Számítógépes számelmélet

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

1. A prímek eloszlása, szitálás

2. Egyszerű faktorizálási módszerek

```
> 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*] (2.1)

2.1. Próbaosztás.

```
> #
  # This is a simple factorization
  # procedure using trial division.
  # The result is a sequence of pairs
  # [p,e] where the p's are the prime
  # factors and the e's are the exponents.
  # The factors are anyway in increasing order.
  # Only primes <= P are tried, hence the
  # last "factor" may composite, if
  # it is greater then P^2;
  #

  trialdiv:=proc(n::posint,P::posint) local L,p,i,d,nn;
  L:=[]; nn:=n;
  if type(nn,even) and 2<=P then
    for i from 0 while type(nn,even) do nn:=nn/2; od;
    L:=[[2,i]];
  fi;
  if nn mod 3=0 and 3<=P then
    for i from 0 while nn mod 3=0 do nn:=nn/3; od;
    L:=[op(L),[3,i]];
```

```
  fi;
  d:=2; p:=5;
  while p<=P and nn>=p^2 do
    if nn mod p=0 then
      for i from 0 while nn mod p=0 do nn:=nn/p; od;
      L:=[op(L),[p,i]];
    fi;
    p:=p+d; d:=6-d;
  od;
  if nn>1 then L:=[op(L),[nn,1]] fi;
  L;
  end;
```

```
trialdiv := proc(n::posint, P::posint)                                  (2.1.1)
    local L, p, i, d, nn;
    L := [ ];
    nn := n;
    if type(nn, even) and 2 <= P then
        for i from 0 while type(nn, even) do
            nn := 1 / 2 * nn
        end do;
        L := [[2, i]]
    end if;
    if mod(nn, 3) = 0 and 3 <= P then
        for i from 0 while mod(nn, 3) = 0 do
            nn := 1 / 3 * nn
        end do;
        L := [op(L), [3, i]]
    end if;
    d := 2;
    p := 5;
    while p <= P and p^2 <= nn do
        if mod(nn, p) = 0 then
            for i from 0 while mod(nn, p) = 0 do
                nn := nn / p
            end do;
            L := [op(L), [p, i]]
```

```
        end if;
        p := p + d;
        d := 6 - d
    end do;
    if 1 < nn then
        L := [op(L), [nn, 1]]
    end if;
    L
end proc
```

```
> trialdiv(2^32+1,1000);
```

$$[[641, 1], [6700417, 1]] \tag{2.1.2}$$

```
> #
  # This is a simple primality testing
  # procedure using trial division.
  # The result is true if n is prime, else false.
  #

  trialprime:=proc(n::posint) local L,p,i,d,nn;
  if n=1 then RETURN(false) fi;
  if n=2 or n=3 or n=5 then RETURN(true) fi;
  if type(n,even) then RETURN(false) fi;
  if n mod 3=0 then RETURN(false) fi;
  d:=2; p:=5;
  while n>=p^2 do
    if n mod p=0 then RETURN(false) fi;
    p:=p+d; d:=6-d;
  od; true; end;
```

$$trialprime := \mathbf{proc}(n::posint) \tag{2.1.3}$$

```
    local L, p, i, d, nn;
    if n = 1 then
        RETURN(false)
    end if;
    if n = 2 or n = 3 or n = 5 then
        RETURN(true)
    end if;
    if type(n, even) then
        RETURN(false)
    end if;
```

```
    if mod(n, 3) = 0 then
        RETURN(false)
    end if;
    d := 2;
    p := 5;
    while p^2 <= n do
        if mod(n, p) = 0 then
            RETURN(false)
        end if;
        p := p + d;
        d := 6 - d
    end do;
    true
end proc
```

```
> trialprime(2^32+1);
```

$$false \tag{2.1.4}$$

▶ 2.2. Feladat.

▼ 2.3. Feladat.

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

$$1 \tag{2.3.1}$$

```
> print(ifactor);
```

$$\mathbf{proc}(n) \ldots \mathbf{end\ proc} \tag{2.3.2}$$

```
> print(`ifactor/ifact235`);
```

$$\mathbf{proc}(n) \ldots \mathbf{end\ proc} \tag{2.3.3}$$

```
> print(`ifactor/ifact0th`);
```

$$\mathbf{proc}(n) \ldots \mathbf{end\ proc} \tag{2.3.4}$$

```
> print(`ifactor/ifact1st`);
```

$$\mathbf{proc}(n) \ldots \mathbf{end\ proc} \tag{2.3.5}$$

```
> print(`ifactor/wheelfact`);
```

$$\mathbf{proc}(n, s) \ldots \mathbf{end\ proc} \tag{2.3.6}$$

```
> print(`ifactor/pollp1`);
```

$$\mathbf{proc}(n, seed) \ldots \mathbf{end\ proc} \tag{2.3.7}$$

```
> print(`ifactor/pp100000`);
```

proc(n) ... end proc (2.3.8)

```
> print(`ifactor/easy`);
```

proc(n) ... end proc (2.3.9)

```
> print(`ifactor/ifact235`);
```

proc(n) ... end proc (2.3.10)

2.4. A prímosztók eloszlásáról.

2.5. A prímosztók számának határeloszlása.

2.6. Fermat módszere.

```
> #
  # This procedure prepare the sieve table S for
  # Fermat's factorization procedure. Parameter n is the
  # integer to factor and m is the vector of moduli.
  #

  preparefermatsieve:=proc(n,S,m) local i,j,k,x2,x2n;
  x2:=table; x2n:=table;
  for i to nops(m) do
    for j from 0 to m[i]-1 do S[i,j]:=0; od;
    for j from 0 to m[i]-1 do
      x2[j]:=j^2 mod m[i];
      x2n[j]:=j^2-n mod m[i];
    od;
    for j from 0 to m[i]-1 do
      for k from 0 to m[i]-1 do
        if x2n[j]=x2[k] then S[i,j]:=1 fi;
      od;
    od;
  od; end;
```

preparefermatsieve := proc(n, S, m) (2.6.1)

local i, j, k, x2, x2n;

x2 := table;

x2n := table;

for i to nops(m) do

for j from 0 to m[i] − 1 do

S[i, j] := 0

end do;

```
        for j from 0 to m[i] - 1 do
            x2[j] := mod(j^2, m[i]);
            x2n[j] := mod(j^2 - n, m[i])
        end do;
        for j from 0 to m[i] - 1 do
            for k from 0 to m[i] - 1 do
                if x2n[j] = x2[k] then
                    S[i, j] := 1
                end if
            end do
        end do
    end do
end proc
```

```
> #
  # This procedure do factorization with
  # Fermat's method. Parameter n is
  # the odd number to factor and m is the list of moduli.
  # Returns with u where u is the largest
  # factor of n less then or equal to sqrt(n).
  #

  fermatfactorization:=proc(n::posint,m::list(posint))
  local k,x,y,i,S,r,f;
  if type(n,even) then error "first argument must be odd" fi;
  S:=table(); preparefermatsieve(n,S,m); r:=nops(m);
  k:=array(1..r); x:=isqrt(n);
  for i to r do k[i]:=-x mod m[i]; od;
  while true do
    f:=true;
    for i to r do if S[i,k[i]]<>1 then f:=false; break; fi; od;
    if f then
      y:=isqrt(x^2-n);
      if y^2=x^2-n then RETURN(x-y) fi;
    fi;
    x:=x+1;
    for i to r do k[i]:=k[i]-1 mod m[i]; od;
  od; end;
```

```
fermatfactorization := proc(n::posint, m::(list(posint)))
    local k, x, y, i, S, r, f;
    if type(n, even) then
```

(2.6.2)

```
        error "first argument must be odd"
    end if;
    S := table( );
    preparefermatsieve(n, S, m);
    r := nops(m);
    k := array(1..r);
    x := isqrt(n);
    for i to r do
        k[i] := mod(-x, m[i])
    end do;
    do
        f := true;
        for i to r do
            if S[i, k[i]] <> 1 then
                f := false;
                break
            end if
        end do;
        if f then
            y := isqrt(x^2 - n);
            if y^2 = x^2 - n then
                RETURN(x - y)
            end if
        end if;
        x := x + 1;
        for i to r do
            k[i] := mod(k[i] - 1, m[i])
        end do
    end do
end proc
```

```
> debug(fermatfactorization);
  fermatfactorization(13*17,[3,5,7,8,11]);
                          fermatfactorization
```

```
{--> enter fermatfactorization, args = 221, [3, 5, 7, 8,
11]
```

$$S := table([\,])$$

$$r := 5$$

$$k := array(1..5, [\,])$$

$$x := 15$$

$$k_1 := 0$$

$$k_2 := 0$$

$$k_3 := 6$$

$$k_4 := 1$$

$$k_5 := 7$$

$$f := true$$

$$y := 2$$

```
<-- exit fermatfactorization (now at top level) = 13}
```

$$13 \tag{2.6.3}$$

```
> undebug(fermatfactorization);
  fermatfactorization(11111,[3,5,7,8,11]);
```

$$fermatfactorization$$

$$41 \tag{2.6.4}$$

▶ 2.7. Feladat.

▶ 2.8. Feladat.

▼ 2.9. Pollard ϱ módszere.

Az n szám hasítása az x->x^2+c függvény felhasználásával; g egy iterációcsoport mérete és legfeljebb maxgs csoport fog végrehajtódni.

```
> pollardrhosplit:=proc(n::posint,c::posint,g::posint,
  maxgs::posint)
  local x,xx,xp,xo,xpo,i,j,k,ko,l,lo;
  x:=1+c mod n; xp:=1; i:=0; k:=1; l:=1; xx:=1;
  while igcd(xx,n)=1 and i<maxgs do
    xo:=x; xpo:=xp; ko:=k; lo:=l; j:=0; xx:=1;
    while j<g do
      xx:=xx*(xp-x) mod n;
      k:=k-1; if k=0 then xp:=x; k:=l; l:=2*l; fi;
      x:=x^2+c mod n; j:=j+1;
```

```
    od; i:=i+1;
  od;
  if igcd(xx,n)<n then return(igcd(xx,n)) fi;
  x:=xo; xp:=xpo; k:=ko; l:=lo; j:=0;
  while igcd(xp-x,n)=1 and j<g do
    k:=k-1; if k=0 then xp:=x; k:=l; l:=2*l; fi;
    x:=x^2+c mod n; j:=j+1;
  od;
  igcd(xp-x,n); end;
```

$$pollardrhosplit := \mathbf{proc}(n::posint,\ c::posint,\ g::posint,\ maxgs::posint) \tag{2.9.1}$$

local $x,\ xx,\ xp,\ xo,\ xpo,\ i,\ j,\ k,\ ko,\ l,\ lo;$

$x := mod(1 + c,\ n);$

$xp := 1;$

$i := 0;$

$k := 1;$

$l := 1;$

$xx := 1;$

while $igcd(xx,\ n) = 1$ **and** $i < maxgs$ **do**

$xo := x;$

$xpo := xp;$

$ko := k;$

$lo := l;$

$j := 0;$

$xx := 1;$

while $j < g$ **do**

$xx := mod(xx * (xp - x),\ n);$

$k := k - 1;$

if $k = 0$ **then**

$xp := x;$

$k := l;$

$l := 2 * l$

end if;

$x := mod(x\text{^}2 + c,\ n);$

$j := j + 1$

end do;

```
        i := i + 1
    end do;
    if igcd(xx, n) < n then
        return igcd(xx, n)
    end if;
    x := xo;
    xp := xpo;
    k := ko;
    l := lo;
    j := 0;
    while igcd(xp - x, n) = 1 and j < g do
        k := k - 1;
        if k = 0 then
            xp := x;
            k := l;
            l := 2 * l
        end if;
        x := mod(x^2 + c, n);
        j := j + 1
    end do;
    igcd(xp - x, n)
end proc
```

```
> pollardrhosplit(999863*999883,1,2^4,2^5);
```

999863 (2.9.2)

▶ 2.10. Feladat.

▶ 2.11. Fermat tétele.

▶ 2.12. Euler tétele.

▼ 2.13. Kínai maradéktétel.

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

23 (2.13.1)

▶ 2.14. Tétel.

▼ 2.15. Gyors hatványozás.

```
> #
  # Calculation of modular power of a
  # with the left-to-right binary method.
  #

  left2right:=proc(a,e::posint,mult::procedure) local b,x,n;
  b:=convert(e,base,2); x:=a;
  for n from nops(b)-1 by -1 to 1 do
    x:=mult(x,x);
    if b[n]>0 then x:=mult(x,a); fi;
  od; x; end;
```

$$\mathit{left2right} := \mathbf{proc}(a, e::\mathit{posint}, \mathit{mult}::\mathit{procedure}) \tag{2.15.1}$$

$\mathbf{local}\ b, x, n;$

$b := \mathit{convert}(e, \mathit{base}, 2);$

$x := a;$

$\mathbf{for}\ n\ \mathbf{from}\ \mathit{nops}(b) - 1\ \mathbf{by} - 1\ \mathbf{to}\ 1\ \mathbf{do}$

$x := \mathit{mult}(x, x);$

$\mathbf{if}\ 0 < b[n]\ \mathbf{then}$

$x := \mathit{mult}(x, a)$

$\mathbf{end\ if}$

$\mathbf{end\ do};$

x

$\mathbf{end\ proc}$

```
> debug(left2right); left2right(2,11,(x,y)->x*y);
```

$$\mathit{left2right}$$

```
{--> enter left2right, args = 2, 11, proc (x, y) options
operator, arrow; x*y end proc
```

$$b := [1, 1, 0, 1]$$

$$x := 2$$

$$x := 4$$

$$x := 16$$

$$x := 32$$

$$x := 1024$$

$$x := 2048$$

2048

```
<-- exit left2right (now at top level) = 2048}
```

2048 (2.15.2)

```
> #
  # Calculation of a modular power
  # with the left-to-right 2^m-ary method.
  #

  fastexp:=proc(a,e::posint,m::posint,mult::procedure)
  local i,j,k,P,x,b,aa,n;
  b:=convert(e,base,2); n:=nops(b)-1; x:=a; P:=[a]; aa:=mult(a,
  a);
  for j from 2 to 2^(m-1) do P:=[op(P),mult(P[nops(P)],aa)];
  od;
  while true do
    if n=0 then return(x) fi;
    if b[n]=0 then x:=mult(x,x); n:=n-1; next; fi;
    i:=1; j:=1; k:=0; x:=mult(x,x); n:=n-1;
    while n>0 and k+j<m do
      if b[n]=0 then k:=k+1; n:=n-1;
      else k:=k+1; j:=j+k; i:=i*2^k;
        while k>0 do x:=mult(x,x); k:=k-1; od;
        n:=n-1;
      fi;
    od;
    x:=mult(x,P[i]);
    while k>0 do x:=mult(x,x); k:=k-1; od;
  od; end;
```

$fastexp := \mathbf{proc}(a, e::posint, m::posint, mult::procedure)$ (2.15.3)

local i, j, k, P, x, b, aa, n;

$b := convert(e, base, 2)$;

$n := nops(b) - 1$;

$x := a$;

$P := [a]$;

$aa := mult(a, a)$;

for j **from** 2 **to** $2^{(m-1)}$ **do**

$P := [op(P), mult(P[nops(P)], aa)]$

end do;

do

if $n = 0$ **then**

return x

```
        end if;
        if b[n] = 0 then
            x := mult(x, x);
            n := n − 1;
            next
        end if;
        i := 1;
        j := 1;
        k := 0;
        x := mult(x, x);
        n := n − 1;
        while 0 < n and k + j < m do
            if b[n] = 0 then
                k := k + 1;
                n := n − 1
            else
                k := k + 1;
                j := k + j;
                i := i * 2 ^ k;
                while 0 < k do
                    x := mult(x, x);
                    k := k − 1
                end do;
                n := n − 1
            end if
        end do;
        x := mult(x, P[i]);
        while 0 < k do
            x := mult(x, x);
            k := k − 1
        end do
    end do
```

end proc

```
> debug(fastexp); fastexp(2,11,1,(x,y)->x*y);
```

fastexp

```
{--> enter fastexp, args = 2, 11, 1, proc (x, y) options
operator, arrow; x*y end proc
```

$b := [1, 1, 0, 1]$

$n := 3$

$x := 2$

$P := [2]$

$aa := 4$

$x := 4$

$n := 2$

$i := 1$

$j := 1$

$k := 0$

$x := 16$

$n := 1$

$x := 32$

$i := 1$

$j := 1$

$k := 0$

$x := 1024$

$n := 0$

$x := 2048$

```
<-- exit fastexp (now at top level) = 2048}
```

2048 (2.15.4)

```
> debug(fastexp); fastexp(2,11,2,(x,y)->x*y);
```

fastexp

```
{--> enter fastexp, args = 2, 11, 2, proc (x, y) options
operator, arrow; x*y end proc
```

$b := [1, 1, 0, 1]$

$n := 3$

$x := 2$

$P := [2]$

$aa := 4$

$$P := [2, 8]$$

$$x := 4$$

$$n := 2$$

$$i := 1$$

$$j := 1$$

$$k := 0$$

$$x := 16$$

$$n := 1$$

$$k := 1$$

$$j := 2$$

$$i := 2$$

$$x := 256$$

$$k := 0$$

$$n := 0$$

$$x := 2048$$

```
<-- exit fastexp (now at top level) = 2048}
```

$$2048 \tag{2.15.5}$$

▶ 2.16. Feladat.

▼ 2.17. Pollard p–1 módszere.

```
> #
  # This procedure is Pollard's p-1 method for
  # factorization. The base is a, and powers of
  # primes up to P are considered so that they
  # are not less then the bound B.
  # The result is the power x of a mod n,  where
  # n isthe number to factorize, so the factor is gcd(x-1,n).
  #

  pollardpsplit:=proc(n,a,B,P) local e,d,p,x;
  x:=a mod n;
  if igcd(x-1,n)>1 or P<2 then return(x) fi;
  if P<2 then return(x) fi;
  e:=1; while 2^e<B do e:=e+1 od;
  x:=x&^(2^e) mod n;
  if igcd(x-1,n)>1 or P=2 then return(x) fi;
  while 3^e>3*B and e>1 do e:=e-1 od;
```

```
   x:=x&^(3^e) mod n;
   d:=2; p:=5;
   while true do
     if igcd(x-1,n)>1 or P<p then return(x) fi;
     while p^e>p*B and e>1 do e:=e-1 od;
     x:=x&^(p^e) mod n;
     p:=p+d; d:=6-d;
   od; x; end;
```

$$pollardpsplit := \mathbf{proc}(n, a, B, P) \tag{2.17.1}$$

local e, d, p, x;
$x := mod(a, n)$;
if $1 < igcd(x-1, n)$ **or** $P < 2$ **then**
return x
end if;
if $P < 2$ **then**
return x
end if;
$e := 1$;
while $2\hat{}e < B$ **do**
$e := e + 1$
end do;
$x := mod(x\,\&\hat{}\,(2\hat{}e), n)$;
if $1 < igcd(x-1, n)$ **or** $P = 2$ **then**
return x
end if;
while $3 * B < 3\hat{}e$ **and** $1 < e$ **do**
$e := e - 1$
end do;
$x := mod(x\,\&\hat{}\,(3\hat{}e), n)$;
$d := 2$;
$p := 5$;
do
if $1 < igcd(x-1, n)$ **or** $P < p$ **then**
return x
end if;

```
        while p*B < p^e and 1 < e do
            e := e - 1
        end do;
        x := mod(x &^ (p^e), n);
        p := p + d;
        d := 6 - d
    end do;
    x
end proc
```

```
> pollardpsplit(25852,2,100,100);
                              23324                              (2.17.2)

> igcd(%-1,25852);
                               281                               (2.17.3)

> pollardpsplit(999863*999917*999961,23,2000,1000);
                        16252910338466315                        (2.17.4)

> igcd(%-1,999863*999917*999961);
                              999917                             (2.17.5)
```

▶ 2.18. Feladat.

▼ 2.19. Pollard p–1 módszere, második lépcső.

```
> #
  # This procedure is the second step of Pollard's p-1 method
  for
  # factorization. The base is a, and primes from list P
  # are considered. The result is the power x of a mod n,
  where
  # n is the number to factorize, so the factor is gcd(x-1,n).
  #

  pollardp2split:=proc(n::posint,a::posint,N::posint,m::posint,
  M::posint)
  local x,i,j,E,aa,p,pp,d;
  E:=Array(1..N); aa:=a*a mod n; E[1]:=aa;
  for j from 2 to N do E[j]:=E[j-1]*aa od;
  p:=ithprime(m); x:=a&^p mod n;
  for i from m+1 to M while gcd(x-1,n)=1 do
    pp:=nextprime(p); d:=pp-p; p:=pp;
    if d<=2*N then x:=x*E[d/2] mod n;
```

```
    else x:=x*(a&^d) mod n; fi;
  od; x; end;
```

$$pollardp2split := \mathbf{proc}(n::posint,\ a::posint,\ N::posint,\ m::posint,\ M::posint) \tag{2.19.1}$$

```
    local x, i, j, E, aa, p, pp, d;
    E:= Array(1..N);
    aa:= mod(a*a, n);
    E[1]:= aa;
    for j from 2 to N do
        E[j]:= E[j-1]*aa
    end do;
    p:= ithprime(m);
    x:= mod(a &^ p, n);
    for i from m+1 to M while gcd(x-1, n)=1 do
        pp:= nextprime(p);
        d:= pp-p;
        p:= pp;
        if d<=2*N then
            x:= mod(x*E[1/2*d], n)
        else
            x:= mod(x*a &^ d, n)
        end if
    end do;
    x
end proc
```

```
> pollardpsplit(8174912477117*23528569104401,3,1000,1000);
```

$$146645799608527753237179827 \tag{2.19.2}$$

```
> igcd(%-1,8174912477117*23528569104401);
```

$$1 \tag{2.19.3}$$

```
> pollardp2split(8174912477117*23528569104401,%%,100,100,10000)
  ;
```

$$391453341919440359125466 6 \tag{2.19.4}$$

```
> igcd(%-1,8174912477117*23528569104401);
```

$$23528569104401 \tag{2.19.5}$$

```
> ifactor(%-1);
```

(2.19.6)

$$(2)^4\,(5)^2\,(67)\,(107)\,(199)\,(41231) \tag{2.19.6}$$

▶ **2.20. Feladat.**

▶ **3. Egyszerű prímtesztelési módszerek**

▶ **4. Lucas–sorozatok**

▶ **5. Alkalmazások**

▶ **6. Számok és polinomok**

▶ **7. Gyors Fourier–transzformáció**

▶ **8. Elliptikus függvények**

▶ **9. Számolás elliptikus görbéken**

▶ **10. Faktorizálás elliptikus görbékkel**

▶ **11. Prímteszt elliptikus görbékkel**

▶ **12. Polinomfaktorizálás**

▶ **13. Az AKS teszt**

▶ **14. A szita módszerek alapjai**