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*, *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)

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*::*posint*) (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;
```

$$n := 38347921$$

$$1$$

$$10225489 \quad (3.4.1)$$

▶ 3.5. Feladat.

▶ 3.6. Feladat.

▶ 3.7. Feladat.

▶ 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}
```

$$-1 \qquad (3.18.2)$$

3.19. Feladat.

3.20. Feladat.

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

$$1 \qquad (3.20.1)$$

```
> print(jacobi);
```

proc(*a*::(Or(*integer*, Not(*constant*))), *b*::(Or(*nonnegint*, Not(*constant*)))) (3.20.2)

...

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);
```

$$n := 561$$
$$1$$
$$67$$
$$1 \tag{3.22.1}$$

▶ 3.23. Feladat.

▼ 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.

▼ 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
```

3.30. Feladat.

3.31. Pépin-teszt.

3.32. Feladat.

3.33. Pocklington-Lehmer-teszt.

3.34. Feladat.

3.35. Feladat.

3.36. Proth-teszt.

3.37. Feladat.

```
> proth:=proc(h::posint,m::posint) local a,b,n,ad; n:=h*2^m+1;
    if h>=2^m or not type(h,odd) then return FAIL fi;
    if m=1 or m=2 then return true fi;
    # 2 is not a quadratic residue mod n, we start with 3
    a:=3; b:=2&^m mod a; b:=h*b+1 mod a;
    # by quadratic reciprocity, (a|n)=(n|a)=(b|a)
    if jacobi(b,a)=0 then return false fi;
    if jacobi(b,a)=-1 then
      if a&^((n-1)/2) mod n=n-1 then return true else return
  false fi;
    fi;
    # we shall use the quasi-prime sequence 5,7,11,13,17,19,23,
  25,..
    # a random base would give only two trials in mean,
    # and this looks even better
    a:=5; ad:=2;
    while true do
      b:=2&^m mod a; b:=h*b+1 mod a;
      if jacobi(b,a)=0 then return false fi;
      if jacobi(b,a)=-1 then
        if a&^((n-1)/2) mod n=n-1 then return true else return
```

```
false fi;
    fi;
    a:=a+ad; ad:=6-ad;
  od;
end;
```

proth := **proc**(h::posint, m::posint) (3.37.1)

```
    local a, b, n, ad;
    n := h*2^m + 1;
    if 2^m <= h or not type(h, odd) then
        return FAIL
    end if;
    if m = 1 or m = 2 then
        return true
    end if;
    a := 3;
    b := mod(2 &^ m, a);
    b := mod(h*b + 1, a);
    if numtheory:-jacobi(b, a) = 0 then
        return false
    end if;
    if numtheory:-jacobi(b, a) = -1 then
        if mod(a &^ (1/2*n - 1/2), n) = n - 1 then
            return true
        else
            return false
        end if
    end if;
    a := 5;
    ad := 2;
    do
        b := mod(2 &^ m, a);
        b := mod(h*b + 1, a);
        if numtheory:-jacobi(b, a) = 0 then
```

```
            return false
        end if;
        if numtheory:-jacobi(b, a) = -1 then
            if mod(a &^ (1 / 2 * n - 1 / 2), n) = n - 1 then
                return true
            else
                return false
            end if
        end if;
        a := a + ad;
        ad := 6 - ad
    end do
end proc
```

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

$$true \tag{3.37.2}$$

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

$$(353) \tag{3.37.3}$$

▶ 3.38. Tétel.

▶ 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. A szita módszerek alapjai**
- ▶ **14. Az AKS teszt**