Számítógépes számelmélet

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

1. A prímek eloszlása, szitálás

2. Egyszerű faktorizálási módszerek

```
> restart;
```

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 (2.1.4)

▶ 2.2. Feladat.

▼ 2.3. Feladat.

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

1 (2.3.1)

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

proc(*n*) ... **end proc** (2.3.2)

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

proc(*n*) ... **end proc** (2.3.3)

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

proc(*n*) ... **end proc** (2.3.4)

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

proc(*n*) ... **end proc** (2.3.5)

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

proc(*n*, *s*) ... **end proc** (2.3.6)

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

proc(*n*, *seed*) ... **end proc** (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)

```
> interface(echo=3);
```

(2.3.11)

```
> read("../../../tmp/ifactor");
#
# This is a commented version of the Maple procedure
# ifactor for analysis of the methods.
#
# The input parameter n in general a positive integer.
#
# After a parameter cheking other cases: negative
# integer, rational numbers, etc. are treated.
#
# The first step is remove factors 2, 3, 5,7, 11, 13.
# gcd(n,2^4*3^2*5*7*11*13) calculated until became 1.
# If it is greater then 1, then this gcd is factored by
# the small routine ifactor/ifact235.
#
# The second step is remove somewhat larger prime factors below
# from 17 to 1699. This also happens with the gcd trick. Until
the
# gcd is larger then 1, the routine ifactor/ifact1st find these
# factors from the gcd.
#
# If the remainder still greater then 1, then the procedure
# corresponding to the second parameter is loaded until the name
# `ifactor/bottom` and the procedure ifactor/ifact0th called
# with passing thierd and further parameters.
# This procedure use `ifactor/bottom`.
#
# If no second parameter is passed, then the `ifactor/morrbril`
# procedure loaded and the procedure ifactor/ifact0th called.
#
>
> ifactor=proc(n)
> local sol,r;
> global `ifactor/bottom`;
> options remember,system,`Copyright 1993 by Waterloo Maple
Software`;
>     if nargs < 1 or 1 < nargs and not type(args[2],name) then
>         ERROR(`invalid arguments`)
>     fi;
>     if type(n,integer) then
>         if 0 < n then sol := 1; r := n
>         elif n < 0 then sol := -1; r := -n
>         else RETURN(0)
>         fi
>     elif type(n,fraction) then RETURN(ifactor(op(1,n))/ifactor
(op(2,n)))
>     elif type(n,{`*`,set,list,relation}) then RETURN(map
```

```
(ifactor,n))
>     elif type(n,`^`) and type(op(2,n),integer) then
>         RETURN(ifactor(op(1,n))^op(2,n))
>     elif type(n,``(integer)) then RETURN(ifactor(op(1,n)))
>     else ERROR(`invalid arguments`)
>     fi;
>     igcd(r,720720);
>     while % <> 1 do
>         sol := sol*`ifactor/ifact235`(%); r := iquo(r,%%);
igcd(%%%,r)
>     od;
>     igcd(r,
116542951143701442118256425539411806278705518107369035248436272442
929886551970895780845374261270209629822427348440139234569670132540
666860138774356183909763696443067069059862065192038988478418219080
294387922301947266843024378112299030349298539707741678899215627540
549809976535794519247972221385722721200060812856017119765934942419
619611716722790220172291221277043483457977929715054446536085051102
596639253777494582990197958881314319454754358382573986765277067020
999834807973358746434146103434351501879897281549965776192088074890
382147598111758441747971820027670613338896899878156111058787897300
794721580125932049284399286767844431923823398185219113121941888470
578706603100075404043208113667334249021102156850452413232502897090
233);
>     while % <> 1 do
>         sol := sol*`ifactor/ifact1st`(%); r := iquo(r,%%);
igcd(%%%,r)
>     od;
>     if r <> 1 then
>         if nargs = 1 then
>             `ifactor/bottom` := readlib(``ifactor/morrbril`');
>             `ifactor/ifact0th`(r)
>         else
>             `ifactor/bottom` := readlib(`ifactor/`.args[2]);
>             `ifactor/ifact0th`(r,args[3 .. nargs])
>         fi;
>         if % <> FAIL then sol*% else FAIL fi
>     else sol
>     fi
> end;
```

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

```
>
>
#
# This small routine find the factors of
# a positive integer known to have the
# gcd of 2^4*3^2*5*7*11*13 and the number
# to factor. The factorization is given back.
```

```
#
>
> ifactor/ifact235=proc(n)
> local q;
> options remember,system,`Copyright 1993 by Waterloo Maple
Software`;
>     if n = 1 then 1
>     elif irem(n,13,'q') = 0 then `ifactor/ifact235`(q)*``(13)
>     elif irem(n,11,'q') = 0 then `ifactor/ifact235`(q)*``(11)
>     elif irem(n,7,'q') = 0 then `ifactor/ifact235`(q)*``(7)
>     elif irem(n,2,'q') = 0 then ``(2)*`ifactor/ifact235`(q)
>     elif irem(n,3,'q') = 0 then ``(3)*`ifactor/ifact235`(q)
>     else ``(5)
>     fi
> end;
```

$$\frac{ifactor}{ifact235} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This routine find the factors from 17 to 1699. The
# input is a positive integer containing only factors
# between 17 and 1699.
#
# The routine divides first the factors into
# smaller groups 17--19, 23--37, 41--67,
# 71--137, 139-281, 283--659, 661--1699 using the gcd trick.
# Then, depending on the gcd obtained, a search routine
# ifactor/wheelfact is started. The search limit is
# choosen supposing two factors find: 17*19, 23*29, 41*43,
# 71*73, 139*149, 283*293, 661*673.
#
>
> ifactor/ifact1st=proc(n)
> local i,ig,r,sol;
> options remember,system,`Copyright 1993 by Waterloo Maple
Software`;
>     sol := 1;
>     r := n;
>     for i to 7 while r <> 1 do
>         ig := igcd(r,(323,765049,1058967640189,
916834684886440380235864 1659,
>
342104831177853836844000918542910563842436194397212591435983889,
792709889190322888122151467214412384719082407423339586081109312 11
993759168976599935600233699017567563522606409741742682779591373 41
840564310562263449502011246389,
179121216345076044215350912828530796966835016565315142254527442 69
```

```
6632957708868116350808841702657561144525143196579319864354269715960674364042924779291399178564036467383384545401413952549356161834532756441918356224214016169612042405530047600574836123437652470013895386096822766977086159615663835095759274694220644554276246948965989658475426719075582366414921545743413548876681972015541364188265006676517759651237727461023124369799987063929201186649
>         )[i]);
>         if ig <> 1 then
>             r := iquo(r,ig);
>             while [323,667,1763,5183,20711,82919,444853][i] <=
ig do
>                 `ifactor/wheelfact`(ig,[17,23,41,71,139,283,
661][i]);
>                 sol := sol*``(%);
>                 ig := iquo(ig,%%)
>             od;
>             sol := sol*``(ig)
>         fi
>     od;
>     sol
> end;
```

$$\frac{ifactor}{ifact1st} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This routine find the prime factors with searcing.
# The input is a positive integer known to have only factor
# between 17 and 1699.
#
# The "large searching" here goes using gcd too, using
# intervals with length 30. The find gcd's larger then 1
# finally factored by a smaller search trying odd numbers.
#
>
> ifactor/wheelfact=proc(n,s)
> local i,ii;
> options `Copyright 1993 by Waterloo Maple Software`;
>     for i from 30*iquo(s,30)+15 by 30 do
>         i^2;
>         if igcd(n,(%-4)*(%-16)*(%-64)*(%-196)) <> 1 then
>             for ii from i-14 by 2 do
>                 if igcd(ii,n) <> 1 then RETURN(ii) fi
>             od
>         fi
>     od
> end;
```

$$\frac{ifactor}{wheelfact} = \mathbf{proc}(n, s) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This is the general end-factoring routine.
# The input is a positive integer containing no
# factor below 1700.
#
# If the number is less then 1709^2, then it is
# a prime and simple returned. Otherwise the number is
# tested and if it is found to be prime, returned.
#
# The second step is to try whether the numbers is
# a power using the routine ifactor/power.
#
# The thierd step is to try Pollard's p-1 method.
# If there are chanches then tried again.
# After the second failer or if no chances
# another version called ifactor/pp100000
# called to find factors p for which p-1
# has only factors below 100000.
#
# The last try is the procedure ifactor/bottom,
# which depends on the second input parameter.
#
# Finally, if no factor is found, then the
# routine returns with fail. Otherwise if is called
# for the factors found and returns with the product.
#
>
> ifactor/ifact0th=proc(n)
> local sol,p;
> options `Copyright 1993 by Waterloo Maple Software`;
>     if n < 2920681 then RETURN(``(n)) fi;
>     if isprime(n) then RETURN(``(n)) fi;
>     sol := `ifactor/power`(n,'p');
>     if sol <> FAIL then RETURN(`ifactor/ifact0th`(sol)^p) fi;
>     sol := `ifactor/pollp1`(n,13003);
>     if sol = _tryagain then sol := `ifactor/pollp1`(n,13004)
fi;
>     if sol = _tryagain then sol := FAIL fi;
>     if sol = FAIL then sol := `ifactor/pp100000`(n) fi;
>     if sol = FAIL then
>         sol := `ifactor/bottom`(args);
>         if not type(sol,integer) then RETURN(sol) fi
>     fi;
>     `ifactor/ifact0th`(n/sol,args[2 .. nargs])*
```

```
>          `ifactor/ifact0th`(sol,args[2 .. nargs])
> end;
```

$$\frac{ifactor}{ifact0th} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This small procedure check whether the input
# positive number is a prime power. It is known
# that there are no too small factors. The factor
# is given back and the exponent in the parameter
# 'p'.
#
>
> ifactor/power:=proc(n,p)
> local i,r;
> options `Copyright 1993 by Waterloo Maple Software`;
>     r := isqrt(n);
>     if r^2 = n then p := 2; RETURN(r) fi;
>     for i from 3 by 2 to iquo(length(n)+2,3) do
>         if not isprime(i) then next fi;
>         r := readlib('iroot')(n,i);
>         if r^i = n then p := i; RETURN(r) fi
>     od;
>     FAIL
> end;
Error, invalid left hand side of assignment
>
>
#
# This procedure try to find factors of the input
# positive integer n using Pollard's p-1 method.
# The second parameter is the base to find the
# factors.
#
# The base is in a cycle powered gradually to 2^9,
# 3^6*5^4*7^2*11^2, 7*11*13^2*31^2*1229,
# 17^2*19^2*23^2*37*41*263, 29^2*43*47*53*83*443,
# ... 1699*1733*1741*1811*1867*1907*1913.
# After each step the gcd of the power-1 and n
# is calculated. If the power was not 1, then
# this gcd is returned. If the power was 1,
# i.e. more factor steps in, then the power
# calculation is repeated in smaller steps.
# If this separates the factors, then a factor
# is given back. Otherwise we try again.
#
>
```

```
> ifactor/pollp1=proc(n,seed)
> local d,i,k,primes,w;
> options `Copyright 1993 by Waterloo Maple Software`;
>     primes := [512,2701400625,15369250897,22019225847811,
3312226271377,
>         573380756513,9895915431937,9901032144937,
26758334120513,
>         221345652736259,351896335286371,602532741650737,
705905813463569,
>         1146860257543171,1197923438167619,4101182822350853,
18093632169489029,
>         1578734133234919 9,30541698536834113,32272865510432777,
>         90193223385948289,94859279632319593,
220957856942948429,
>         217426779959284993,353818522181190721,
419399945462884489,
>         577244189067313457,576566448802761223,
635010439316635813,
>         1163208606573930629,1274527543260713153,
964201483442864677,
>         1189549054560841981,1730661029236703533,
2324525996860803761,
>         1749379074918976547,2379318098186474761,
3906310006235021863,
>         4400036932968546433,3486357872137291841,
4051550852530311307,
>         7062924201913552913,9946204387896728657,
9800419932800824021,
>         14016416453138321531,18451547056634890273,
20770839191296803529,
>         23262237873880485889,28282922465278428769,
23779420029816701443,
>         29006427360004220003,6322941213293968642 9249];
>     seed;
>     for i to nops(primes) do
>         modp(power(%,primes[i]),n);
>         if igcd(%-1,n) <> 1 then
>             if % <> 1 then RETURN(igcd(%-1,n)) fi;
>             w := %%;
>             d := numtheory[factorset](primes[i]);
>             for k in d do
>                 w := modp(power(w,k),n);
>                 if w = 1 then
>                     if 1 < i then RETURN(procname(n,modp(power
(seed,k),n)))
>                     fi
>                 elif igcd(w-1,n) <> 1 then RETURN(igcd(w-1,n))
>                 fi
```

```
>             od;
>             RETURN(_tryagain)
>         fi
>     od;
>     FAIL
> end;
```

$$\frac{ifactor}{pollp1} = \mathbf{proc}(n, seed) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This procedure is a conterpart of the
# procedure above. Using the huge integers
# _prpr4000, _prpr10000, ... , _prpr94000
# which are products of primes from 4000 to 10000,
# from 10000 to 16000, etc. for which p-1
# is not smooth, with the gcd method and using
# ifactor/wheelfract find these factors too.
#
>
> ifactor/pp100000=proc(n)
> local i;
> options `Copyright 1993 by Waterloo Maple Software`;
>     for i from 4000 by 6000 to 94000 do
>         igcd(_prpr.i,n);
>         if % <> 1 then
>             if % <> n then RETURN(%)
>             else RETURN(`ifactor/wheelfact`(n,i+1))
>             fi
>         fi
>     od;
>     FAIL
> end;
```

$$\frac{ifactor}{pp100000} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# The smallest constant used in ifactor/pp100000,
# as an example.
#
>
> _prpr4000:=
99650580605048944632656882899614563572409972083619836610534846707
34\
>
83901267943339014801270129828448962441064073276626649484616165544
```

```
72768889371966705406747239720527290803677864361380156879525642460
43526647401039910701757660785727431053299423;
```

$$_prpr4000 := 9965058060504894463265688289961456357240997208361983661053484670734839012679433390148012701298284489624410640732766266494846161655447276888937196670540674723972052729080367786436138015687952564246043526647401039910701757660785727431053299423$$

```
>
>
#
# In the `easy` case no more further effort done.
#
>
> ifactor/easy=proc(n)
> options `Copyright 1993 by Waterloo Maple Software`;
> _c.(length(n))
> end;
```

$$\frac{ifactor}{easy} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
#
# This is the default procedure for
# factorization of complicated cases.
# It use a variation of the Morrison--Brillhard
# algorithm with continued fractions (??).
#
>
> ifactor/morrbril=proc(n)
> local i,j,A0,A1,Q0,Q1,PQ,iPQ,iPQc,g,r0,r1,qn,p,X,primes;
> global _sq,_X2,_XX2,_Nprimes4,_Heap4,_Prpri;
> options `Copyright 1993 by Waterloo Maple Software`;
>     _sq := '_sq';
>     PQ := [];
>     iPQ := 1;
>     iPQc := 1;
>     userinfo(1,ifactor,`ifactor final stage:`,n,
>         lprint(`Using a variation of the Morrison-Brillhart
algorithm`));
>     g := isqrt(n);
>     if n < g^2 then g := g-1 fi;
>     X := isqrt(isqrt(isqrt(isqrt(10^(isqrt(iquo(599*length(n),
4))-11)))));
>     _X2 := min(X^2,20*X);
>     _XX2 := X*_X2;
```

```
>     userinfo(9,ifactor,'n' = n,'X' = X,'_X2' = _X2);
>     primes[1] := 2;
>     for _Nprimes4 while primes[_Nprimes4] < X do
>         primes[_Nprimes4];
>         do  nextprime(%); if modp(power(n,1/2*%-1/2),%) = 1
then break fi od;
>         primes[_Nprimes4+1] := %
>     od;
>     p := 1;
>     while p <= _Nprimes4 do  p := 2*p od;
>     p := p-1;
>     for i from _Nprimes4-1 by -1 to 0 do
>         1;
>         j := 2*i+1;
>         while j < _Nprimes4 do  %%*_Heap4[j]; j := 2*j od;
>         if p < j then primes[j-p] else primes[j-p+_Nprimes4]
fi;
>         _Heap4[i] := %%%*%
>     od;
>     _Prpri := 1440*_Heap4[0];
>     A0 := 0;
>     A1 := 1;
>     Q1 := n;
>     Q0 := 1;
>     r1 := g;
>     do
>         qn := iquo(2*g-r1,Q0,'r0');
>         A0 := modp(qn*A1+A0,n);
>         Q1 := Q1+qn*(r0-r1);
>         if [qn,A0,r1] = PQ then RETURN(readlib
(`ifactor/pollard`)(n))
>         elif iPQc = iPQ then iPQ := 2*iPQ; PQ := [qn,A0,r1];
iPQc := 1
>         else iPQc := iPQc+1
>         fi;
>         analyze_resid(A0,-Q1,n);
>         if % <> FAIL then RETURN(%) fi;
>         qn := iquo(2*g-r0,Q1,'r1');
>         A1 := modp(qn*A0+A1,n);
>       Q0 := Q0+qn*(r1-r0);
>         analyze_resid(A1,Q0,n);
>         if % <> FAIL then RETURN(%) fi
>     od
> end;
```

$$\frac{ifactor}{morrbril} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
```

```
#
# This subroutine works for the Morrison--Brillhard
# algorithm.
#
>
> analyze_resid=proc(a,a2,n)
> local lrgpr,resid,base,g;
> global _sq;
> options `Copyright 1993 by Waterloo Maple Software`;
>     abs(a2);
>     igcd(%,_Prpri);
>     while % <> 1 do
>         iquo(%%,%); if _XX2 < % then RETURN(FAIL) fi; igcd(%,
%%)
>     od;
>     if _X2 < %% then RETURN(FAIL) fi;
>     lrgpr := %%;
>     base := a;
>     resid := a2;
>     do
>         igcd(resid,_Heap4[0]);
>         g := igcd(iquo(resid,%),%);
>         while g <> 1 do
>             resid := iquo(resid,g^2);
>             base := modp(base/g,n);
>             igcd(g,resid);
>             g := igcd(%,iquo(resid,%))
>         od;
>         if resid = 1 then
>             if base <> 1 and base <> n-1 then RETURN(igcd(base
-1,n))
>             else userinfo(9,ifactor,`Bad luck (+-1)^2=1`);
RETURN(FAIL)
>             fi
>         fi;
>         if lrgpr = 1 then lrgpr := lrgst_factor(resid) fi;
>         userinfo(9,ifactor,`*** factorable residue ***`,base,
resid,lrgpr);
>         if assigned(_sq[lrgpr]) then
>             userinfo(9,ifactor,`----- collision at`,lrgpr);
>             resid := resid*_sq[lrgpr][2]/lrgpr^2;
>             base := modp(base*_sq[lrgpr][1]/lrgpr,n);
>             lrgpr := 1
>         else _sq[lrgpr] := [base,resid]; RETURN(FAIL)
>         fi
>     od
> end;
```

$$analyze\_resid = \mathbf{proc}(a, a2, n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This is D. Shanks' undocumented square-free factorization.
#
>
> ifactor/squfof=proc(a)
>    local l,p,q,r,s,w;
>    options `Copyright 1993 by Waterloo Maple Software`;
>        p := isqrt(a);
>        if a < p^2 then p := p-1 fi;
>        q := a-p^2;
>        s := p;
>        if q = 0 then RETURN(p) elif q = 1 then RETURN(squfof
(2*a)) fi;
>        l := 2*isqrt(2*s);
>        do
>            if q <= l then w[q/igcd(q,2)] := 1 fi;
>            p := s-modp(s+p,q);
>            q := (a-p^2)/q;
>            r := isqrt(q);
>            if q = r^2 and w[r] <> 1 then break fi;
>            if q <= l then w[q/igcd(q,2)] := 1 fi;
>            p := s-modp(s+p,q);
>            q := (a-p^2)/q
>        od;
>        p := p+r*iquo(s-p,r);
>        q := (a-p^2)/r;
>        do
>            s-modp(s+%%,%); if % = %%% then RETURN(%%/igcd(%%,
2)) fi; (a-%^2)/%%
>        od
>    end;
```

$$\frac{ifactor}{squfof} = \mathbf{proc}(a) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This is J. M. Pollard's rho method.
#
>
> ifactor/pollard=proc(n,ex)
>     local EX;
>     options `Copyright 1993 by Waterloo Maple Software`;
>         if nargs = 1 then EX := 2
>         elif nargs <> 2 or not type(ex,integer) or ex < 2 then
>             ERROR(`invalid arguments`)
>         else EX := ex
```

```
>         fi;
>         2;
>         modp(power(%,EX)+1,n);
>         do
>             modp(power(%%,EX)+1,n);
>             modp(power(power(%%,EX)+1,EX)+1,n);
>             if 1 < igcd(%%-%,n) then
>                 igcd(%%-%,n); if % = n then RETURN(FAIL) else
RETURN(%) fi
>             fi
>         od;
>         FAIL
>     end;
```

$$\frac{ifactor}{pollard} = \mathbf{proc}(n, ex) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This is Lenstra's elliptic curve method.
#
>
> ifactor/lenstra=proc(n)
>     local i,s,prime,f,A,X,Z,rgen,sp,a,r,curves,B1,kg,kgg;
>     options `Copyright 1993 by Waterloo Maple Software`;
>         if nargs < 1 or 3 < nargs then ERROR(`wrong number of
arguments.`) fi;
>         if nargs < 3 then B1 := 1000000 else B1 := args[3] fi;
>         if nargs < 2 then curves := 30 else curves := args[2]
fi;
>         if modp(n,2) = 0 then RETURN(2) fi;
>         if modp(n,3) = 0 then RETURN(3) fi;
>         s := evalf(1/2*sqrt(5)-1/2,30);
>         A := array(1 .. curves);
>         X := array(1 .. curves);
>         Z := array(1 .. curves,[1 $ curves]);
>         rgen := rand(1 .. n-1);
>         for i to curves do
>             a := 0;
>             while modp(a*(a^2-1)*(9*a^2-1),n) = 0 do
>                 r := rgen();
>                 kg := r^2+6;
>                 kgg := igcd(kg,n);
>                 if kgg <> 1 then RETURN(kgg) fi;
>                 a := modp(6*r/kg,n)
>             od;
>             A[i] := modp(1/16*(-3*a^4-6*a^2+1)/a^3+1/2,n);
>             X[i] := modp(3/4*a,n)
>         od;
```

```
>         prime := 2;
>         while prime <= B1 do
>             sp := round(s*prime);
>             `ifactor/lenstra/mulpp`(1,n,A,sp,prime,B1,X,Z);
>             f := Z[1];
>             for i from 2 to curves do
>                 `ifactor/lenstra/mulpp`(i,n,A,sp,prime,B1,X,Z)
;
>                 f := modp(f*Z[i],n)
>             od;
>             f := igcd(f,n);
>             if f <> 1 then RETURN(f) fi;
>             prime := nextprime(prime)
>         od;
>         FAIL
>     end;
```

$$\frac{ifactor}{lenstra} = \mathbf{proc}(n) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This subroutine the multiplications on a given
# elliptic curve.
#
>
> ifactor/lenstra/mulpp=proc(i,n,A,mm,nn,B1,X,Z)
> local pow,ax,az;
> options `Copyright 1993 by Waterloo Maple Software`;
>     ax := X[i];
>     az := Z[i];
>     pow := nn;
>     while pow <= B1 do
>         `ifactor/lenstra/ellmul`(n,A[i],mm,nn,ax,az,'ax','az')
; pow := pow*nn
>     od;
>     X[i] := ax;
>     Z[i] := az
> end;
```

$$\frac{ifactor}{lenstra\ mulpp} = \mathbf{proc}(i, n, A, mm, nn, B1, X, Z) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This procedure do some multiplications
# on an elliptic curve.
#
>
```

```
> ifactor/lenstra/ellmul=proc(n,A,mm,nn,px,pz,aax,aaz)
> local ax,az,bx,bz,cx,cz,tmpx,tmpz,d,e,t1,t2;
> options `Copyright 1993 by Waterloo Maple Software`;
>     cx := px;
>     cz := pz;
>     e := mm;
>     d := nn-mm;
>     if e < d then
>         `ifactor/lenstra/elldoub`(n,A,px,pz,'bx','bz');
>         ax := px;
>         az := pz;
>         d := d-e
>     else
>         `ifactor/lenstra/elldoub`(n,A,px,pz,'ax','az');
>         bx := px;
>         bz := pz;
>         e := e-d
>     fi;
>     while e <> 0 do
>         if e < d then
>             tmpx := bx;
>             tmpz := bz;
>             t1 := modp((ax-az)*(bx+bz),n);
>             t2 := modp((ax+az)*(bx-bz),n);
>             bx := modp(cz*modp((t1+t2)^2,n),n);
>             bz := modp(cx*modp((t1-t2)^2,n),n);
>             d := d-e
>         else
>             tmpx := ax;
>             tmpz := az;
>             t1 := modp((ax-az)*(bx+bz),n);
>             t2 := modp((ax+az)*(bx-bz),n);
>             ax := modp(cz*modp((t1+t2)^2,n),n);
>             az := modp(cx*modp((t1-t2)^2,n),n);
>             e := e-d
>         fi;
>         cx := tmpx;
>         cz := tmpz
>     od;
>     aax := ax;
>     aaz := az
> end;
```

$$\frac{ifactor}{lenstra\,ellmul} = \mathbf{proc}(n, A, mm, nn, px, pz, aax, aaz) \ldots \mathbf{end\ proc}$$

```
>
>
#
# This procedure do a doubling on the elliptic curve.
```

```
#
>
> ifactor/lenstra/elldoub=proc(n,A,ax,az,cx,cz)
> local t1,t2;
> options `Copyright 1993 by Waterloo Maple Software`;
>     t1 := modp((ax+az)^2,n);
>     t2 := modp((ax-az)^2,n);
>     cx := modp(t1*t2,n);
>     cz := modp((t1-t2)*modp(A*(t1-t2)+t2,n),n)
> end;
```

$$\frac{ifactor}{lenstra\ elldoub} = \mathbf{proc}(n, A, ax, az, cx, cz) \ldots \mathbf{end\ proc}$$

```
>
>
>
```

▶ 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;
```

$preparefermatsieve := \mathbf{proc}(n, S, m)$ (2.6.1)

$\mathbf{local}\ i, j, k, x2, x2n;$

$x2 := table;$

```
    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
        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])
```

(2.6.2)

```
        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.

```
> interface(echo=3);
```

$$1 \tag{2.7.1}$$

```
> #
  # This procedure do factorization with addition and
  # substraction. Returns with the largest factor of
  # odd number n less then or equal to sqrt(n).
  #

  addsubfactor:=proc(n) local xp,yp,r;
```

```
   if type(n,even) then error "argument must be odd" fi;
   isqrt(n); xp:=2*%+1; r:=%%^2-n; yp:=1;
   while true do
     while r>0 do r:=r-yp; yp:=yp+2 od;
     if r=0 then return (xp-yp)/2 fi;
     r:=r+xp; xp:=xp+2;
   od; end;
```

$$addsubfactor := \mathbf{proc}(n) \tag{2.7.2}$$

local xp, yp, r;

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

error "argument must be odd"

end if;

$isqrt(n)$;

$xp := 2 * \% + 1$;

$r := \%\%\hat{}2 - n$;

$yp := 1$;

do

while $0 < r$ **do**

$r := r - yp$;

$yp := yp + 2$

end do;

if $r = 0$ **then**

return $1 / 2 * xp - 1 / 2 * yp$

end if;

$r := r + xp$;

$xp := xp + 2$

end do

end proc

```
> debug(addsubfactor); addsubfactor(111);
```

$addsubfactor$

```
{--> enter addsubfactor, args = 111
```

11

$xp := 23$

$r := 10$

$yp := 1$

$r := 9$

$yp := 3$

$r := 6$

$yp := 5$

$r := 1$

$yp := 7$

$r := -6$

$yp := 9$

$r := 17$

$xp := 25$

$r := 8$

$yp := 11$

$r := -3$

$yp := 13$

$r := 22$

$xp := 27$

$r := 9$

$yp := 15$

$r := -6$

$yp := 17$

$r := 21$

$xp := 29$

$r := 4$

$yp := 19$

$r := -15$

$yp := 21$

$r := 14$

$xp := 31$

$r := -7$

$yp := 23$

$r := 24$

$xp := 33$

$$r := 1$$

$$yp := 25$$

$$r := -24$$

$$yp := 27$$

$$r := 9$$

$$xp := 35$$

$$r := -18$$

$$yp := 29$$

$$r := 17$$

$$xp := 37$$

$$r := -12$$

$$yp := 31$$

$$r := 25$$

$$xp := 39$$

$$r := -6$$

$$yp := 33$$

$$r := 33$$

$$xp := 41$$

$$r := 0$$

$$yp := 35$$

```
<-- exit addsubfactor (now at top level) = 3}
```

$$3 \tag{2.7.3}$$

```
> undebug(addsubfactor); addsubfactor(6463); addsubfactor
  (999863*999883);
```

$$23$$

$$999863 \tag{2.7.4}$$

▶ 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^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);
```

$$999863 \tag{2.9.2}$$

▶ 2.10. Feladat.

▶ 2.11. Fermat tétele.

2.12. Euler tétele.

2.13. Kínai maradéktétel.

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

$$23 \tag{2.13.1}$$

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;
```

$$\begin{aligned}
&\mathit{left2right} := \mathbf{proc}(a, e::\mathit{posint}, \mathit{mult}::\mathit{procedure}) \\
&\quad \mathbf{local}\ b, x, n; \\
&\quad b := \mathit{convert}(e, \mathit{base}, 2); \\
&\quad x := a; \\
&\quad \mathbf{for}\ n\ \mathbf{from}\ \mathit{nops}(b) - 1\ \mathbf{by} - 1\ \mathbf{to}\ 1\ \mathbf{do} \\
&\qquad x := \mathit{mult}(x, x); \\
&\qquad \mathbf{if}\ 0 < b[n]\ \mathbf{then} \\
&\qquad\quad x := \mathit{mult}(x, a) \\
&\qquad \mathbf{end\ if} \\
&\quad \mathbf{end\ do}; \\
&\quad x \\
&\mathbf{end\ proc}
\end{aligned} \tag{2.15.1}$$

```
> 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}
```

$$2048 \tag{2.15.2}$$

```
> #
  # 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;
```

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

$$\mathbf{local}\ i, j, k, P, x, b, aa, n;$$

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

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

$$x := a;$$

$$P := [a];$$

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

```
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}
```

$$2048 \tag{2.15.4}$$

```
> 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 $2\text{^}e < B$ **do**

$e := e + 1$

end do;

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

if $1 < igcd(x - 1, n)$ **or** $P = 2$ **then**

return x

end if;

while $3 * B < 3\text{^}e$ **and** $1 < e$ **do**

$e := e - 1$

end do;

$x := mod(x\,\&\text{^}\,(3\text{^}e), n)$;

```
    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);
```

$$23324 \tag{2.17.2}$$

```
> igcd(%-1,25852);
```

$$281 \tag{2.17.3}$$

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

$$16252910338466315 \tag{2.17.4}$$

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

$$999917 \tag{2.17.5}$$

▶ 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;
```

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

local x, i, j, E, aa, p, pp, d;

$E := \mathit{Array}(1..N)$;

$aa := \mathit{mod}(a * a, n)$;

$E[1] := aa$;

for j **from** 2 **to** N **do**

$E[j] := E[j-1] * aa$

end do;

$p := \mathit{ithprime}(m)$;

$x := \mathit{mod}(a \,\&\hat{}\, p, n)$;

for i **from** $m+1$ **to** M **while** $\mathit{gcd}(x-1, n) = 1$ **do**

$pp := \mathit{nextprime}(p)$;

$d := pp - p$;

$p := pp$;

if $d <= 2 * N$ **then**

$x := \mathit{mod}(x * E[1/2 * d], n)$

else

$x := \mathit{mod}(x * a \,\&\hat{}\, d, n)$

end if

end do;

x

end proc

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

$$14664579960852753237179827 \quad (2.19.2)$$

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

$$1 \quad (2.19.3)$$

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

$$391453341919440359125466 6 \tag{2.19.4}$$

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

$$23528569104401 \tag{2.19.5}$$

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

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

▶ 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

▶ 15. Számtest szita

▶ 16. Vegyes problémák