Számítógépes számelmélet

Járai Antal

Ezek a programok csak szemléltetésre szolgálnak

1. A prímek eloszlása, szitálás

2. Egyszerű faktorizálási módszerek

3. Egyszerű prímtesztelési módszerek

4. Lucas-sorozatok

5. Alkalmazások

6. Számok és polinomok

7. Gyors Fourier-transzformáció

```
> restart;
```

7.1. Polinomszorzás gyors Fourier-transzformációval.

7.2. Gyors Fourier-transzformáció (FFT).

```
> #
  # This procedure do the bit reversation of the
  # number x which is k bit long.
  #

  reverse:=proc(x,k) local xx,i,y;
    xx:=x; y:=0;
    for i to k do
      if type(xx,odd) then
        y:=2*y+1; xx:=(xx-1)/2;
      else
        y:=2*y; xx:=xx/2;
      fi;
    od; y; end;
```

(7.2.1)

$reverse := \mathbf{proc}(x, k)$ (7.2.1)

```
    local xx, i, y;
    xx := x;
    y := 0;
    for i to k do
        if type(xx, odd) then
            y := 2*y + 1;
            xx := 1/2*xx - 1/2
        else
            y := 2*y;
            xx := 1/2*xx
        end if
    end do;
    y
end proc
```

```
> #
  # This procedure do the complex butterfly operation
  # between the two terms A[i] and A[j] of the
  # table A. The multiplier is w.
  #

  cbutterfly:=proc(A,i,j,w) local X,Y;
    X:=A[i]; Y:=A[j]*w; A[i]:=X+Y; A[j]:=X-Y;
  end;
```

$cbutterfly := \mathbf{proc}(A, i, j, w)$ (7.2.2)

```
    local X, Y;
    X := A[i];
    Y := A[j]*w;
    A[i] := X + Y;
    A[j] := X - Y
end proc
```

```
> #
  # This is a complex FFT procedure.
  # It use the butterfly procedure to operate on the vector A.
  # The number of rounds is k in the FFT.
  # T is a table of the powers of primitive root of unity.
  #
```

```
cfft:=proc(A,T,k) local l,s,w,t;
for l from 0 to k-1 do
  for s from 0 to 2^l-1 do
    w:=T[s];
    for t from 0 to 2^(k-l-1)-1 do
      cbutterfly(A,2^(k-l)*s+t,2^(k-l)*s+t+2^(k-l-1),w);
    od;
  od;
od; end;
```

$cfft := \mathbf{proc}(A, T, k)$ (7.2.3)

```
local l, s, w, t;
for l from 0 to k − 1 do
  for s from 0 to 2^l − 1 do
    w := T[s];
    for t from 0 to 2^(k − l − 1) − 1 do
      cbutterfly(A, 2^(k − l) * s + t, 2^(k − l) * s + t + 2^(k − l −
        1),
      w)
    end do
  end do
end do
end proc
```

```
> #
  # The pre procedure do the preparation for the
  # table T[0..2^k-1)] of powers of the primitive
  # root of unity.
  #

  pre:=proc(T,k) local i,xx;
  Digits:=2*Digits;
  for i from 0 to 2^k-1 do
    T[i]:=evalf(exp(-2*Pi*I*reverse(i,k)/2^(k+1)));
  od;
  Digits:=Digits/2;
  end;
```

$pre := \mathbf{proc}(T, k)$ (7.2.4)

```
local i, xx;
Digits := 2 * Digits;
for i from 0 to 2^k − 1 do
```

$T[i] := evalf(\exp(-2 * I * \pi * reverse(i, k) / 2^{\wedge}(k+1)))$

end do;

$Digits := 1/2 * Digits$

end proc

```
> k:=5;
  A:=array(0..2^k-1); for i from 0 to 2^k-1 do A[i]:=0 od:
  T:=array(0..2^(k-1)-1); pre(T,k-1):
```

$$k := 5$$

$$A := array(0..31, [\,])$$

$$T := array(0..15, [\,]) \qquad (7.2.5)$$

```
> A[1]:=1+I; cfft(A,T,5): print(A);
```

$$A_1 := 1 + I$$

$ARRAY([0..31], [0 = 1. + 1.\,I, 1 = -1. - 1.\,I, 2 = 1. - 1.\,I, 3 = -1.$ (7.2.6)

$+ 1.\,I, 4 = 1.414213562$

$+ 0.\,I, 5 = -1.414213562 - 0.\,I, 6 = 0. - 1.414213562\,I, 7 = 0.$

$+ 1.414213562\,I, 8 = 1.306562965$

$+ 0.5411961001\,I, 9 = -1.306562965 - 0.5411961001\,I, 10$

$= 0.5411961001 - 1.306562965\,I, 11 = -0.5411961001$

$+ 1.306562965\,I, 12 = 1.306562965 - 0.5411961001\,I, 13$

$= -1.306562965 + 0.5411961001\,I, 14$

$= -0.5411961001 - 1.306562965\,I, 15 = 0.5411961001$

$+ 1.306562965\,I, 16 = 1.175875602$

$+ 0.7856949584\,I, 17 = -1.175875602 - 0.7856949584\,I, 18$

$= 0.7856949584 - 1.175875602\,I, 19 = -0.7856949584$

$+ 1.175875602\,I, 20 = 1.387039845 - 0.2758993793\,I, 21$

$= -1.387039845 + 0.2758993793\,I, 22$

$= -0.2758993793 - 1.387039845\,I, 23 = 0.2758993793$

$+ 1.387039845\,I, 24 = 1.387039845$

$+ 0.2758993793\,I, 25 = -1.387039845 - 0.2758993793\,I, 26$

$= 0.2758993793 - 1.387039845\,I, 27 = -0.2758993793$

$+ 1.387039845\,I, 28 = 1.175875602 - 0.7856949584\,I, 29$

$= -1.175875602 + 0.7856949584\,I, 30$

$= -0.7856949584 - 1.175875602\,I, 31 = 0.7856949584$

+ 1.175875602 I])

```
> for i from 0 to 2^k-1 do
    j:=reverse(i,k);
    if i<j then x:=A[i]; A[i]:=A[j]; A[j]:=x; fi;
  od: print(A);
```

$ARRAY([0..31], [0 = 1. + 1. I, 1 = 1.175875602$ (7.2.7)

+ 0.7856949584 I, 2 = 1.306562965
+ 0.5411961001 I, 3 = 1.387039845
+ 0.2758993793 I, 4 = 1.414213562
+ 0. I, 5 = 1.387039845 − 0.2758993793 I, 6
= 1.306562965 − 0.5411961001 I, 7
= 1.175875602 − 0.7856949584 I, 8 = 1. − 1. I, 9
= 0.7856949584 − 1.175875602 I, 10
= 0.5411961001 − 1.306562965 I, 11
= 0.2758993793 − 1.387039845 I, 12 = 0. − 1.414213562 I, 13
= −0.2758993793 − 1.387039845 I, 14
= −0.5411961001 − 1.306562965 I, 15
= −0.7856949584 − 1.175875602 I, 16 = −1. − 1. I, 17
= −1.175875602 − 0.7856949584 I, 18
= −1.306562965 − 0.5411961001 I, 19
= −1.387039845 − 0.2758993793 I, 20 = −1.414213562 − 0. I, 21
= −1.387039845 + 0.2758993793 I, 22 = −1.306562965
+ 0.5411961001 I, 23 = −1.175875602 + 0.7856949584 I, 24 = −1.
+ 1. I, 25 = −0.7856949584 + 1.175875602 I, 26 = −0.5411961001
+ 1.306562965 I, 27 = −0.2758993793 + 1.387039845 I, 28 = 0.
+ 1.414213562 I, 29 = 0.2758993793
+ 1.387039845 I, 30 = 0.5411961001
+ 1.306562965 I, 31 = 0.7856949584 + 1.175875602 I])

```
> cfft(A,T,5): print(A);
```

ARRAY([0..31], [0 = 0. + 0. I, 1 = 0. + 0. I, 2 = 0. + 0. I, 3 = 0. + 0. I, 4 = 0. (7.2.8)

+ 0. I, 5 = 0. + 0. I, 6 = 0. + 0. I, 7 = 0. + 0. I, 8 = 0. + 0. I, 9 = 0.
+ 0. I, 10 = 0. + 0. I, 11 = 0. + 0. I, 12 = 0. + 0. I, 13 = 0. + 0. I, 14 = 0.
+ 0. I, 15 = 0. + 0. I, 16 = 0. + 0. I, 17 = 0. + 0. I, 18 = 0. + 0. I, 19 = 0.

$+ 0. I, 20 = 0. + 0. I, 21 = 0. + 0. I, 22 = 0. + 0. I, 23 = 0.$

$+ 0. I, 24 = 9.760966906\ 10^{-10}$

$+ 2.790051372\ 10^{-9}\ I, 25$

$= -8.238557556\ 10^{-10} - 1.555418236\ 10^{-9}\ I, 26 = 2.525224419\ 10^{-9}$

$+ 2.834458414\ 10^{-9}\ I, 27 = 1.322534645\ 10^{-9} - 6.9091550\ 10^{-11}\ I, 28$

$= -3.8429439\ 10^{-11} - 1.609819356\ 10^{-9}\ I, 29$

$= -3.961570561\ 10^{-9} - 2.390180644\ 10^{-9}\ I, 30 = 1.\ 10^{-8}$

$+ 1.\ 10^{-8}\ I, 31 = 31.99999999 + 31.99999999\ I\,]\,)$

```
> for i from 0 to 2^k-1 do
    j:=reverse(i,k);
    if i<j then x:=A[i]; A[i]:=A[j]; A[j]:=x; fi;
  od: print(A);
```

$ARRAY\big([0..31], [0 = 0. + 0. I, 1 = 0. + 0. I, 2 = 0.$ (7.2.9)

$+ 0. I, 3 = 9.760966906\ 10^{-10} + 2.790051372\ 10^{-9}\ I, 4 = 0. + 0. I, 5 = 0.$

$+ 0. I, 6 = 0. + 0. I, 7 = -3.8429439\ 10^{-11} - 1.609819356\ 10^{-9}\ I, 8 = 0.$

$+ 0. I, 9 = 0. + 0. I, 10 = 0. + 0. I, 11 = 2.525224419\ 10^{-9}$

$+ 2.834458414\ 10^{-9}\ I, 12 = 0. + 0. I, 13 = 0. + 0. I, 14 = 0.$

$+ 0. I, 15 = 1.\ 10^{-8} + 1.\ 10^{-8}\ I, 16 = 0. + 0. I, 17 = 0. + 0. I, 18 = 0.$

$+ 0. I, 19 = -8.238557556\ 10^{-10} - 1.555418236\ 10^{-9}\ I, 20 = 0.$

$+ 0. I, 21 = 0. + 0. I, 22 = 0.$

$+ 0. I, 23 = -3.961570561\ 10^{-9} - 2.390180644\ 10^{-9}\ I, 24 = 0.$

$+ 0. I, 25 = 0. + 0. I, 26 = 0.$

$+ 0. I, 27 = 1.322534645\ 10^{-9} - 6.9091550\ 10^{-11}\ I, 28 = 0. + 0. I, 29 = 0.$

$+ 0. I, 30 = 0. + 0. I, 31 = 31.99999999 + 31.99999999\ I\,]\big)$

7.3. Inverz FFT.

```
> #
  # This procedure do the inverse complex butterfly operation
  # between the two terms A[i] and A[j] of the
  # table A. The power of primitive unity is w.
  #

  icbutterfly:=proc(A,i,j,w) local X,Y,e;
    X:=A[i]+A[j]; Y:=A[i]-A[j]; A[i]:=X; A[j]:=Y/w;
```

```
end;
```

icbutterfly := **proc**(*A*, *i*, *j*, *w*) (7.3.1)

```
    local X, Y, e;
    X := A[i] + A[j];
    Y := A[i] - A[j];
    A[i] := X;
    A[j] := Y / w
end proc
```

```
> #
  # This procedure is a complex IFFT procedure.
  # It use the ibutterfly procedure to operate on the vector A.
  # The number of round in the FFT is k and T is a table of the
  # powers of primitive root of unity.
  #

  icfft:=proc(A,T,k) local l,s,w,t;
  for l from k-1 to 0 by -1 do
    for s from 0 to 2^l-1 do
      w:=T[s];
      for t from 0 to 2^(k-l-1)-1 do
        icbutterfly(A,2^(k-l)*s+t,2^(k-l)*s+t+2^(k-l-1),w);
      od;
    od;
  od; end;
```

icfft := **proc**(*A*, *T*, *k*) (7.3.2)

```
    local l, s, w, t;
    for l from k - 1 by -1 to 0 do
        for s from 0 to 2^l - 1 do
            w := T[s];
            for t from 0 to 2^(k - l - 1) - 1 do
                icbutterfly(A, 2^(k - l)*s + t, 2^(k - l)*s + t
                + 2^(k - l - 1), w)
            end do
        end do
    end do
end proc
```

```
> for i from 0 to 2^k-1 do A[i]:=0 od:
  A[1]:=1+I; cfft(A,T,5): print(A);
```

$$A_1 := 1 + I$$

$ARRAY([0..31], [0 = 1. + 1. I, 1 = -1. - 1. I, 2 = 1. - 1. I, 3 = -1. + 1. I, 4 = 1.414213562 + 0. I, 5 = -1.414213562 - 0. I, 6 = 0. - 1.414213562 I, 7 = 0. + 1.414213562 I, 8 = 1.306562965 + 0.5411961001 I, 9 = -1.306562965 - 0.5411961001 I, 10 = 0.5411961001 - 1.306562965 I, 11 = -0.5411961001 + 1.306562965 I, 12 = 1.306562965 - 0.5411961001 I, 13 = -1.306562965 + 0.5411961001 I, 14 = -0.5411961001 - 1.306562965 I, 15 = 0.5411961001 + 1.306562965 I, 16 = 1.175875602 + 0.7856949584 I, 17 = -1.175875602 - 0.7856949584 I, 18 = 0.7856949584 - 1.175875602 I, 19 = -0.7856949584 + 1.175875602 I, 20 = 1.387039845 - 0.2758993793 I, 21 = -1.387039845 + 0.2758993793 I, 22 = -0.2758993793 - 1.387039845 I, 23 = 0.2758993793 + 1.387039845 I, 24 = 1.387039845 + 0.2758993793 I, 25 = -1.387039845 - 0.2758993793 I, 26 = 0.2758993793 - 1.387039845 I, 27 = -0.2758993793 + 1.387039845 I, 28 = 1.175875602 - 0.7856949584 I, 29 = -1.175875602 + 0.7856949584 I, 30 = -0.7856949584 - 1.175875602 I, 31 = 0.7856949584 + 1.175875602 I])$ (7.3.3)

```
> icfft(A,T,k): print(A);
```

$ARRAY([0..31], [0 = 0. + 0. I, 1 = 32.00000000 + 32.00000000 I, 2 = 0. + 0. I, 3 = 0. + 0. I, 4 = 0. + 0. I, 5 = -8.28427124\ 10^{-10} - 8.28427124\ 10^{-10} I, 6 = 0. + 0. I, 7 = 0. + 0. I, 8 = 0. + 0. I, 9 = -4.000000000\ 10^{-9} - 4.000000000\ 10^{-9} I, 10 = 0. + 0. I, 11 = 0. + 0. I, 12 = 0. + 0. I, 13 = 2.\ 10^{-9} + 2.\ 10^{-9} I, 14 = 0. + 0. I, 15 = 0. + 0. I, 16 = 0. + 0. I, 17 = 0. + 0. I, 18 = 0. + 0. I, 19 = 0. + 0. I, 20 = 0. + 0. I, 21 = 4.828427124\ 10^{-9}$ (7.3.4)

$+ 4.828427124\ 10^{-9}\ I,\ 22 = 0. + 0.\ I,\ 23 = 0. + 0.\ I,\ 24 = 0. + 0.\ I,\ 25 = 0.$
$+ 0.\ I,\ 26 = 0. + 0.\ I,\ 27 = 0. + 0.\ I,\ 28 = 0. + 0.\ I,\ 29 = 2.\ 10^{-9}$
$+ 2.\ 10^{-9}\ I,\ 30 = 0. + 0.\ I,\ 31 = 0. + 0.\ I])$

7.4. Szorzás komplex FFT-vel.

```
> #
  # The cdigmul procedure do the digit-by-digit
  # multiplication of the two numbers after the
  # cfft's. The result will be in the first table.
  #

  cdigmul:=proc(T,S,k) local i;
  for i from 0 to 2^k-1 do T[i]:=T[i]*S[i]; od;
  end;
```

cdigmul := **proc**(*T*, *S*, *k*) (7.4.1)

local *i*;

for *i* **from** 0 **to** 2^*k* − 1 **do**

T[*i*] := *T*[*i*] * *S*[*i*]

end do

end proc

```
> A:=array(0..2^k-1); for i from 0 to 2^k -1 do A[i]:=0 od:
  B:=array(0..2^k-1); for i from 0 to 2^k -1 do B[i]:=0 od:
```

A := *array*(0..31, [ ])

B := *array*(0..31, [ ]) (7.4.2)

```
> A[0]:=1: A[1]:=1: A[2]:=1: print(A);
```

ARRAY([0..31], [0 = 1, 1 = 1, 2 = 1, 3 = 0, 4 = 0, 5 = 0, 6 = 0, 7 = 0, 8 = 0, 9 = 0, 10 = 0, 11 = 0, 12 = 0, 13 = 0, 14 = 0, 15 = 0, 16 = 0, 17 = 0, 18 = 0, 19 = 0, 20 = 0, 21 = 0, 22 = 0, 23 = 0, 24 = 0, 25 = 0, 26 = 0, 27 = 0, 28 = 0, 29 = 0, 30 = 0, 31 = 0]) (7.4.3)

```
> #
  # This is the polynom multiplication, do multiplication or
  squaring.
  # A and B are the two polynomials, the FFT and IFFT use k
  rounds.
  #

  polmulcfft:=proc(A,B,k) global T;
  if A=B then
```

```
    cfft(A,T,k);
    cdigmul(A,A,k);
    icfft(A,T,k);
  else
    cfft(A,T,k);
    cfft(B,T,k);
    cdigmul(A,B,k);
    icfft(A,T,k);
  fi; end;
```

$polmulcfft := \mathbf{proc}(A, B, k)$ (7.4.4)

global T;

if $A = B$ **then**

$cfft(A, T, k)$;

$cdigmul(A, A, k)$;

$icfft(A, T, k)$

else

$cfft(A, T, k)$;

$cfft(B, T, k)$;

$cdigmul(A, B, k)$;

$icfft(A, T, k)$

end if

end proc

```
> polmulcfft(A,A,k): print(map(x->x/2^k,A));
```

$ARRAY([0..31], [0 = 0.9999999994 + 0. I, 1 = 1.999999999$ (7.4.5)

$+ 0. I, 2 = 2.999999999 + 0. I, 3 = 2.000000000 + 0. I, 4 = 1.000000000$

$+ 0. I, 5 = 8.080582619\ 10^{-11} + 0. I, 6 = -1.061335052\ 10^{-10}$

$+ 0. I, 7 = 2.251727930\ 10^{-10} + 0. I, 8 = 0.$

$+ 0. I, 9 = -1.875000000\ 10^{-10} + 0. I, 10 = -1.419231619\ 10^{-10}$

$+ 0. I, 11 = -6.243093422\ 10^{-10} + 0. I, 12 = -2.758883476\ 10^{-10}$

$+ 0. I, 13 = 1.325825215\ 10^{-10} + 0. I, 14 = 6.893616238\ 10^{-10}$

$+ 0. I, 15 = 5.787261838\ 10^{-10} + 0. I, 16 = 6.250000000\ 10^{-10}$

$+ 0. I, 17 = 6.250000000\ 10^{-10} + 0. I, 18 = 6.250000000\ 10^{-10}$

$+ 0. I, 19 = 3.125000000\ 10^{-10} + 0. I, 20 = 6.250000000\ 10^{-10}$

$+ 0. I, 21 = 1.691941738\ 10^{-10} + 0. I, 22 = 1.945218528\ 10^{-10}$

$+0.\,I,\ 23 = 7.660390225\ 10^{-11} + 0.\,I,\ 24 = 0.$

$+0.\,I,\ 25 = 1.875000000\ 10^{-10} + 0.\,I,\ 26 = 1.419231619\ 10^{-10}$

$+0.\,I,\ 27 = -6.906577500\ 10^{-13} + 0.\,I,\ 28 = -9.911165238\ 10^{-11}$

$+0.\,I,\ 29 = -1.325825215\ 10^{-10}$

$+0.\,I,\ 30 = -7.777499712\ 10^{-10} - 0.\,I,\ 31 = -6.305028788\ 10^{-10} - 0.\,I])$

```
> #
  # The cfftpre procedure do the preparation for the
  # table for cfft, where x is the number, A is the
  # table, m the modulus, and k is the number of
  # rounds for the cfft, hence the table is 2^k long.
  #

  cfftpre:=proc(x,A,m,k) local i,xx;
  xx:=x;
  for i from 0 to 2^k-1 do A[i]:=irem(xx,m,'xx'); od;
  end;
```

cfftpre := **proc**(*x*, *A*, *m*, *k*) (7.4.6)

```
    local i, xx;
    xx := x;
    for i from 0 to 2^k - 1 do
        A[i] := irem(xx, m, 'xx')
    end do
end proc
```

```
> #
  # The cnorm procedure do the normalization
  # after the icfft; A is the table, m the modulus,
  # and k is the number of rounds for the cfft,
  # hence the table is 2^k long. The fraction parts are
  # left in the table A.
  #

  cnorm:=proc(A,m,k) local i,x;
  x:=0;
  for i from 2^k-1 to 0 by -1 do
    A[i]:=A[i]/2^k;
    x:=m*x+round(A[i]);
    A[i]:=A[i]-round(A[i]);
  od; x; end;
```

cnorm := **proc**(*A*, *m*, *k*) (7.4.7)

```
    local i, x;
```

$x := 0;$

for i **from** $2^k - 1$ **by** -1 **to** 0 **do**

$A[i] := A[i] / 2^k;$

$x := m * x + \text{round}(A[i]);$

$A[i] := A[i] - \text{round}(A[i])$

end do;

x

end proc

```
> #
  # This is the main procedure, do the multiplication or the
  squaring.
  # a and b are the two numbers, m is the modulus for the
  preparation.
  # The complex FFT and IFFT use k rounds.
  #

  mulcfft:=proc(a,b,m,k) global A,B,T;
  if a=b then
    cfftpre(a,A,m,k);
    cfft(A,T,k);
    cdigmul(A,A,k);
    icfft(A,T,k);
    cnorm(A,m,k);
  else
    cfftpre(a,A,m,k);
    cfft(A,T,k);
    cfftpre(b,B,m,k);
    cfft(B,T,k);
    cdigmul(A,B,k);
    icfft(A,T,k);
    cnorm(A,m,k);
  fi; end;
```

mulcfft := **proc**(a, b, m, k) (7.4.8)

global A, B, T;

if $a = b$ **then**

cfftpre(a, A, m, k);

cfft(A, T, k);

cdigmul(A, A, k);

icfft(A, T, k);

cnorm(A, m, k)

```
    else
        cfftpre(a, A, m, k);
        cfft(A, T, k);
        cfftpre(b, B, m, k);
        cfft(B, T, k);
        cdigmul(A, B, k);
        icfft(A, T, k);
        cnorm(A, m, k)
    end if
end proc
> mulcfft(123456789,987654321,20,5);
```

$$121932631112635269 \quad (7.4.9)$$

```
> 123456789*987654321;
```

$$121932631112635269 \quad (7.4.10)$$

```
> print(A);
```

$$ARRAY([0..31], [0 = 6.2\,10^{-8} + 0.\,I, 1 = -1.\,10^{-7} + 0.\,I, 2 = 0. + 0.\,I, 3 = 0. + 0.\,I, 4 = 0. + 0.\,I, 5 = 0. + 0.\,I, 6 = 0. + 0.\,I, 7 = 0. + 0.\,I, 8 = 0. + 0.\,I, 9 = 0. + 0.\,I, 10 = 0. + 0.\,I, 11 = 1.\,10^{-7} + 0.\,I, 12 = 7.\,10^{-8} + 0.\,I, 13 = -1.830582619\,10^{-8} + 0.\,I, 14 = 0. + 0.\,I, 15 = 1.325825215\,10^{-7} + 0.\,I, 16 = -6.250000000\,10^{-8} + 0.\,I, 17 = 1.250000000\,10^{-7} + 0.\,I, 18 = -1.250000000\,10^{-7} + 0.\,I, 19 = 0. + 0.\,I, 20 = 0. + 0.\,I, 21 = 0. + 0.\,I, 22 = 0. + 0.\,I, 23 = 0. + 0.\,I, 24 = 0. + 0.\,I, 25 = -6.250000000\,10^{-8} + 0.\,I, 26 = 0. + 0.\,I, 27 = -6.250000000\,10^{-8} + 0.\,I, 28 = -6.875000000\,10^{-8} + 0.\,I, 29 = -1.066941738\,10^{-7} + 0.\,I, 30 = 0. - 0.\,I, 31 = -1.325825215\,10^{-7} - 0.\,I]) \quad (7.4.11)$$

7.5. Valós FFT.

```
> #
  # The CtoR procedure do the conversion from complex
  # representation to real representation.
  # The result will be in the same table.
  #
```

```
CtoR:=proc(A,T,k) local i,x,y;
x:=Re(A[0]); y:=Im(A[0]);
A[0]:=2*(x+y)+I*2*(x-y);
x:=Re(A[1]); y:=Im(A[1]);
A[1]:=2*x-I*2*y;
for i to k-1 do CtoRsteps(A,T,2^i); od;
end;
```

$CtoR := \mathbf{proc}(A, T, k)$ (7.5.1)

```
    local i, x, y;
    x := ℜ(A[0]);
    y := ℑ(A[0]);
    A[0] := 2*x + 2*y + 2*I*(x - y);
    x := ℜ(A[1]);
    y := ℑ(A[1]);
    A[1] := 2*x - 2*I*y;
    for i to k - 1 do
        CtoRsteps(A, T, 2^i)
    end do
end proc
```

```
> #
# The CtoRstep procedure do the conversion from complex
# representation to real representation for one pair
# ll, uu with weight w.
#

CtoRstep:=proc(ll,uu,w) local al,be,ga,de,xi,et,a,b,x,y,u,v;
xi:=Re(w); et:=Im(w);
al:=Re(ll); be:=Im(ll);
ga:=Re(uu); de:=Im(uu);
x:=al+ga; y:=be-de;
a:=al-ga; b:=be+de;
u:=et*b-xi*a; v:=xi*b+et*a;
[x+v+I*(y+u),x-v+I*(u-y)]
end;
```

$CtoRstep := \mathbf{proc}(ll, uu, w)$ (7.5.2)

```
    local al, be, ga, de, ξ, et, a, b, x, y, u, v;
    ξ := ℜ(w);
    et := ℑ(w);
    al := ℜ(ll);
```

$be := \Im(ll);$

$ga := \Re(uu);$

$de := \Im(uu);$

$x := al + ga;$

$y := be - de;$

$a := al - ga;$

$b := be + de;$

$u := et * b - \xi * a;$

$v := \xi * b + et * a;$

$[x + v + I * (y + u), x - v + I * (u - y)]$

end proc

```
> #
  # The CtoRsteps procedure do the conversion from complex
  # representation to real representation for one series.
  # The index i is the lower index for the first pair.
  # The result will be in the same table.
  #

  CtoRsteps:=proc(A,T,i) local k,j,z;
  k:=i; j:=2*i-1;
  while j>k do
    z:=CtoRstep(A[k],A[j],T[k]); A[k]:=z[1]; A[j]:=z[2]; k:=
  k+1; j:=j-1;
  od; end;
```

$CtoRsteps :=$ **proc**(A, T, i) (7.5.3)

local $k, j, z;$

$k := i;$

$j := 2 * i - 1;$

while $k < j$ **do**

$z := CtoRstep(A[k], A[j], T[k]);$

$A[k] := z[1];$

$A[j] := z[2];$

$k := k + 1;$

$j := j - 1$

end do

end proc

```
> #
```

```
# The RtoC procedure do the conversion from real
# representation to complex representation.
# The result will be in the same table.
#

RtoC:=proc(A,T,k) local i,x,y;
x:=Re(A[0]); y:=Im(A[0]);
A[0]:=x+y+I*(x-y);
x:=Re(A[1]); y:=Im(A[1]);
A[1]:=2*x-I*2*y;
for i to k-1 do RtoCsteps(A,T,2^i); od;
end;
```

$RtoC := \textbf{proc}(A, T, k)$ (7.5.4)

local $i, x, y;$

$x := \Re(A[0]);$

$y := \Im(A[0]);$

$A[0] := x + y + \mathrm{I} * (x - y);$

$x := \Re(A[1]);$

$y := \Im(A[1]);$

$A[1] := 2 * x - 2 * \mathrm{I} * y;$

for i **to** $k - 1$ **do**

$RtoCsteps(A, T, 2 \wedge i)$

end do

end proc

```
> #
# The RtoCstep procedure do the conversion from real
representation
# to complex representation for one pair ll, uu using weight
w.
#

RtoCstep:=proc(ll,uu,w) local al,be,ga,de,xi,et,a,b,x,y,u,v;
xi:=Re(w); et:=Im(w);
al:=Re(ll); be:=Im(ll);
ga:=Re(uu); de:=Im(uu);
x:=al+ga; y:=be-de;
a:=al-ga; b:=be+de;
u:=xi*a+et*b; v:=xi*b-et*a;
[x-v+I*(u+y),x+v+I*(u-y)]
end;
```

$RtoCstep := \textbf{proc}(ll, uu, w)$ (7.5.5)

local $al, be, ga, de, \xi, et, a, b, x, y, u, v;$

$\xi := \Re(w);$

$et := \Im(w);$

$al := \Re(ll);$

$be := \Im(ll);$

$ga := \Re(uu);$

$de := \Im(uu);$

$x := al + ga;$

$y := be - de;$

$a := al - ga;$

$b := be + de;$

$u := \xi * a + et * b;$

$v := \xi * b - et * a;$

$[x - v + I * (y + u), x + v + I * (u - y)]$

end proc

```
> #
  # The RtoCsteps procedure do the conversion from real
  # representation to complex representation for one series.
  # The index i is the lower index for the first pair.
  #

  RtoCsteps:=proc(A,T,i) local k,j,z;
  k:=i; j:=2*i-1;
  while j>k do
    z:=RtoCstep(A[k],A[j],T[k]);
    A[k]:=z[1]; A[j]:=z[2]; k:=k+1; j:=j-1; od;
  end;
```

$$RtoCsteps := \textbf{proc}(A, T, i) \tag{7.5.6}$$

local $k, j, z;$

$k := i;$

$j := 2 * i - 1;$

while $k < j$ **do**

$z := RtoCstep(A[k], A[j], T[k]);$

$A[k] := z[1];$

$A[j] := z[2];$

$k := k + 1;$

$j := j - 1$

```
    end do
end proc
```

7.6. Szorzás komplex FFT-vel a gyakorlatban.

```
> #
  # The srfftpre procedure do the preparation for the
  # signed table for rfft; x is the number, A is the
  # table, m the even positive modulus, and k is the
  # number of rounds for the rfft, hence the table is 2^k long.
  # All number is multipled by the factor f.
  #

  srfftpre:=proc(x,A,m,k,f) local i,c,d,xx;
  xx:=x; c:=0;
  for i from 0 to 2^k-1 do
    d:=irem(xx,m,'xx');
    if d>=m/2 then d:=d-m+c; c:=1; else d:=d+c; c:=0; fi;
    A[i]:=evalf(d*f);
    d:=irem(xx,m,'xx');
    if d>=m/2 then d:=d-m+c; c:=1; else d:=d+c; c:=0; fi;
    A[i]:=A[i]+I*evalf(d*f);
  od; end;
```

srfftpre := **proc**(x, A, m, k, f) (7.6.1)

```
    local i, c, d, xx;
    xx := x;
    c := 0;
    for i from 0 to 2^k - 1 do
        d := irem(xx, m, 'xx');
        if 1/2*m <= d then
            d := d - m + c;
            c := 1
        else
            d := c + d;
            c := 0
        end if;
        A[i] := evalf(d*f);
        d := irem(xx, m, 'xx');
        if 1/2*m <= d then
```

$d := d - m + c;$

$c := 1$

else

$d := c + d;$

$c := 0$

end if;

$A[i] := A[i] + \mathrm{I} * evalf(d * f)$

end do

end proc

```
> #
  # The rnorm procedure do the normalization after the irfft.
  # A is the table, m the modulus, and k is the number of
  # rounds for the rfft, hence the table is 2^k long.
  # Before conversion, all entry in A multiplied by the factor
  f.
  # The fraction parts are left in the table A.
  #

  rnorm:=proc(A,m,k,f) local i,x;
  x:=0;
  for i from 2^k-1 to 0 by -1 do
    A[i]:=evalf(A[i]*f);
    x:=m*x+round(Im(A[i]));
    A[i]:=A[i]-I*round(Im(A[i]));
    x:=m*x+round(Re(A[i]));
    A[i]:=A[i]-round(Re(A[i]));
  od; x; end;
```

$rnorm :=$ **proc**(A, m, k, f) (7.6.2)

local $i, x;$

$x := 0;$

for i **from** $2^{\wedge}k - 1$ **by** -1 **to** 0 **do**

$A[i] := evalf(A[i] * f);$

$x := m * x + \mathrm{round}(\Im(A[i]));$

$A[i] := A[i] - \mathrm{I} * \mathrm{round}(\Im(A[i]));$

$x := m * x + \mathrm{round}(\Re(A[i]));$

$A[i] := A[i] - \mathrm{round}(\Re(A[i]))$

end do;

x

end proc

```
> #
  # The rdigmul procedure do the digit-by-digit
  # multiplication of the two numbers after the
  # rfft's. The result will be in the first table.
  #

  rdigmul:=proc(A,B,k) local i;
  A[0]:=Re(A[0])*Re(B[0])+I*Im(A[0])*Im(B[0]);
  for i from 1 to 2^k-1 do A[i]:=A[i]*B[i]; od;
  end;
```

$rdigmul :=$ **proc**(A, B, k) (7.6.3)

local i;

$A[0] := \Re(A[0]) * \Re(B[0]) + I * \Im(A[0]) * \Im(B[0])$;

for i **to** $2^{\wedge}k - 1$ **do**

$A[i] := A[i] * B[i]$

end do

end proc

```
> #
  # This is the main procedure, do the multiplication
  # or the squaring using real FFT and IFFF.
  # a and b are the two numbers, m is the modulus for
  # the preparation. The FFT and IFFT use k rounds.
  #

  mulrfft:=proc(a,b,m,k) local f1,f2,r; global A,B,T;
  if type(k,odd) then
    f1:=2^(-(k+1)/2-HWS);
    f2:=2^(2*HWS-2);
  else
    f1:=2^(-k/2-HWS);
    f2:=2^(2*HWS-3);
  fi;
  if a=b then
    srfftpre(a,A,m,k,f1); print(A);
    cfft(A,T,k); print(A);
    CtoR(A,T,k); print(A);
    rdigmul(A,A,k); print(A);
    RtoC(A,T,k); print(A);
    icfft(A,T,k); print(A);
    rnorm(A,m,k,f2);
  else
    srfftpre(a,A,m,k,f1);
    cfft(A,T,k);
```

```
    CtoR(A,T,k);
    srfftpre(b,B,m,k,f1);
    cfft(B,T,k);
    CtoR(B,T,k);
    rdigmul(A,B,k);
    RtoC(A,T,k);
    icfft(A,T,k);
    rnorm(A,m,k,f2);
  fi; end;
```

mulrfft := **proc**(*a*, *b*, *m*, *k*) (7.6.4)

local *f1*, *f2*, *r*;

global *A*, *B*, *T*;

if *type*(*k*, *odd*) **then**

f1 := 2^(−1 / 2 * *k* − 1 / 2 − *HWS*);

f2 := 2^(2 * *HWS* − 2)

else

f1 := 2^(−1 / 2 * *k* − *HWS*);

f2 := 2^(2 * *HWS* − 3)

end if;

if *a* = *b* **then**

srfftpre(*a*, *A*, *m*, *k*, *f1*);

print(*A*);

cfft(*A*, *T*, *k*);

print(*A*);

CtoR(*A*, *T*, *k*);

print(*A*);

rdigmul(*A*, *A*, *k*);

print(*A*);

RtoC(*A*, *T*, *k*);

print(*A*);

icfft(*A*, *T*, *k*);

print(*A*);

rnorm(*A*, *m*, *k*, *f2*)

else

srfftpre(*a*, *A*, *m*, *k*, *f1*);

```
            cfft(A, T, k);
            CtoR(A, T, k);
            srfftpre(b, B, m, k, f1);
            cfft(B, T, k);
            CtoR(B, T, k);
            rdigmul(A, B, k);
            RtoC(A, T, k);
            icfft(A, T, k);
            rnorm(A, m, k, f2)
      end if
  end proc
```

```
> HWS:=16; k:=5;
  A:=array(0..2^k-1); B:=array(0..2^k-1);
  T:=array(0..2^k-1); pre(T,k):
```

$$HWS := 16$$

$$k := 5$$

$$A := array(0..31, [\,])$$

$$B := array(0..31, [\,])$$

$$T := array(0..31, [\,]) \tag{7.6.5}$$

```
> mulrfft(123456789,987654321,18,5);
```

$$121932631112635269 \tag{7.6.6}$$

```
> 123456789*987654321;
```

$$121932631112635269 \tag{7.6.7}$$

```
> print(A);
```

$$ARRAY([0..31], [0 = 6.\,10^{-8} - 2.\,10^{-8}\,I, 1 = 1.\,10^{-8} - 4.\,10^{-8}\,I, 2 \tag{7.6.8}$$

$$= 2.\,10^{-8} - 1.\,10^{-8}\,I, 3 = 1.\,10^{-8} - 2.\,10^{-8}\,I, 4 = 0. + 1.\,10^{-8}\,I, 5 = -1.\,10^{-8}$$

$$+ 1.\,10^{-8}\,I, 6 = -1.\,10^{-8} - 6.\,10^{-9}\,I, 7$$

$$= 1.194142679\,10^{-8} - 1.675746100\,10^{-8}\,I, 8$$

$$= 0. - 1.073741824\,10^{-8}\,I, 9 = -1.073741824\,10^{-9}$$

$$+ 9.663676416\,10^{-9}\,I, 10$$

$$= -4.294967296\,10^{-9} - 3.221225472\,10^{-9}\,I, 11 = -1.073741824\,10^{-8}$$

$$+ 1.073741824\,10^{-8}\,I, 12 = 2.147483648\,10^{-8} - 2.147483648\,10^{-8}\,I, 13$$

$$= 1.073741824\,10^{-8} + 0.\,I, 14 = -1.610612736\,10^{-8}$$

$+ 9.663676416\ 10^{-9}\ I,\ 15$
$= -2.319991194\ 10^{-8} - 2.016291144\ 10^{-8}\ I,\ 16$
$= 0. - 1.073741824\ 10^{-8}\ I,\ 17 = 4.294967296\ 10^{-9}$
$+ 0.\ I,\ 18 = 0. - 1.073741824\ 10^{-8}\ I,\ 19 = 1.073741824\ 10^{-8}$
$+ 1.073741824\ 10^{-8}\ I,\ 20 = -1.073741824\ 10^{-8}$
$+ 3.221225472\ 10^{-8}\ I,\ 21 = -1.073741824\ 10^{-8}$
$+ 0.\ I,\ 22 = 2.147483648\ 10^{-8} - 1.395864371\ 10^{-8}\ I,\ 23$
$= 7.385926042\ 10^{-9}$
$+ 6.020042757\ 10^{-9}\ I,\ 24 = 0. - 1.073741824\ 10^{-8}\ I,\ 25$
$= -2.254857830\ 10^{-8} + 1.181116006\ 10^{-8}\ I,\ 26 = -1.717986918\ 10^{-8}$
$+ 1.825361101\ 10^{-8}\ I,\ 27 = 1.073741824\ 10^{-8} - 1.073741824\ 10^{-8}\ I,\ 28$
$= 0. - 2.147483648\ 10^{-8}\ I,\ 29 = 1.073741824\ 10^{-8}$
$+ 0.\ I,\ 30 = 1.073741824\ 10^{-9} - 3.221225472\ 10^{-9}\ I,\ 31$
$= -4.224081867\ 10^{-10} - 3.459408689\ 10^{-9}\ I])$

7.7. Példa.

```
> HWS:=16; k:=15;
  A:=array(0..2^k-1); B:=array(0..2^k-1);
  T:=array(0..2^k-1); pre(T,k):
```

$$HWS := 16$$

$$k := 15$$

$$A := array(0..32767, [\,])$$

$$B := array(0..32767, [\,])$$

$$T := array(0..32767, [\,]) \quad (7.7.1)$$

```
> mulrfft(123456789,987654321,16,5);
```

$$121932631112635269 \quad (7.7.2)$$

```
> 123456789*987654321;
```

$$121932631112635269 \quad (7.7.3)$$

7.8. FFT véges testek felett.

```
>
```

7.9. Fermat-szám transzformáció.

```
> #
  # This procedure adds 1 to the number represented by the
  # bit vector b (used in reverse bit order) on length k.
  #

  incrementreverse:=proc(b,k) local i,c;
  c:=b;
  for i to k do if c[i]=0 then c[i]:=1; return c; else c[i]:=0
  fi;
  od; c; end;
```

incrementreverse := **proc**(b, k) (7.9.1)

```
    local i, c;
    c := b;
    for i to k do
        if c[i] = 0 then
            c[i] := 1;
            return c
        else
            c[i] := 0
        end if
    end do;
    c
end proc
```

```
> #
  # This procedure convert the bits in interval II of the
  # bit vector b. The bit b[1] represents the highest bit,
  # 2^(m-2+op(1,II)), usually 2^(m-1).
  #

  convertreverse:=proc(b,m,II) local i,lw; lw:=0;
  for i from op(1,II) to op(2,II) do
    lw:=lw+b[i]*2^(m-1-i+op(1,II))
  od; lw; end;
```

convertreverse := **proc**(b, m, II) (7.9.2)

```
    local i, lw;
    lw := 0;
    for i from op(1, II) to op(2, II) do
```

$lw := lw + b[i] * 2^{(m - 1 - i + op(1, II))}$

end do;

lw

end proc

```
> #
  # This procedure do a Fermat FFT round modulo 2^(2^m)+1 on
  # array A having 2^n elements. The siccor size is 2^k and
  # the weights are get from the conversion of bits in interval
  II
  # from a counter starting with zero and incremented after
  # each butterfly sequence.
  #

  ffftround:=proc(A,n,m,k,II) local i,j,x,y,w,b;
  j:=2^k; b:=[0$i=1..m+1];
  while j<2^n do
    w:=2^convertreverse(b,m,II); b:=incrementreverse(b,m+1);
    for i from j-2^k while i<j do
      x:=A[i]; y:=A[i+2^k];
      A[i]:=x+w*y mod (2^(2^m)+1); A[i+2^k]:=x-w*y mod (2^(2^m)
  +1);
    od; j:=j+2^(k+1);
  od; end;
```

ffftround := **proc**(*A*, *n*, *m*, *k*, *II*) (7.9.3)

local *i*, *j*, *x*, *y*, *w*, *b*;

$j := 2^k$;

$b := [\$(0, i = 1 .. m + 1)]$;

while $j < 2^n$ **do**

$w := 2^{convertreverse(b, m, II)}$;

$b := incrementreverse(b, m + 1)$;

for *i* **from** $j - 2^k$ **while** $i < j$ **do**

$x := A[i]$;

$y := A[i + 2^k]$;

$A[i] := mod(x + w * y, 2^{(2^m)} + 1)$;

$A[i + 2^k] := mod(x - w * y, 2^{(2^m)} + 1)$

end do;

$j := j + 2^{(k + 1)}$

end do

end proc

```
> #
  # This procedure do a Fermat IFFT round modulo 2^(2^m)+1 on
  # array A having 2^n elements. The siccor size is 2^k and
  # the weights are get from the conversion of bits in interval
  II
  # from a counter starting with zero and incremented after
  # each butterfly sequence.
  #

  iffftround:=proc(A,n,m,k,II) local i,j,x,y,w,b;
  j:=2^k; b:=[0$i=1..m+1];
  while j<2^n do
    w:=2^convertreverse(b,m,II); b:=incrementreverse(b,m+1);
    for i from j-2^k while i<j do
      x:=A[i]; y:=A[i+2^k];
      A[i]:=x+y mod (2^(2^m)+1); A[i+2^k]:=(x-y)/w mod (2^(2^m)
  +1);
    od; j:=j+2^(k+1);
  od; end;
```

iffftround := **proc**(*A*, *n*, *m*, *k*, *II*) (7.9.4)

```
local i, j, x, y, w, b;
j := 2^k;
b := [$(0, i = 1..m + 1)];
while j < 2^n do
    w := 2^convertreverse(b, m, II);
    b := incrementreverse(b, m + 1);
    for i from j − 2^k while i < j do
        x := A[i];
        y := A[i + 2^k];
        A[i] := mod(x + y, 2^(2^m) + 1);
        A[i + 2^k] := mod((x − y) / w, 2^(2^m) + 1)
    end do;
    j := j + 2^(k + 1)
end do
end proc
```

```
> #
  # This is the digit-by-digit multiplication modulo 2^(2^m)+1
  # of arrays A and B having 2^n elements.
  #
```

```
fdigmul:=proc(A,B,n,m) local i;
for i from 0 to 2^n-1 do A[i]:=A[i]*B[i] mod (2^(2^m)+1) od;
end;
```

fdigmul := **proc**(*A*, *B*, *n*, *m*) (7.9.5)

local *i*;

for *i* **from** 0 **to** 2^*n* − 1 **do**

A[*i*] := *mod*(*A*[*i*]*B*[*i*], 2^(2^*m*) + 1)

end do

end proc

```
> #
# This procedure divides by 2^k modulo 2^(2^m)+1
# all elements of array A having 2^n elements.
#

fshift:=proc(A,n,m,k) local i,r;
r:=1/2^k mod (2^(2^m)+1);
for i from 0 to 2^n-1 do A[i]:=A[i]*r mod (2^(2^m)+1) od;
end;
```

fshift := **proc**(*A*, *n*, *m*, *k*) (7.9.6)

local *i*, *r*;

r := *mod*(1 / 2^*k*, 2^(2^*m*) + 1);

for *i* **from** 0 **to** 2^*n* − 1 **do**

A[*i*] := *mod*(*A*[*i*]**r*, 2^(2^*m*) + 1)

end do

end proc

```
> #
# This procedure multiplies by sqrt(2) modulo 2^(2^m)+1
# all odd-indexed elements in the upper half of
# array A having 2^n elements.
#

mulsqrt:=proc(A,n,m) local i,sqrttwo;
sqrttwo:=2^(3*2^(m-2))-2^(2^(m-2));
for i from 2^(n-1)+1 to 2^n-1 by 2 do
  A[i]:=A[i]*sqrttwo mod (2^(2^m)+1)
od; end;
```

mulsqrt := **proc**(*A*, *n*, *m*) (7.9.7)

local *i*, *sqrttwo*;

sqrttwo := 2^(3*2^(*m* − 2)) − 2^(2^(*m* − 2));

for *i* **from** 2^(*n* − 1) + 1 **by** 2 **to** 2^*n* − 1 **do**

$A[i] := mod(A[i]*sqrttwo, 2^(2^m)+1)$

end do

end proc

```
> #
  # This procedure divides by sqrt(2) modulo 2^(2^m)+1
  # all odd-indexed elements in the upper half of
  # array A having 2^n elements.
  #

  divsqrt:=proc(A,n,m) local i,sqrthalf;
  sqrthalf:=1/(2^(3*2^(m-2))-2^(2^(m-2))) mod (2^(2^m)+1);
  for i from 2^(n-1)+1 to 2^n-1 by 2 do
    A[i]:=A[i]*sqrthalf mod (2^(2^m)+1)
  od; end;
```

$divsqrt :=$ **proc**(A, n, m) (7.9.8)

local $i, sqrthalf;$

$sqrthalf := mod(1/(2^(3*2^(m-2)) - 2^(2^(m-2))), 2^(2^m)+1);$

for i **from** $2^(n-1)+1$ **by** 2 **to** 2^n-1 **do**

$A[i] := mod(A[i]*sqrthalf, 2^(2^m)+1)$

end do

end proc

```
> #
  # This procedure do Fermat FFT modulo 2^(2^m)+1
  # for array A having 2^n elements.
  #

  ffft:=proc(A,n,m) local i;
  for i from 0 to n-2 do ffftround(A,n,m,n-1-i,1..i) od;
  if m>=n-1 then
    ffftround(A,n,m,0,1..n-1)
  else
    mulsqrt(A,n,m); ffftround(A,n,m,0,1..n-2)
  fi; end;
```

$ffft :=$ **proc**(A, n, m) (7.9.9)

local $i;$

for i **from** 0 **to** $n-2$ **do**

$ffftround(A, n, m, n-1-i, 1..i)$

end do;

if $n-1 <= m$ **then**

fffftround(*A*, *n*, *m*, 0, 1 ..*n* − 1)

else

mulsqrt(*A*, *n*, *m*);

fffftround(*A*, *n*, *m*, 0, 1 ..*n* − 2)

end if

end proc

```
> #
  # This procedure do Fermat IFFT modulo 2^(2^m)+1
  # for array A having 2^n elements.
  #

  iffft:=proc(A,n,m) local i;
  if m>=n-1 then
    iffftround(A,n,m,0,1..n-1)
  else
    iffftround(A,n,m,0,1..n-2); divsqrt(A,n,m)
  fi;
  for i from n-2 to 0 by -1 do iffftround(A,n,m,n-1-i,1..i) od;
  end;
```

iffft := **proc**(*A*, *n*, *m*) (7.9.10)

local *i*;

if *n* − 1 <= *m* **then**

iffftround(*A*, *n*, *m*, 0, 1 ..*n* − 1)

else

iffftround(*A*, *n*, *m*, 0, 1 ..*n* − 2);

divsqrt(*A*, *n*, *m*)

end if;

for *i* **from** *n* − 2 **by** −1 **to** 0 **do**

iffftround(*A*, *n*, *m*, *n* − 1 − *i*, 1 ..*i*)

end do

end proc

```
> #
  # This procedure do multiplication or squaring using Fermat
  FFT
  # and IFFT modulo 2^(2^m)+1 for array having 2^n elements.
  # 2^(m-1)-k bits is in one array element.
  #

  ffftmul:=proc(a,b,n,m) local i,c,A,B;
```

```
if a=b then
  A:=Array(0..2^n-1,convert(a,base,2^(2^(m-1)-k)));
  ffft(A,n,m);
  fdigmul(A,A,n,m);
  iffft(A,n,m);
  fshift(A,n,m,n);
  c:=0; for i from 0 to 2^n-1 do c:=c+(2^(2^(m-1)-k))^i*A[i]
od;
else
  A:=Array(0..2^n-1,convert(a,base,2^(2^(m-1)-k)));
  ffft(A,n,m);
  B:=Array(0..2^n-1,convert(b,base,2^(2^(m-1)-k)));
  ffft(B,n,m);
  fdigmul(A,B,n,m);
  iffft(A,n,m);
  fshift(A,n,m,n);
  c:=0; for i from 0 to 2^n-1 do c:=c+(2^(2^(m-1)-k))^i*A[i]
od;
fi; c; end;
```

$$ffftmul := \mathbf{proc}(a, b, n, m) \tag{7.9.11}$$

```
local i, c, A, B;
if a = b then
    A := Array(0..2^n − 1, convert(a, base, 2^(2^(m − 1) − k)));
    ffft(A, n, m);
    fdigmul(A, A, n, m);
    iffft(A, n, m);
    fshift(A, n, m, n);
    c := 0;
    for i from 0 to 2^n − 1 do
        c := c + (2^(2^(m − 1) − k))^i*A[i]
    end do
else
    A := Array(0..2^n − 1, convert(a, base, 2^(2^(m − 1) − k)));
    ffft(A, n, m);
    B := Array(0..2^n − 1, convert(b,
    base, 2^(2^(m − 1) − k)));
    ffft(B, n, m);
    fdigmul(A, B, n, m);
    iffft(A, n, m);
```

```
        fshift(A, n, m, n);
        c := 0;
        for i from 0 to 2^n - 1 do
            c := c + (2^(2^(m - 1) - k))^i*A[i]
        end do
    end if;
    c
end proc
```

```
> n:=6; m:=4; k:=4; ffftmul(123456789,987654321,n,m,k);
```

$$n := 6$$

$$m := 4$$

$$k := 4$$

$$121932631112635269 \qquad (7.9.12)$$

```
> 123456789*987654321;
```

$$121932631112635269 \qquad (7.9.13)$$

7.10. Schönhage–Strassen–féle gyorsszorzó algoritmus.

```
>
```

7.11. Példa.

```
>
```

7.12. Ritka polinomok es ritka számok.

7.13. Feladat.

7.14. Osztás, polinomosztás.

7.15. Polinom kiértékelése tetszőleges helyeken.

7.16. Interpoláció.

7.17. Feladat.

7.18. Feladat.