAIL

{--> enter speclucas, args = 2, 2, 16, 2
                             S:= {2}

                             S:= {2}

                          N:= 131073

<-- exit speclucas (now in largewithlucas) = false}
                           f:= false

                            aa:= 2

                      131073, 2, false

                            f:= FAIL
```

```
{--> enter speclucas, args = 3, 2, 16, 2
                                S:= {3}

                               S:= {2, 3}

                              N:= 196609

<-- exit speclucas (now in largewithlucas) = false}
                               f:= false

                                aa:= 2

                          196609, 2, false

                                f:= FAIL

{--> enter speclucas, args = 4, 2, 16, 2
                                S:= {2}

                                S:= {2}

                              N:= 262145

<-- exit speclucas (now in largewithlucas) = false}
                               f:= false

                                aa:= 2

                          262145, 2, false

                                f:= FAIL

{--> enter speclucas, args = 5, 2, 16, 2
                                S:= {5}

                               S:= {2, 5}

                              N:= 327681

<-- exit speclucas (now in largewithlucas) = false}
                               f:= false

                                aa:= 2

                          327681, 2, false

                                f:= FAIL

{--> enter speclucas, args = 6, 2, 16, 2
                               S:= {2, 3}

                               S:= {2, 3}

                              N:= 393217

<-- exit speclucas (now in largewithlucas) = false}
                               f:= false

                                aa:= 2

                          393217, 2, false
```

```
f:= FAIL
{--> enter speclucas, args = 7, 2, 16, 2
S:= {7}
S:= {2, 7}
N:= 458753
<-- exit speclucas (now in largewithlucas) = false}
f:= false
aa:= 2
458753, 2, false
f:= FAIL
{--> enter speclucas, args = 8, 2, 16, 2
S:= {2}
S:= {2}
N:= 524289
<-- exit speclucas (now in largewithlucas) = false}
f:= false
aa:= 2
524289, 2, false
f:= FAIL
{--> enter speclucas, args = 9, 2, 16, 2
S:= {3}
S:= {2, 3}
N:= 589825
<-- exit speclucas (now in largewithlucas) = false}
f:= false
aa:= 2
589825, 2, false
f:= FAIL
{--> enter speclucas, args = 10, 2, 16, 2
S:= {2, 5}
S:= {2, 5}
N:= 655361
<-- exit speclucas (now in largewithlucas) = false}
f:= false
aa:= 2
```

```
                         655361, 2, false
                            f:= FAIL
{--> enter speclucas, args = 11, 2, 16, 2
                            S:= {11}
                           S:= {2, 11}
                           N:= 720897
<-- exit speclucas (now in largewithlucas) = false}
                            f:= false
                             aa:= 2
                         720897, 2, false
                            f:= FAIL
{--> enter speclucas, args = 12, 2, 16, 2
                           S:= {2, 3}
                           S:= {2, 3}
                           N:= 786433
                             p:= 2
<-- exit speclucas (now in largewithlucas) = FAIL}
                            f:= FAIL
                             aa:= 2
{--> enter speclucas, args = 12, 2, 16, 3
                           S:= {2, 3}
                           S:= {2, 3}
                           N:= 786433
                             p:= 2
<-- exit speclucas (now in largewithlucas) = FAIL}
                            f:= FAIL
                             aa:= 3
{--> enter speclucas, args = 12, 2, 16, 4
                           S:= {2, 3}
                           S:= {2, 3}
                           N:= 786433
                             p:= 2
<-- exit speclucas (now in largewithlucas) = FAIL}
                            f:= FAIL
                             aa:= 4
```

```
{--> enter speclucas, args = 12, 2, 16, 5
                                        S:= {2, 3}

                                        S:= {2, 3}

                                       N:= 786433

                                          p:= 2

                                          p:= 3

<-- exit speclucas (now in largewithlucas) = FAIL}
                                        f:= FAIL

                                         aa:= 5

{--> enter speclucas, args = 12, 2, 16, 6
                                        S:= {2, 3}

                                        S:= {2, 3}

                                       N:= 786433

                                          p:= 2

<-- exit speclucas (now in largewithlucas) = FAIL}
                                        f:= FAIL

                                         aa:= 6

{--> enter speclucas, args = 12, 2, 16, 7
                                        S:= {2, 3}

                                        S:= {2, 3}

                                       N:= 786433

                                          p:= 2

<-- exit speclucas (now in largewithlucas) = FAIL}
                                        f:= FAIL

                                         aa:= 7

{--> enter speclucas, args = 12, 2, 16, 8
                                        S:= {2, 3}

                                        S:= {2, 3}

                                       N:= 786433

                                          p:= 2

<-- exit speclucas (now in largewithlucas) = FAIL}
                                        f:= FAIL

                                         aa:= 8

{--> enter speclucas, args = 12, 2, 16, 9
                                        S:= {2, 3}
S:= {2, 3}
N:= 786433
p:= 2
<-- exit speclucas (now in largewithlucas) = FAIL}
f:= FAIL
aa:= 9
{--> enter speclucas, args = 12, 2, 16, 10
S:= {2, 3}
S:= {2, 3}
N:= 786433
p:= 2
p:= 3
true
<-- exit speclucas (now in largewithlucas) = true}
f:= true
aa:= 10
786433, 10, true
f:= FAIL
{--> enter speclucas, args = 13, 2, 16, 2
S:= {13}
S:= {2, 13}
N:= 851969
<-- exit speclucas (now in largewithlucas) = false}
f:= false
aa:= 2
851969, 2, false
f:= FAIL
{--> enter speclucas, args = 14, 2, 16, 2
S:= {2, 7}
S:= {2, 7}
N:= 917505
<-- exit speclucas (now in largewithlucas) = false}
f:= false
aa:= 2
917505, 2, false
```

```
                         f:= FAIL
{--> enter speclucas, args = 15, 2, 16, 2
                        S:= {3, 5}
                       S:= {2, 3, 5}
                       N:= 983041
<-- exit speclucas (now in largewithlucas) = false}
                         f:= false
                          aa:= 2
                    983041, 2, false
                         f:= FAIL
{--> enter speclucas, args = 16, 2, 16, 2
                          S:= {2}
                          S:= {2}
                      N:= 1048577
<-- exit speclucas (now in largewithlucas) = false}
                         f:= false
                          aa:= 2
                   1048577, 2, false
                         f:= FAIL
{--> enter speclucas, args = 17, 2, 16, 2
                         S:= {17}
                        S:= {2, 17}
                      N:= 1114113
<-- exit speclucas (now in largewithlucas) = false}
                         f:= false
                          aa:= 2
                   1114113, 2, false
                         f:= FAIL
{--> enter speclucas, args = 18, 2, 16, 2
                        S:= {2, 3}
<-- ERROR in speclucas (now in largewithlucas) = interrupted}
<-- ERROR in largewithlucas (now at top level) = interrupted}
Warning,  computation interrupted
```

▶ 3.30. Feladat.

3.31. Pépin-teszt.

```
> #
  # This procedure do the Pepin test
  # for the Fermat number 2^(2^m)+1.
  #

  pepin:=proc(m::nonnegint) local N;
  if m=0 then RETURN(true) fi;
  N:=2^(2^m)+1;
  if modp(3&^((N-1)/2),N)=N-1 then RETURN(true) fi;
  false end;
```

```
pepin := proc(m::nonnegint)                                   (3.31.1)
    local N;
    if m = 0 then
        RETURN(true)
    end if;
    N := 2^(2^m) + 1;
    if modp(3 &^ (1 / 2 * N − 1 / 2), N) = N − 1 then
        RETURN(true)
    end if;
    false
end proc
```

```
> pepin(4); pepin(5);
```

```
                              true
                              false                           (3.31.2)
```

3.32. Feladat.

3.32. Pocklington-Lehmer-teszt.

```
> #
  # This procedure do the Lehmer-Pocklington test
  # for the number n using the subset S of factors of
  # n-1 and base a. Returns with the reduced set S.
  #

  LPtest:=proc(n,S,a) local p,newS;
  if n<=a then ERROR(`last parameter is too large`,a) fi;
  if igcd(a,n)>1 then RETURN(false) fi;
```

```
   if modp(a&^(n-1),n)<>1 then RETURN(false) fi;
   newS:=S;
   for p in S do
     if igcd(modp(a&^((n-1)/p),n)-1,n)=1 then newS:=newS minus
   {p} fi
   od; newS; end;
```

```
LPtest := proc(n, S, a)                                              (3.33.1)
    local p, newS;
    if n <= a then
        ERROR(last parameter is too large, a)
    end if;
    if 1 < igcd(a, n) then
        RETURN(false)
    end if;
    if modp(a &^ (n − 1), n) <> 1 then
        RETURN(false)
    end if;
    newS := S;
    for p in S do
        if igcd(modp(a &^ ((n − 1) / p), n) − 1, n) = 1 then
            newS := minus(newS, {p})
        end if
    end do;
    newS
end proc
```

```
> #
  # This procedure try to find a large prime using
  # Lehmer-Pocklington test. The search goes for n,n+1,...
  between
  # numbers n*k^m+1.
  #

  largewithLP:=proc(n,k,m) local i,a,aa,S,S0;
  if k<2 then ERROR(`second parameter is too small`,k) fi;
  S0:=factorset(k);
  for i from n do
    if n<=k^m then S:=S0 else S:=S0 union factorset(i) fi;
    for a from 2 while S<>{} and S<>false do
      S:=LPtest(i*k^m+1,S,a);
```

```
        aa:=a;
      od;
      if S=false then
        print(i*k^m+1,aa,false)
      else
        print(i*k^m+1,aa,true)
      fi
    od; end;
```

```
largewithLP := proc(n, k, m)                                          (3.33.2)
    local i, a, aa, S, S0;
    if k < 2 then
        ERROR(second parameter is too small, k)
    end if;
    S0 := numtheory:-factorset(k);
    for i from n do
        if n <= k^m then
            S := S0
        else
            S := union(S0, numtheory:-factorset(i))
        end if;
        for a from 2 while S <> {} and S <> false do
            S := LPtest(i*k^m + 1, S, a);
            aa := a
        end do;
        if S = false then
            print(i*k^m + 1, aa, false)
        else
            print(i*k^m + 1, aa, true)
        end if
    end do
end proc
```

```
> debug(largewithLP); largewithLP(1,2,16);
                                   largewithLP
{--> enter largewithLP, args = 1, 2, 16
                                    S0 := {2}
                                    S := {2}
```

```
S:= {2}
aa:= 2
S:= {}
aa:= 3
65537, 3, true
S:= {2}
S:= false
aa:= 2
131073, 2, false
S:= {2}
S:= false
aa:= 2
196609, 2, false
S:= {2}
S:= false
aa:= 2
262145, 2, false
S:= {2}
S:= false
aa:= 2
327681, 2, false
S:= {2}
S:= false
aa:= 2
393217, 2, false
S:= {2}
S:= false
aa:= 2
458753, 2, false
S:= {2}
S:= false
aa:= 2
```

```
524289, 2, false
S:= {2}
S:= false
aa:= 2
589825, 2, false
S:= {2}
S:= false
aa:= 2
655361, 2, false
S:= {2}
S:= false
aa:= 2
720897, 2, false
S:= {2}
S:= {2}
aa:= 2
S:= {2}
aa:= 3
S:= {2}
aa:= 4
S:= {}
aa:= 5
786433, 5, true
S:= {2}
S:= false
aa:= 2
851969, 2, false
S:= {2}
S:= false
aa:= 2
917505, 2, false
S:= {2}
```

```
S:= false
aa:= 2
983041, 2, false
S:= {2}
S:= false
aa:= 2
1048577, 2, false
S:= {2}
S:= false
aa:= 2
1114113, 2, false
S:= {2}
S:= {2}
aa:= 2
S:= {2}
aa:= 3
S:= {2}
aa:= 4
S:= {2}
aa:= 5
S:= {2}
aa:= 6
S:= {2}
aa:= 7
S:= {2}
aa:= 8
S:= {2}
aa:= 9
S:= {2}
aa:= 10
S:= {2}
aa:= 11
```

```
S := {2}
aa := 12
S := {2}
aa := 13
S := {2}
aa := 14
S := {2}
aa := 15
S := {2}
aa := 16
S := {2}
aa := 17
S := {2}
aa := 18
S := {}
aa := 19
1179649, 19, true
S := {2}
S := false
aa := 2
1245185, 2, false
S := {2}
S := false
aa := 2
1310721, 2, false
S := {2}
S := {2}
aa := 2
S := {2}
aa := 3
S := {2}
aa := 4
```

```
S:= {}
aa:= 5
1376257, 5, true
S:= {2}
S:= false
aa:= 2
1441793, 2, false
S:= {2}
S:= false
aa:= 2
1507329, 2, false
S:= {2}
S:= false
aa:= 2
1572865, 2, false
S:= {2}
S:= false
aa:= 2
1638401, 2, false
S:= {2}
S:= false
aa:= 2
1703937, 2, false
S:= {2}
S:= {2}
aa:= 2
S:= {2}
aa:= 3
S:= {2}
<-- ERROR in largewithLP (now at top level) = interrupted}
Warning,  computation interrupted
> #
   # This procedure try to find factors of n having the form
   # a*k+b from k=1 until K or the square root of n.
# Note that LPtest capable to prove that factors n have
# the form f*k+1, where f is the factorised part of n-1,
# and a test treated later capable to prove that
# factors have the form f*k+1 or f*k-1, where f is the
# factorised part of n+1.
#

tryfactors:=proc(a::posint,b::integer,n::posint,K::posint)
local nn,i,f;
if a=1 then error "first parameter is too small",a fi;
if b<=1-a then error "second parameter is too small",b fi;
f:=[]; nn:=n;
for i to K while (i*a+b)^2<=nn do
  if modp(nn,i*a+b)=0 then
    f:=[op(f),i*a+b];
    nn:=nn/(i*a+b);
    i:=i-1;
  fi;
od; f end;
```

```
tryfactors := proc(a::posint, b::integer, n::posint, K::posint)                (3.33.3)
    local nn, i, f;
    if a = 1 then
        error "first parameter is too small", a
    end if;
    if b <= 1 - a then
        error "second parameter is too small", b
    end if;
    f := [ ];
    nn := n;
    for i to K while (a*i + b)^2 <= nn do
        if modp(nn, a*i + b) = 0 then
            f := [op(f), a*i + b];
            nn := nn / (a*i + b);
            i := i - 1
        end if
    end do;
    f
end proc
```

3.34. Feladat.

3.35. Feladat.

3.36. Proth-teszt.

3.37. Feladat.

```
> #
  # This procedure do the Proth test for the number n=h*2^m+1.
  #
  proth:=proc(h::posint,m::posint) local a,b,n; n:=h*2^m+1;
  if h>=2^m then error "first parmeter is too large",h fi;
  if not type(h,odd) then error "first parameter must be odd",h
  fi;
  if m=1 or m=2 then return true fi;
  if m=3 then if h=5 then return true else return false fi fi;
  # 2 is not a quadratic residue mod n, we start with 3
  # (2*a|n)=(a|n), hence enough to try odd bases a
  for a from 3 by 2 do
    b:=2&^m mod a; b:=h*b+1 mod a;
    # by quadratic reciprocity, (a|n)=(n|a)=(b|a)
    b:=modp(2&^m,a); b:=modp(h*b+1,a);
    if jacobi(b,a)=0 then return false fi;
    if jacobi(b,a)=-1 then
      if modp(a&^((n-1)/2),n)=n-1 then return true else return
  false fi;
    fi;
  od end;
```

```
proth := proc(h::posint, m::posint)                                (3.37.1)
    local a, b, n;
    n := h*2^m + 1;
    if 2^m <= h then
        error "first parmeter is too large", h
    end if;
    if not type(h, odd) then
        error "first parameter must be odd", h
    end if;
    if m = 1 or m = 2 then
        return true
    end if;
    if m = 3 then
        if h = 5 then
            return true
        else
            return false
        end if
    end if;
    for a from 3 by 2 do
        b := mod(2 &^ m, a);
        b := mod(h*b + 1, a);
        b := modp(2 &^ m, a);
        b := modp(h*b + 1, a);
        if numtheory:-jacobi(b, a) = 0 then
            return false
        end if;
        if numtheory:-jacobi(b, a) = −1 then
            if modp(a &^ (1 / 2 * n − 1 / 2), n) = n − 1 then
                return true
            else
                return false
            end if
        end if
    end do
end proc
```

```
> proth(11,5);
                              true                              (3.37.2)
```

```
> ifactor(11*2^5+1);
                              (353)                             (3.37.3)
```

3.38. Tétel.

```
> #
  # This test may be used if n-1=FR, where the factors
# of F are known, and there is given a bound B such that
# R has no primedivisor less then B, and FB>sqrt(N). First
# we have to use the Lehmer - Pocklington test with
# the set of divisors of F. After this the second step is
# to find an appropriate base a to R.
#

PLtestsecondstep:=proc(n,F,a) local p,newS;
if n<=a then ERROR(`last parameter is too large`,a) fi;
if igcd(a,n)>1 then RETURN(false) fi;
if modp(a&^(n-1),n)<>1 then RETURN(false) fi;
if igcd(modp(a&^F-1,n),n)=1 then RETURN(true) fi;
FAIL end;
```

```
PLtestsecondstep := proc(n, F, a)                                (3.38.1)
    local p, newS;
    if n <= a then
        ERROR(last parameter is too large, a)
    end if;
    if 1 < igcd(a, n) then
        RETURN(false)
    end if;
    if modp(a &^ (n − 1), n) <> 1 then
        RETURN(false)
    end if;
    if igcd(modp(a &^ F − 1, n), n) = 1 then
        RETURN(true)
    end if;
    FAIL
end proc
```

▶ 3.39. Feladat.

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