Arithmétique entière

Paul Zimmermann

Équipe-projet CACAO

Plan

Partie 1 : addition et multiplication.

Partie 2 : division et pgcd.

Références

- *The Art of Computer Programming, volume 2 : Seminumerical Algorithms*, Knuth, Addison-Wesley, 3e édition, 1998.
- *Fast Algorithms, A Multitape Turing Machine Implementation*, Schönhage, Grotefeld et Vetter, BI-Wissenschaftsverlag, 1994.
- *Modern Computer Algebra*, von zur Gathen et Gerhard, CUP, 2e édition, 2003.
- *Reduce Everything to Multiplication*, Steel, `http://www.mathematik.hu-berlin.de/~gaggle/EVENTS/2006/BRENT60/`, 2006.
- *Modern Computer Arithmetic*, Brent et Z., en préparation, `http://wwwmaths.anu.edu.au/~brent/pub/pub226.html`.

Ce qu'il faut retenir

$$n \log n \log\log n \ll M(n) \ll n^2$$

- Addition, soustraction : $O(n)$;
- Multiplication, division, racine carrée : $O(M(n))$;
- pgcd, conversion : $O(M(n) \log n)$;
- exponentiation modulaire (RSA) : $O(nM(n))$.

Addition et multiplication

Lien avec arithmétique polynomiale

$$
\begin{aligned}
p &= 860x^4 + 758x^3 + 750x^2 + 889x + 300 \\
q &= 991x^4 + 5x^3 + 993x^2 + 954x + 299 \\
p + q &= 1851x^4 + 763x^3 + 1743x^2 + 1843x + 599
\end{aligned}
$$

$$
\begin{aligned}
A &= 860758750889300 \\
B &= 991005993954299 \\
A + B &= 1851764744843599
\end{aligned}
$$

Représentation dense vs creuse

- *dense* en base β, exemple 889000991 avec $\beta = 10^3$:

$$[889, 000, 991] = 889\beta^2 + 000\beta^1 + 991\beta^0$$

- *creuse* :

$$[(889, 2), (991, 0)] = 889\beta^2 + 991\beta^0$$

Dans les deux cas, β est implicite.

Choix de la base interne β

- la plus grande possible, 2^{32} ou 2^{64} (GMP) ;
- la plus grande telle que $\beta^2 \leq 2^w$, ou $c\beta^2 \leq 2^w$, ex. MP (Brent) : $8\beta^2 \leq 2^w$;
- idem en décimal (Maple) : $\beta = 10^4$ sur machine 32 bits, $\beta = 10^9$ sur machine 64 bits.

Choix de la base interne β (suite)

Exemple : $3^{2095903}$ (10^6 chiffres décimaux) sur machine 32 bits.

- GMP : 103811 mots de 32 bits, soit 415Ko ;
- MP : $\beta = \lfloor \sqrt{2^{32}/8} \rfloor = 23170$ donne 916Ko (+121%), $\beta = 16384$ donne 949Ko (+129%) ;
- Maple : 10^6 octets (+141%).

```
    |\^/|     Maple 6 (IBM INTEL LINUX22)
._|\|   |/|_. Copyright (c) 2000 by Waterloo Maple Inc.
 \  MAPLE  /  All rights reserved. Maple is a registered trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> ?kernelopts
    memusage        list       displays a matrix of memory usage by
                               Maple internal objects. Each row
                               contains the object type name, the
                               number of objects of that type, and
                               the number of bytes of storage occupied
                               by such objects. Cannot be set.
> kernelopts(memusage);
                          [ INTPOS        2       32]
> a:=3^2095903:
> kernelopts(memusage);
                        [ INTPOS        2    1000020]
```

Représentation à base de flottants

$$889000991 = 889.0E6 + 991.0E0$$

Avantages :

- pas besoin de stocker les exposants (pour représentation creuse) ;
- sur processeur 32 bits on a 53 bits (voire 64) par « mot » ;
- sur certains processeurs (Sparc) les opérations flottantes sont plus rapides (i.e., les opérations entières sont lentes).

Inconvénients :

- utilise plus de mémoire ($53 + 1$ bits sur 64) ;
- exposants limités ($2^{1024} \approx 1.8 \cdot 10^{308}$) : on peut ne pas les utiliser.

Complexité théorique vs complexité pratique

Pentium 4 à 3Ghz avec $a = 3^{2095903}$, $b = 7^{1183294}$, $c = 11^{1920505}$.

a et b ont 10^6 chiffres, c a $2 \cdot 10^6$ chiffres, temps en secondes :

	GMP 4.2.1	Maple 6	Maple 7	Maple 8	Maple 9	Maple 10
$a+b$	0.00044	0.0087	.0089	0.0084	0.0017	0.0015
ab	0.17	9.63	9.59	9.91	0.43	0.29
c/a	0.78	1347.	19.23	19.31	2.07	1.09
$\sqrt{c}$	0.64	3109.	3108.	3081.	2.11	1.10
$\gcd(a,b)$	69.99	$>$2h	$>$2h	$>$2h	107.5	68.21

GMP :

```
$ configure; make
$ cd tune; make speed; ./speed -s 103811 mpn_add_n
            mpn_add_n
103811    0.000439543
```

GMP :

```
$ configure; make
$ cd tune; make speed; ./speed -s 103811 mpn_add_n
            mpn_add_n
103811    0.000439543
```

Maple 9 :

```
> ?GMP
> kernelopts(gmpthreshold), 10^kernelopts(gmpthreshold);
108, 1000000000000000000000000000000000000000000000000000000000000\
    000000000000000000000000000000000000000000000000000000000000
```

GMP :

```
$ configure; make
$ cd tune; make speed; ./speed -s 103811 mpn_add_n
            mpn_add_n
103811    0.000439543
```

Maple 9 :

```
> ?GMP
> kernelopts(gmpthreshold), 10^kernelopts(gmpthreshold);
108, 1000000000000000000000000000000000000000000000000000\
    00000000000000000000000000000000000000000000000000
```

Maple 10 :

```
> kernelopts(maximmediate);
                                    1073741823
```

Addition

Algorithm **IntegerAddition**.

Input : $A = \sum_0^{n-1} a_i \beta^i, \quad B = \sum_0^{n-1} b_i \beta^i$

Output : $C := \sum_0^{n-1} c_i \beta^i$ et $0 \leq d \leq 1 : A + B = d\beta^n + C$

$d \leftarrow 0$

for i from 0 to $n-1$ do

$s \leftarrow a_i + b_i + d$

$c_i \leftarrow s \bmod \beta$

$d \leftarrow s \operatorname{div} \beta$

Return C, d.

Addition (2)

Comment calculer $s \leftarrow a_i + b_i + d$?

On a $0 \leq a_i, b_i \leq \beta - 1$ et $0 \leq d \leq 1$, donc

$$0 \leq s \leq 2\beta - 1$$

Ou bien $2\beta < 2^w$, ou bien il faut détecter la retenue possible :

```
s = ai + d;
d = (s < ai);
ci = s + bi;
d += (ci < bi);
```

Segmentation (Kronecker-Schönhage)

Soit à multiplier p et q deux polynômes à coefficients entiers :

```
> p := 860*x^4+758*x^3+750*x^2+889*x+300:
> q := 991*x^4+5*x^3+993*x^2+954*x+299:
> P := subs(x=10^7, p);
             P := 860000075800007500000889000030 0

> Q := subs(x=10^7, q);
             Q := 991000000500009930000954000029 9

> P*Q;
  852260075547816010202457883202676718264191370256055201100897 00

> sort(expand(p*q));
        8           7            6            5            4
852260 x  + 755478 x  + 1601020 x  + 2457883 x  + 2026767 x

                3            2
     + 1826419 x  + 1370256 x  + 552011 x + 89700
```

Segmentation (suite)

Soit à multiplier p et q de degré $< n$, avec $0 \leq p_i, q_i \leq B$.

Les coefficients de pq sont bornés par nB^2.

Soit $A = p(2^m)$, $B = q(2^m)$.

Si $2^m > nB^2$, il suffit de « lire » les coefficients de pq dans la décomposition en base 2^m de AB.

Question : étendre à des coefficients négatifs ?

Solution

Supposons $B/2 \leq p_i, q_i \leq B/2$.

On peut supposer les coefficients dominants de p, q positifs.

On évalue p et q en base $2^m > nB^2$: soit P et Q.

On multiplie P et Q, et on convertit le produit en base 2^m.

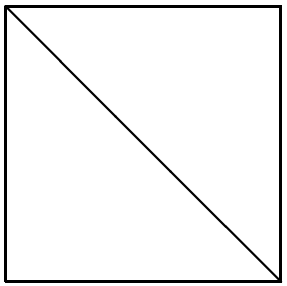
Si le bit de poids fort est 1, alors on a un coefficient négatif : retrancher 2^m, et ajouter 1 au suivant.

Attention : il faut extraire les coefficients par degrés croissants !

Produit court

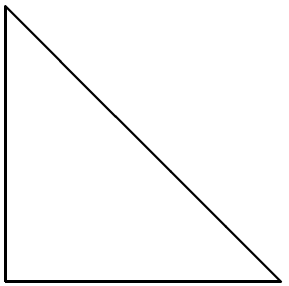
Le *produit court* bas (resp. haut) de deux entiers de n chiffres est constitué des n chiffres de poids faible (resp. fort) de leur produit :

```
> A := 42578166857774688355:
> B := 97371105729070091546:
> A*B;
          41458831868583672550156374758214701468 30

> A*B mod 10^20;
                    15637475821470146830

> iquo(A*B, 10^20);
                    41458831868583672550
```



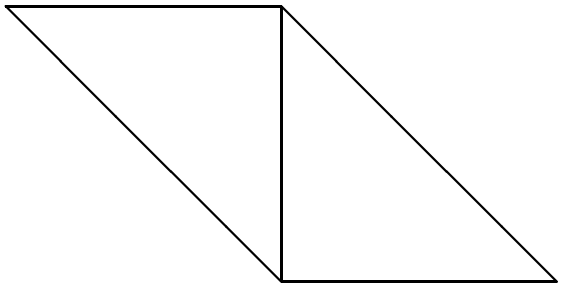
Multiplication

$M(n)$



Produit court

$M^*(n)$



Produit médian

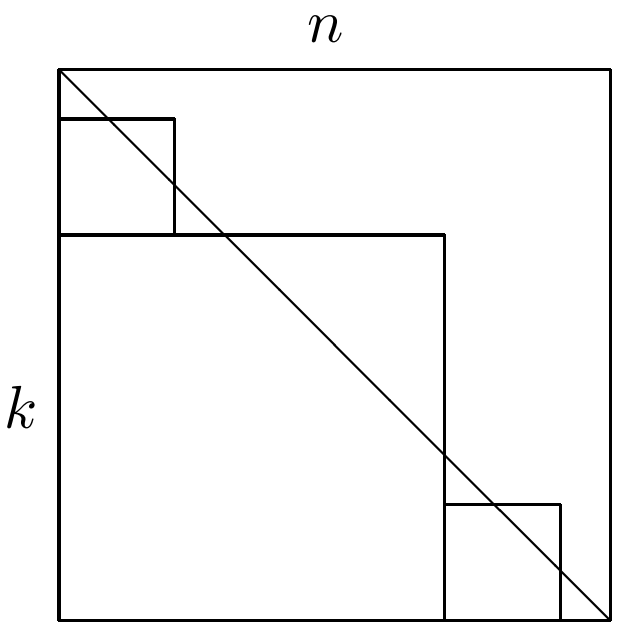
$MP(n)$

$$M^*(n) \leq M(n) \leq 2M^*(n)$$

$$MP(n) \leq 2M^*(n) \leq 2M(n)$$

Tellegen's principle into practice, Bostan, Lecerf, Schost, ISSAC'03, 2003.

Calcul rapide d'un produit court



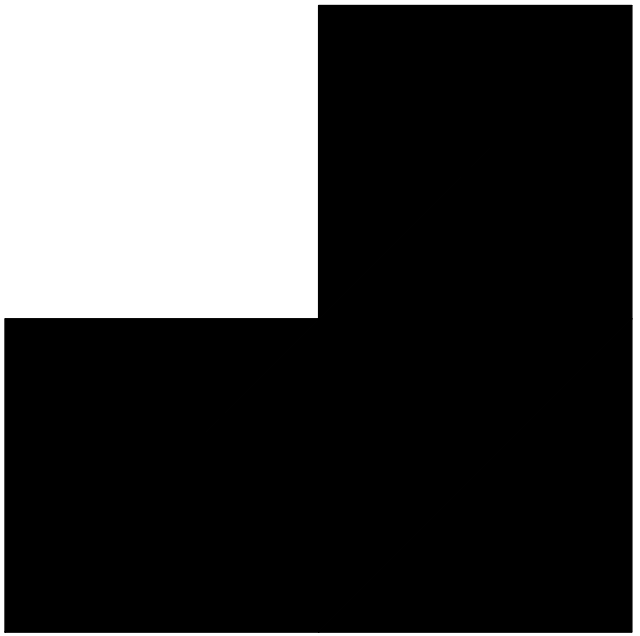
Mulders, *On short multiplications and divisions*, AAECC, 2000.

Karatsuba : $k \approx 0.69n$, gain théorique de 20%.

Analogues poids forts/faibles

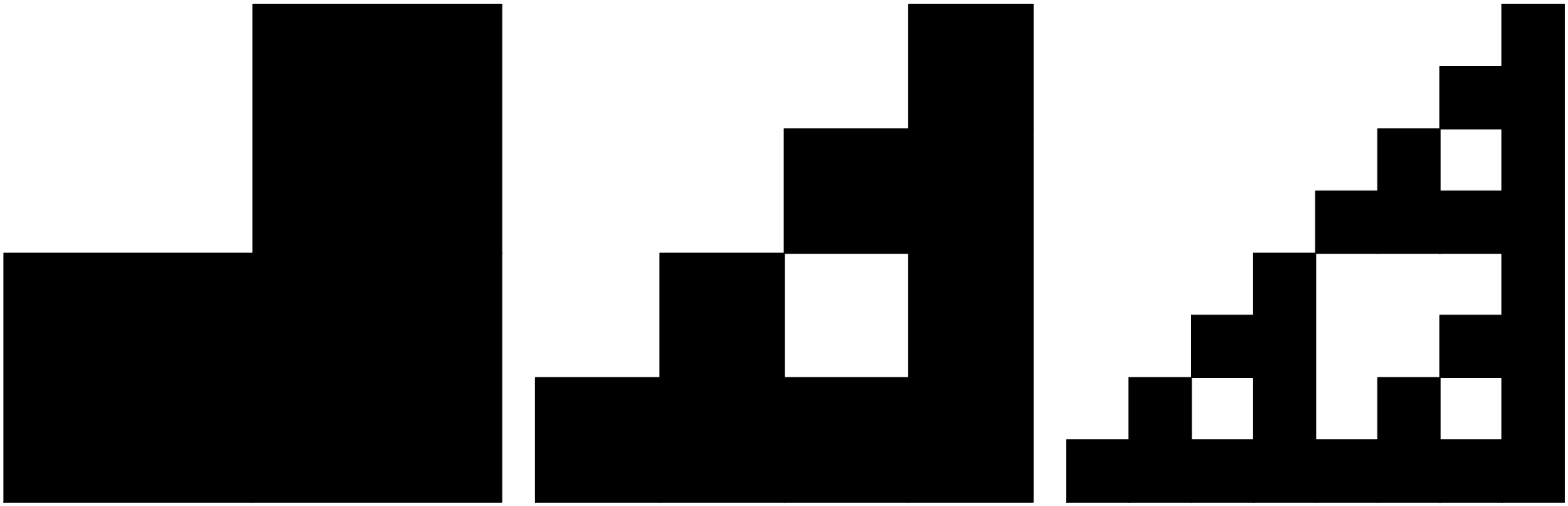
poids forts	poids faibles
produit court haut	produit court bas
division classique	division 2-adique
pgcd classique	pgcd 2-adique

Multiplication de Karatsuba

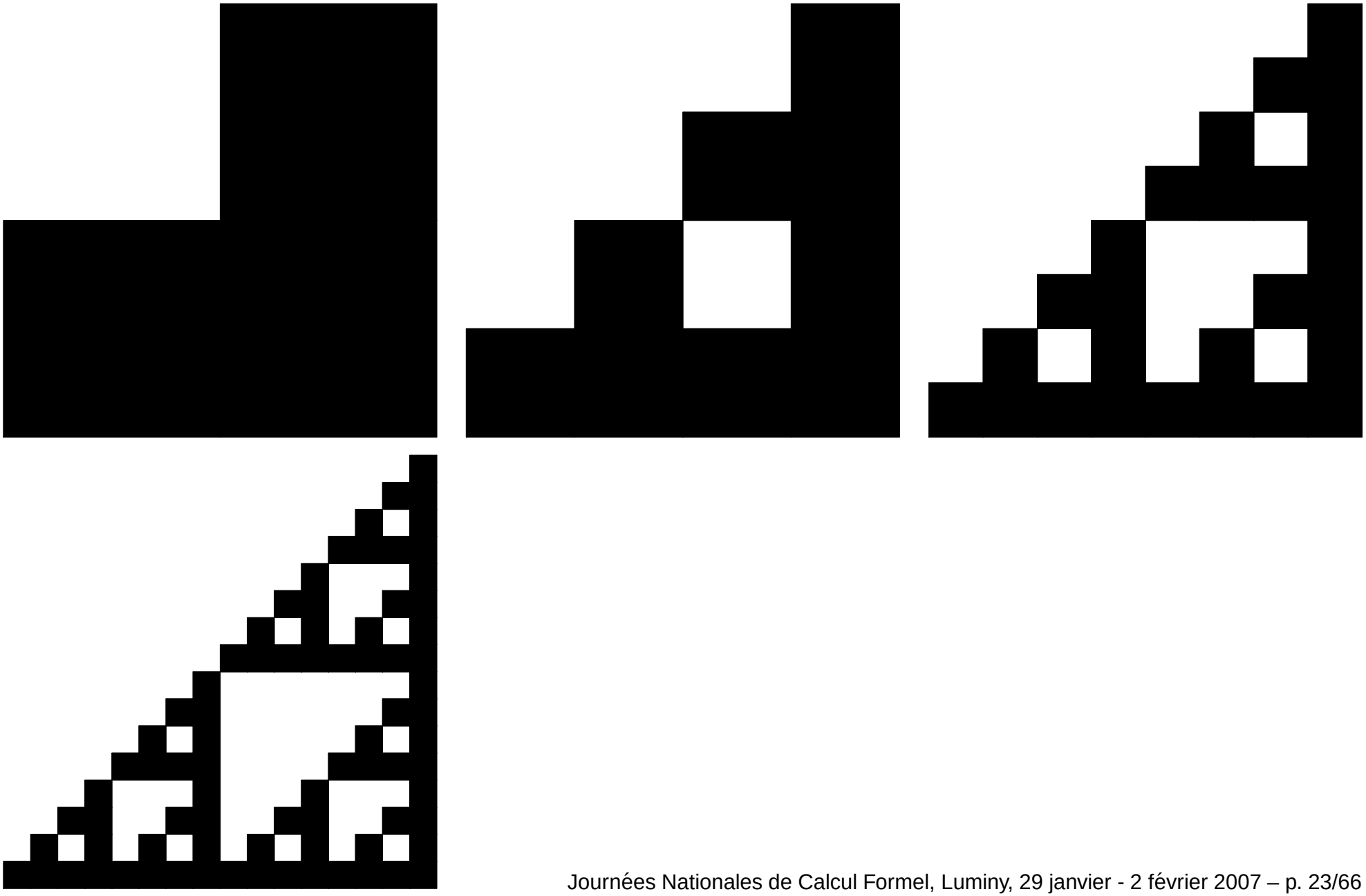


Multiplication de Karatsuba

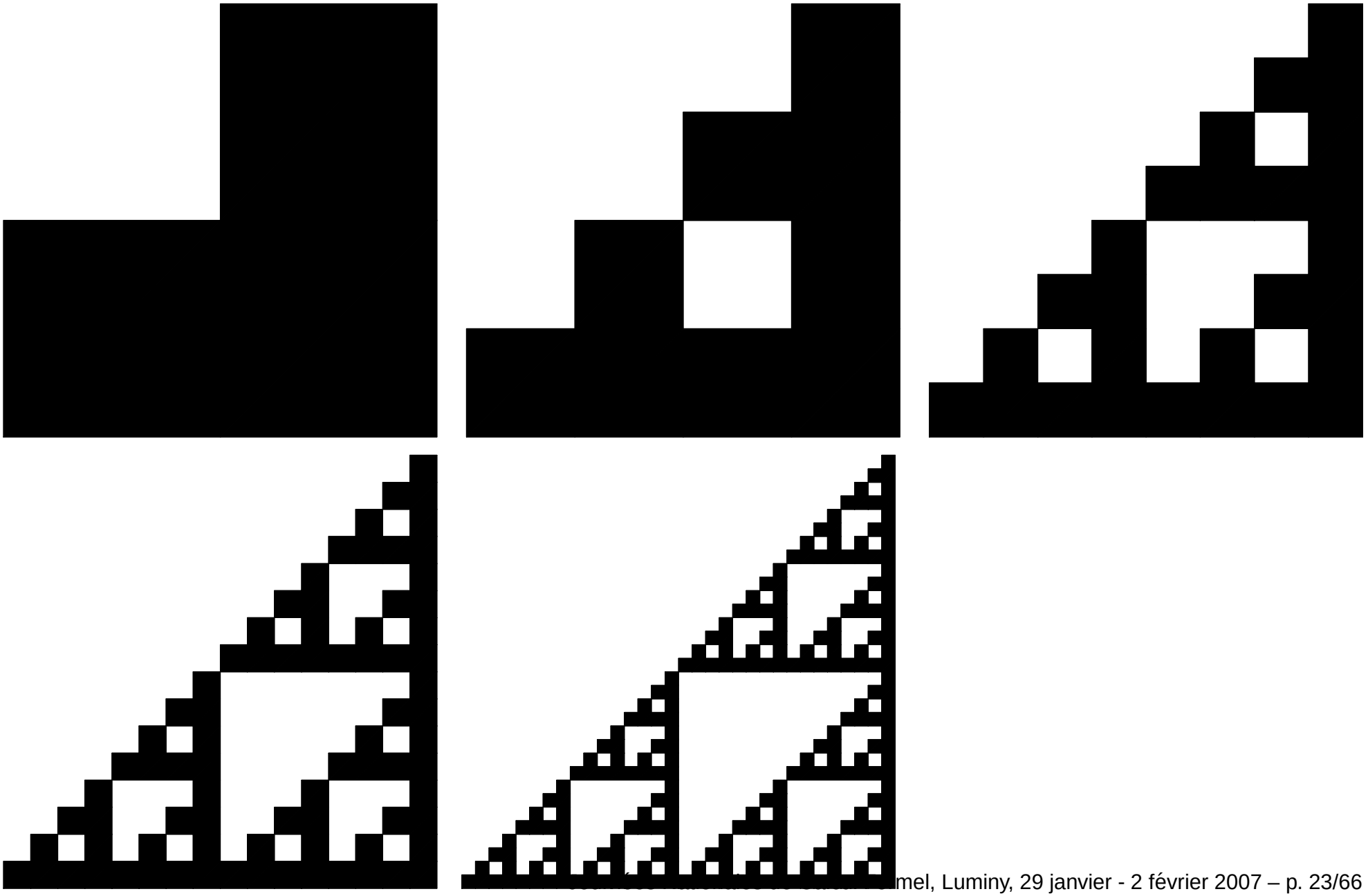
Multiplication de Karatsuba



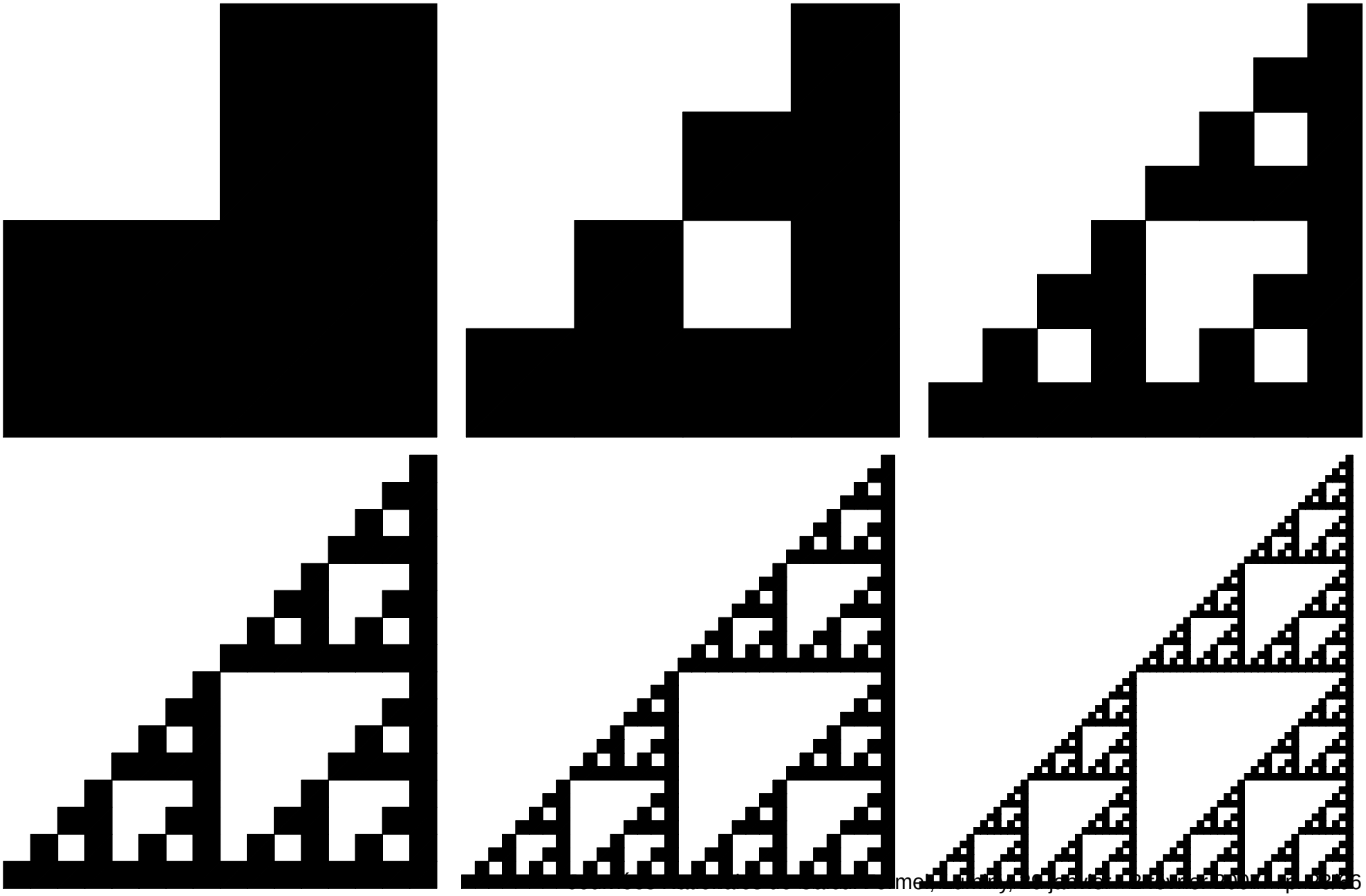
Multiplication de Karatsuba



Multiplication de Karatsuba



Multiplication de Karatsuba



Lecture rapide

Chaîne s en base $10 \longrightarrow$ nombre en base $\beta = 2^w$.

Ex : $s = 120705662053709250011994793957506052438 3$, $w = 32$.

$$L(1207056620537092500 1)10^{20} + L(19947939575060524383)$$

$$[L(1207056620)10^{10} + L(5370925001)]10^{20}$$

$$+[L(1994793957)10^{10} + L(5060524383)]$$

$L(n) = 2L(n/2) + M(n/2) \Longrightarrow L(n) = O(M(n) \log n)$

$\Longrightarrow$ arbre produit (*product tree*)

Écriture rapide

Nombre A en base $\beta = 2^w \rightarrow$ chaîne s en base 10.

$$A = 2350294565 \cdot \beta^3 + 1196140740 \cdot \beta^2 + 809094426 \cdot \beta + 2348838239$$

$$E(A \text{ div } 10^{20}) \quad \| \quad E(A \bmod 10^{20})$$

$E(n) = 2E(n/2) + D(n/2) \Longrightarrow E(n) = O(D(n) \log n)$

$\Longrightarrow$ arbre reste (*remainder tree*)

Écriture (encore plus) rapide

Scaled remainder trees, Bernstein, `cr.yp.to/papers.html#scaledmod`, 2004.

```
> A := 18620951977427710972982411727975589001 5:
> Digits:=45: B:=2^32:
> a := evalf(A/B^4);
     a := 0.54722059640998444949245942256735856233715313 1

> h, l := evalf(a,25), frac(a*B^2);
    0.5472205964099844494924594, 0.1883819761096689661557767

> round(h*B), round(frac(h*B)*B);
                    2350294565, 1196140740

> round(l*B), round(frac(l*B)*B);
                     809094426, 2348838239

> convert(A, base, B);
        [2348838239, 809094426, 1196140740, 2350294565]
```

Calcul de $n!$

Cf Schönhage, Grotefeld et Vetter, 1994.

```
> ifactor(102!);
    98      49      24      16       9       7       6       5       4
 (2)     (3)     (5)     (7)     (11)    (13)    (17)    (19)    (23)

          3       3       2       2       2       2
     (29)    (31)    (37)    (41)    (43)    (47)    (53)    (59)

     (61)   (67)   (71)   (73)   (79)   (83)   (89)   (97)   (101)
```

$$102! = a^2 \cdot 3 \cdot 11 \cdot 13 \cdot 19 \cdot 29 \cdot 31 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71 \cdot 73 \cdot 79 \cdot 83 \cdot 89 \cdot 97 \cdot 101$$

$$a = b^2 \cdot 2 \cdot 13 \cdot 17 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47$$

$$b = c^2 \cdot 13 \cdot 17 \cdot 19 \cdot 23$$

$$c = (((2^2) \cdot 2 \cdot 3)^2 \cdot 3 \cdot 5 \cdot 7)^2 \cdot 5 \cdot 11$$

Division et pgcd

Maple et GMP

http://www.loria.fr/~zimmerma/maple/gmp.txt

```
$ locate librootfindingmp.so
/usr/local/maple10/bin.IBM_INTEL_LINUX/librootfindingmp.so
$ cd /usr/local/maple10/bin.IBM_INTEL_LINUX
$ find . -name libgmp.so -print
./P4SSE2/libgmp.so
./PIIISSE1/libgmp.so
./ATHLON256/libgmp.so
./libgmp.so
./PIII/libgmp.so
./ATHLONXPSSE1/libgmp.so
./ATHLON512/libgmp.so
# cd P4SSE2
# mv libgmp.so.3.3.3 libgmp.so.3.3.3.orig
# ln -s /tmp/install/lib/libgmp.so.3.4.1 libgmp.so.3.3.3
```

Maple et GMP (suite)

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> 3^1000;
maple: fatal error, lost connection to kernel

    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> a:=3^2095903: b:=7^1183294: st:=time(): a*b: time()-st;
                              0.400

    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> a:=3^2095903: b:=7^1183294: st:=time(): a*b: time()-st;
                              0.216
```

Algorithmes de division

- division « naïve » : $O(n^2)$;
- division récursive (Burnikel, Ziegler, Borodin, Moenck, Jebelean) : $O(M(n)\log n)$;
- Newton (avec ou sans Barrett) : $O(M(n))$.

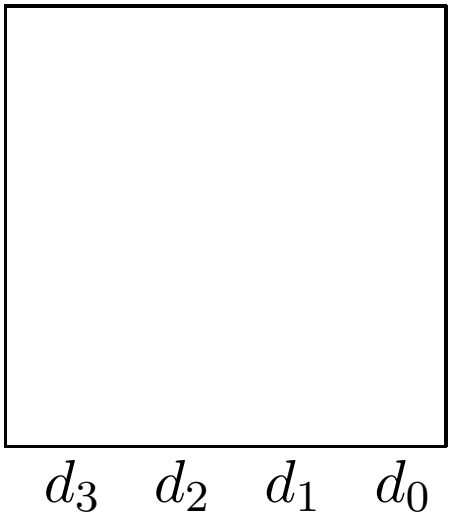
Note : le facteur $\log n$ devient une constante si $M(n) \approx n^\alpha$ avec $\alpha > 1$.

Division naï ve

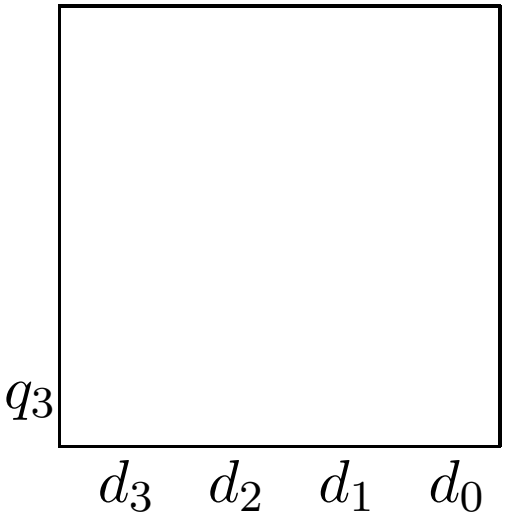
```
> A := 120705662053709250011994793957506052438 3:
> B := 57139535049024513573:
> bh := iquo(B, 10^19): # bh = 5
> Q:=0: R:=A; for j from 19 by -1 to 0 do
     h := iquo(R, 10^(19+j));
     q := iquo(h, bh);
     R := R - q*10^j*B;
     if R<0 then q:=q-1; R:=R+10^j*B fi;
     Q := 10*Q+q;
  od:
  Q, R;
          2112471897962528795 1, 16254514967227665460

> Q := iquo(A,B,'R'): Q, R;
          2112471897962528795 1, 16254514967227665460
```

Sur 20 itérations : 13 corrections.

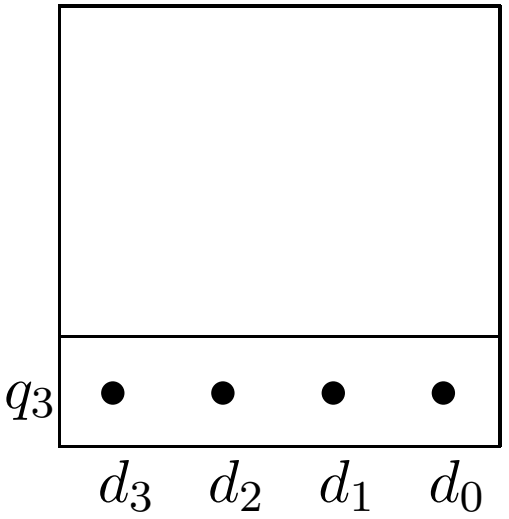
Division naï ve



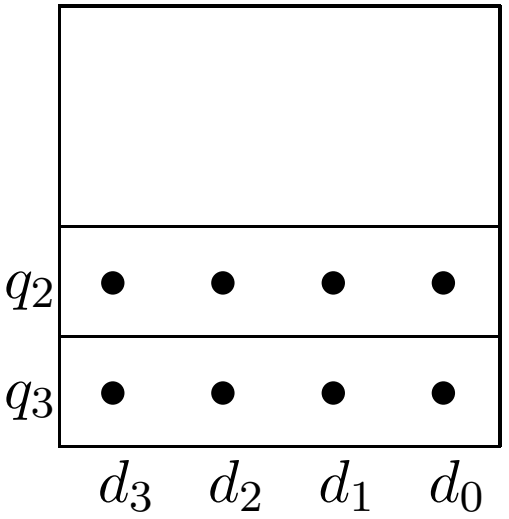
Division naï ve



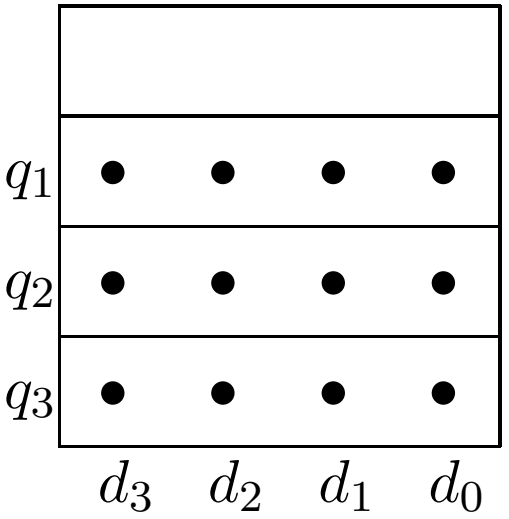
Division naï ve



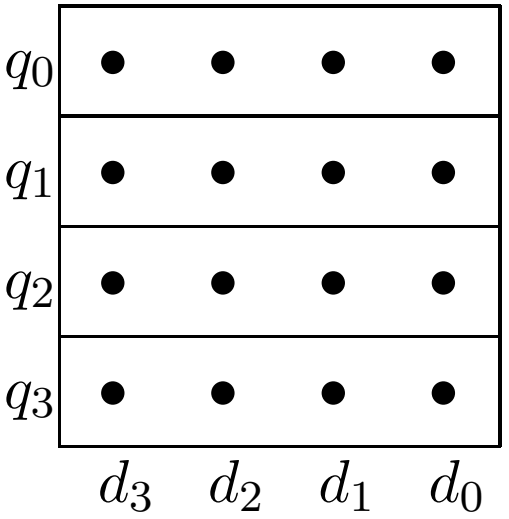
Division naï ve



Division naï ve



Division naï ve



Coût $O(n^2)$.

Algorithme de Svoboda (1963)

```
> A := 120705662053709250011994793957506052438 3:
> B := 57139535049024513573:
> k:=18: kB:=k*B; # bh = 10
                kB := 1028511630882441244314

> Q:=0: R:=A; for j from 18 by -1 to 0 do
     q := iquo(R, 10^(21+j));
     R := R - q*10^j*kB;
     if R<0 then q:=q-1; R:=R+10^j*kB fi;
     Q := 10*Q+q;
  od:
> q:=iquo(R,B): Q, R := k*Q+q, R-q*B;
      Q, R := 2112471897962528795 1, 16254514967227665460

> Q := iquo(A,B,'R'): Q, R;
          2112471897962528795 1, 16254514967227665460
```

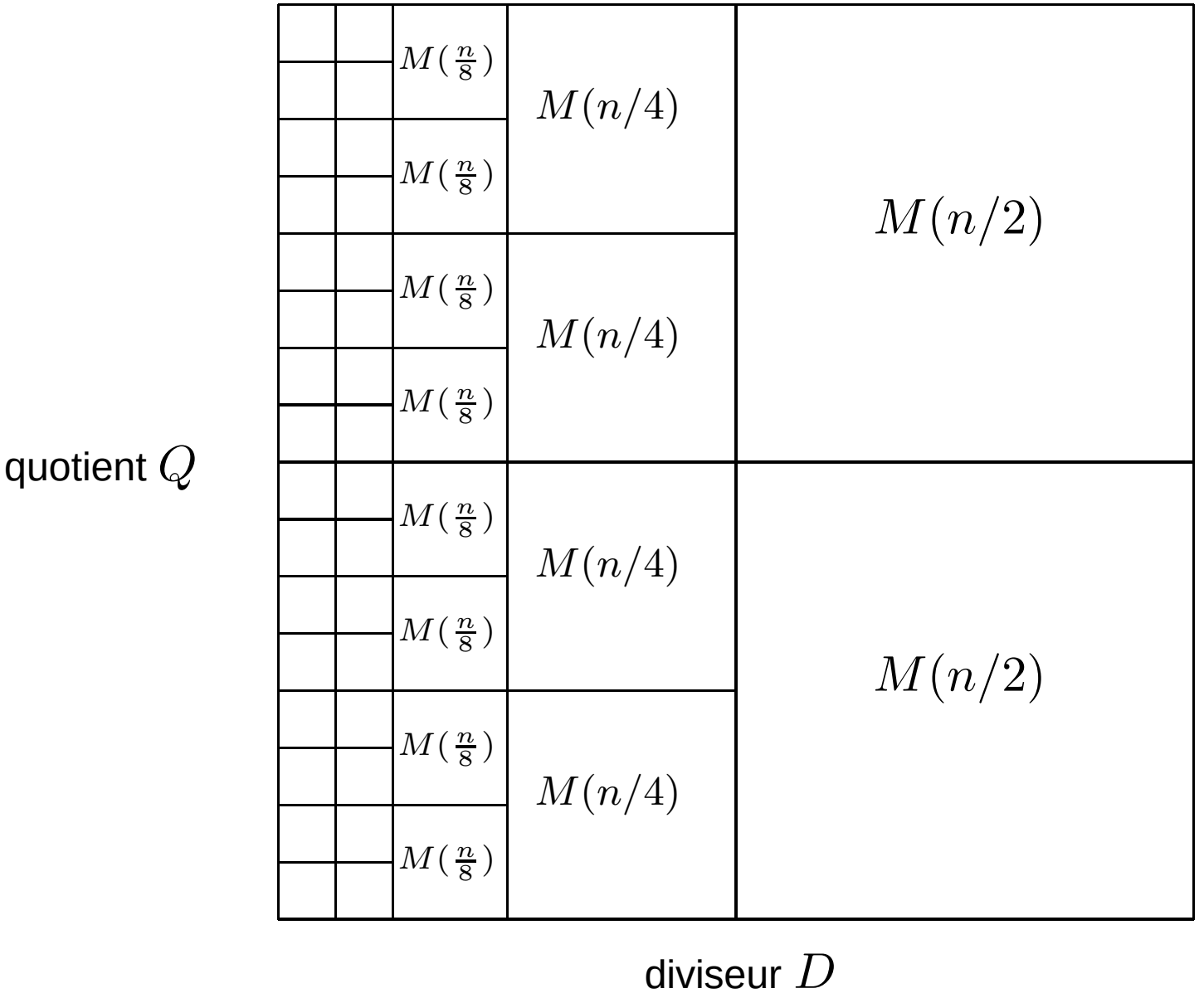
Sur 19 itérations : 5 corrections.

Préselection en double précision

```
> A := 120705662053709250011994793957506052438 3:
> B := 57139535049024513573:
> bh := iquo(B, 10^18): # bh = 57
> Q:=0: R:=A; for j from 19 by -1 to 0 do
     h := iquo(R, 10^(18+j));
     q := iquo(h, bh);
     R := R - q*10^j*B;
     if R<0 then q:=q-1; R:=R+10^j*B fi;
     Q := 10*Q+q;
  od:
  Q, R;
          2112471897962528795 1, 16254514967227665460
```

Sur 20 itérations : 1 seule correction. On peut aussi remplacer la division par b_h par une multiplication par une approximation de son inverse, et faire de même pour Svoboda.

Division « récursive »



$$D(n) = 2M(n/2) + 2D(n/2)$$

$$D(n) = 2K(n)$$

Algorithme de Barrett

```
> A := 120705662053709250011994793957506052438 3:
> B := 57139535049024513573:
> S := iquo(10^40, B);
                  S := 175010174503873917150
> Ah := iquo(A, 10^20);
                  Ah := 1207056620537092500 1
> Q := iquo(Ah * S, 10^20);
                   Q := 2112471897962528795 0
> R := A - Q*B;
                   R := 7339405001625217903 3
> Q, R := Q+1, R-B;
     Q, R := 2112471897962528795 1, 16254514967227665460
> Q := iquo(A,B,'R'): Q, R;
         2112471897962528795 1, 16254514967227665460
```

Un produit court haut ($A_h S$), un produit court bas (QB).

Algorithme de Barrett : complexité

Soit A de $2n$ bits à diviser par B de n bits.

1. $S \approx 1/B$ sur n bits : $I(n)$
2. $Q \approx A_h S$: $M(n)$
3. $R = A - QB$
4. corrections

Coût total $I(n) + 2M(n)$.

Si on a une FFT calculant un *produit cyclique* modulo $2^n - 1$, alors $\mathrm{FFT}(Q, B) = L - H \bmod (2^n - 1)$, où $H \approx A_h$. Coût $I(n) + 3/2M(n)$.

Algorithme de Barrett amorti

1. $S \approx 1/B$ sur n/k bits : $I(n/k)$
2. k fois : $Q \approx A_h S$: $M(n/k)$
3. k fois : $R = A - QB$: $M(n/k, n)$ ou $kM(n/k)$

Coût total $I(n/k) + kM(n/k) + k^2 M(n/k)$.

$k = 2 : I(n/2) + 6M(n/2)$

$I(n) \approx cM(n) : (c/2 + 3)M(n)$ vs $(c + 2)M(n)$.

Plus rapide pour $c \geq 2$.

Analogues poids forts/faibles

poids forts	poids faibles
produit court haut	produit court bas
division classique	???

Division de Hensel (2-adique)

Soit A de $2n$ chiffres, et B de n chiffres, premier avec β :

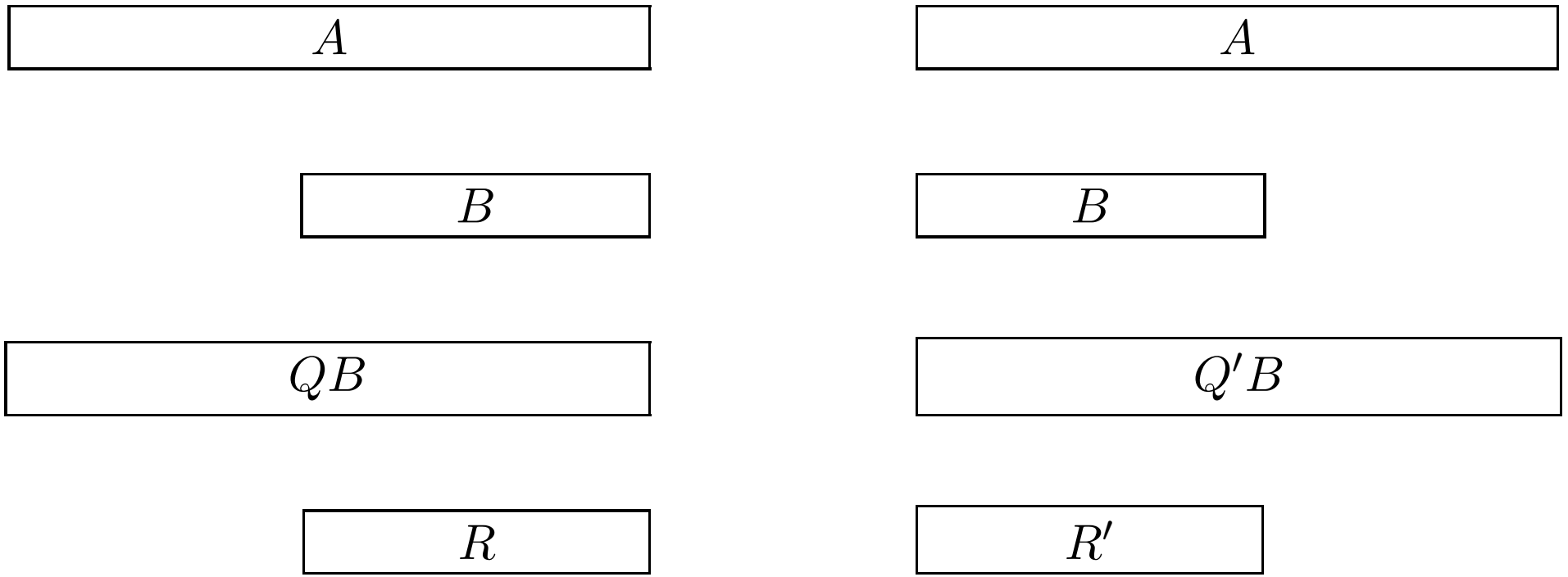
$$A = QB + \beta^n R.$$

Définie de façon unique par $Q = A/B \bmod \beta^n$:

```
> A := 120705662053709250011994793957506052438 3:
> B := 571395350490245135 73:
> Q := A/B mod 10^20;
                    Q := 25534189992557265971
> A - Q*B;
          -251955123491084382200000000000000000000
> R := %/10^20;
                    R := -2519551234910843822
```

Attention : R peut être négatif !

Division de Hensel (2)



Analogues poids forts/faibles

poids forts	poids faibles
produit court haut	produit court bas
division classique	division de Hensel
multiplication modulaire	???

Multiplication de Montgomery

Définition : soit N de n mots en base β, premier avec β, et $0 \leq A, B < n$:

$$\mathrm{MontMul}(A, B) := AB\beta^{-n} \bmod N$$

Kesako ?

Multiplication de Montgomery

Définition : soit N de n mots en base β, premier avec β, et $0 \leq A, B < n$:

$$\mathrm{MontMul}(A, B) := AB\beta^{-n} \bmod N$$

Kesako ? Rien d'autre que le reste de la division de Hensel de $C = AB$ par N !

$$C = QN + \beta^n R$$

Multiplication de Montgomery (2)

Supposons qu'on a une suite de calculs modulaires (additions, multiplications) modulo le même entier N (ex : factorisation).

1. on convertit toutes les données en représentation de Montgomery :

$$A \to A' = A\beta^n \bmod N$$

2. on effectue tous les calculs normalement : additions inchangées, multiplications $\longrightarrow$ multiplications de Montgomery :

$$AB \to A'B'\beta^{-n} \bmod N (= AB\beta^n)$$

3. on revient à la représentation classique :

$$A' \to A = A'\beta^{-n} \bmod N$$

Multiplication de Montgomery (3)

Avantage : MontMul plus facile à calculer que la multiplication classique (pas de retenue).

Inconvénient : surcoût des étapes 1 et 3.

Application idéale : exponentiation modulaire (1 et 3 font $O(1)$ multiplications, 2 fait $O(n)$ multiplications)

Deux versions :

- « naïve » en $O(n^2)$: réduit un mot à la fois ;
- sous-quadratique : réduit n mots à la fois.

MontMul : version naï ve

```
> A := 120705662053709250011994793957506052438 3:
> B := 57139535049024513573:
> s := -1/B mod 10;
                                s := 3

> R:=A; to 20 do
    c := ((R mod 10) * s) mod 10;
    R := (R + c*B)/10;
  od:
> R;
                     546199838141136697 51

> R-B, (A - (A/B mod 10^20)*B)/10^20;
          -2519551234910843822, -2519551234910843822
```

MontMul : version sous-quadratique

```
> A := 120705662053709250011994793957506052438 3:
> B := 571395350490245135 73:
> S := -1/B mod 10^20;
                    S := 56089563221556001363

> Al := A mod 10^20;
                   Al := 19947939575060524383

> C := Al * S mod 10^20;
                    C := 74465810007442734029

> R := (A + C*B)/10^20;
                    R := 54619983814113669751

> R-B, (A - (A/B mod 10^20)*B)/10^20;
          -251955123491084382 2, -251955123491084382 2
```

Un produit court bas ($A_l S$), un produit court haut (CB).

Division exacte

Réduction de fraction rationnelle, pgcd étendu, . . .

```
> A := 120705662053709250010369342460783285892 3:
> B := 571395350490245135 73:
> Digits:=25: evalf(iquo(A,10^20)/B);
                0.21124718979625287950935 36

> A/B mod 10^20;
                  21124718979625287951

> Digits:=15:evalf(iquo(A,10^25)/iquo(B,10^5));
                    0.211247189796252

> A/B mod 10^10;
                        9625287951
```

Analogues poids forts/faibles

poids forts	poids faibles
produit court haut	produit court bas
division classique	division de Hensel
multiplication modulaire	multiplication de Montgomery
algorithme de Barrett	algorithme REDC sous-quadratique

Division détendue

Multiplication d'entiers de n mots, ou polynômes de degré $< n$

Division détendue

Multiplication d'entiers de n mots, ou polynômes de degré $< n$

Algorithme naïf : n^2 opérations

$$(ax+b)(cx+d) = (ac)x^2 + (ad+bc)x + (bd)$$

d	ad	bd
c	ac	bc
	a	b

Division détendue

Multiplication d'entiers de n mots, ou polynômes de degré $< n$

Algorithme naïf : n^2 opérations

$$(ax+b)(cx+d) = (ac)x^2 + (ad+bc)x + (bd)$$

d	ad	bd
c	ac	bc
	a	b

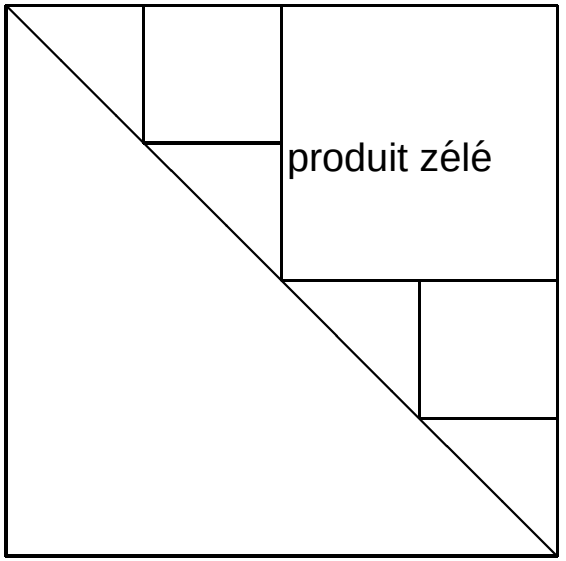
Karatsuba :

$$(ax+b)(cx+d) = (ac)x^2 + [(a+b)(c+d) - ac - db]x + (bd)$$

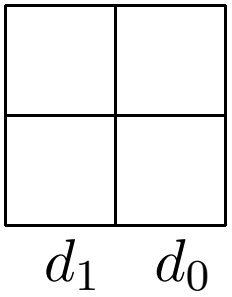
d	•	bd
c	ac	•
	a	b

Produit détendu

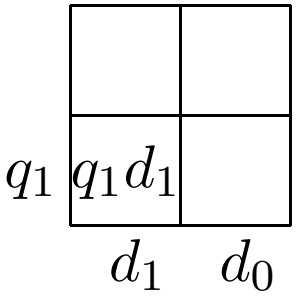
J. van der Hoeven, *Relax, but don't be too lazy*, Journal of Symbolic Computation, 2002.



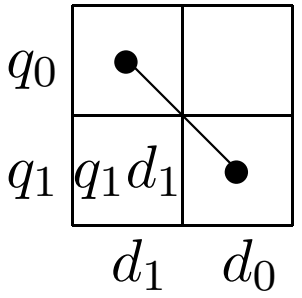
Division détendue



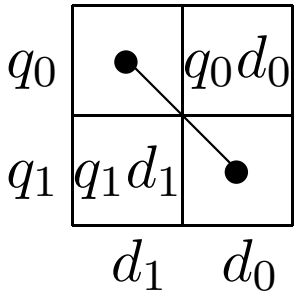
Division détendue



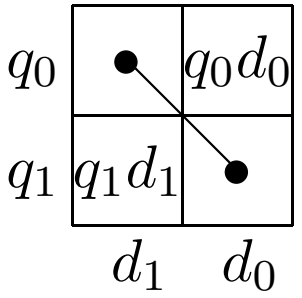
Division détendue



Division détendue

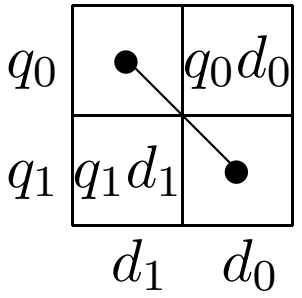


Division détendue



1. Calcul récursif de q_1 et $q_1 d_1$

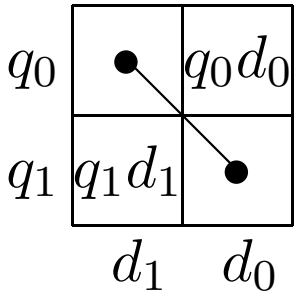
Division détendue



1. Calcul récursif de q_1 et $q_1 d_1$

2. Calcul récursif de q_0 et $(q_1 + q_0)(d_1 + d_0) - q_1 d_1 - q_0 d_0$

Division détendue

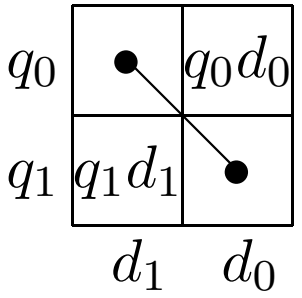


1. Calcul récursif de q_1 et $q_1 d_1$

2. Calcul récursif de q_0 et $(q_1 + q_0)(d_1 + d_0) - q_1 d_1 - q_0 d_0$

Coût $K(n)$ (idem multiplication de Karatsuba).

Division détendue



1. Calcul récursif de q_1 et $q_1 d_1$

2. Calcul récursif de q_0 et $(q_1 + q_0)(d_1 + d_0) - q_1 d_1 - q_0 d_0$

Coût $K(n)$ (idem multiplication de Karatsuba).

Difficultés :

- optimiser la mémoire (sous-produits intermédiaires)
- dé-récursification
- gérer les retenues (cas entier)

Reconstruction rationnelle

Problème. Étant donnés c et n, résoudre en a et b :

$$\frac{a}{b} = c \bmod n.$$

On cherche a et b « petits », i.e., de taille moitié de celle de n.

On en déduit qu'il existe λ tel que :

$$a = bc + \lambda n.$$

« Demi-pgcd » entre n et c

$$\begin{aligned} n &= 1 \cdot n + 0 \cdot c \\ c &= 0 \cdot n + 1 \cdot c \\ \cdots &= \ldots + \cdots \\ r_i &= \lambda_i n + \mu_i c \\ \cdots &= \ldots + \cdots \\ g &= \lambda_k n + \mu_k c \end{aligned}$$

Les r_i diminuent, alors que les λ_i, μ_i grandissent.

Demi-pgcd (étendu) : on arrête quand $\mathrm{size}(r_i) \approx \mathrm{size}(\lambda_i) \approx \mathrm{size}(n)/2$.

« Demi-pgcd » (suite)

Quand $\mathrm{size}(r_i) \approx \mathrm{size}(\lambda_i) \approx \mathrm{size}(n)/2$:

$$r = \lambda n + \mu c,$$

soit :

$$\frac{r}{\mu} = c \bmod n.$$

Sous certaines conditions, $\frac{r}{\mu}$ est la fraction cherchée $\frac{a}{b}$ (reconstruction rationnelle).

Avantages :

- tous les calculs avec des entiers (modulo n assez grand) ;
- maîtrise de la croissance des coefficients.

« Demi-pgcd » (fi n)

$$\begin{aligned} r_i &= \lambda_i n + \mu_i c \\ \cdots &= \ldots + \cdots \\ g &= \lambda_k n + \mu_k c \end{aligned}$$

$$\lambda_{i+1} = -q_i \lambda_i + \lambda_{i-1}$$

$$\mu_{i+1} = -q_i \mu_i + \mu_{i-1}$$

On peut ne calculer que les μ_i, et retrouver λ_k par division exacte :

$$\lambda_k = \frac{g - \mu_k c}{n}.$$

Inversions multiples

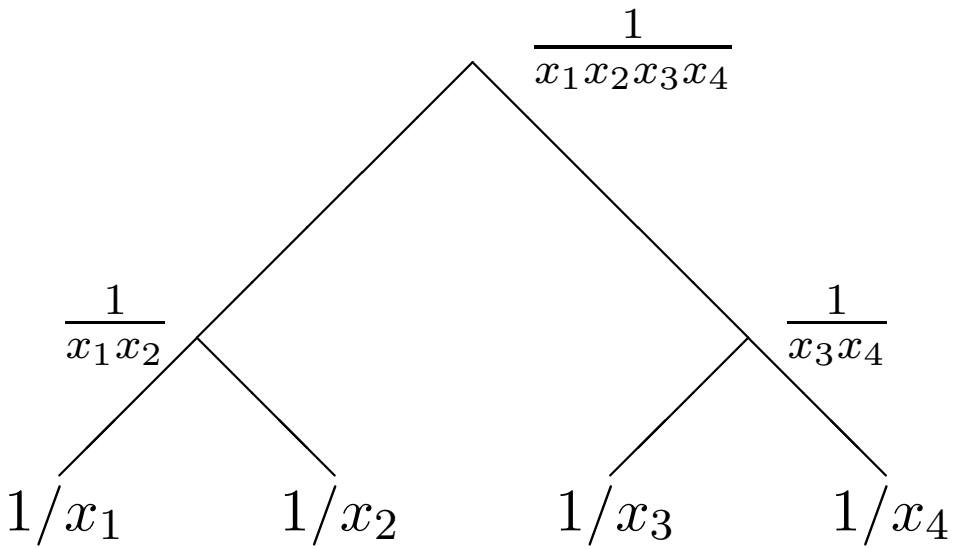
Problème : calculer $1/x \bmod n$ et $1/y \bmod n$.

Algo naïf : 2 pgcds étendus.

Astuce de Montgomery : $z = xy \bmod n$, $t = 1/z \bmod n$, $ty = 1/x \bmod n$, $tx = 1/y \bmod n$.

Remplace une inversion par trois produits modulaires.

Inversions multiples (2)



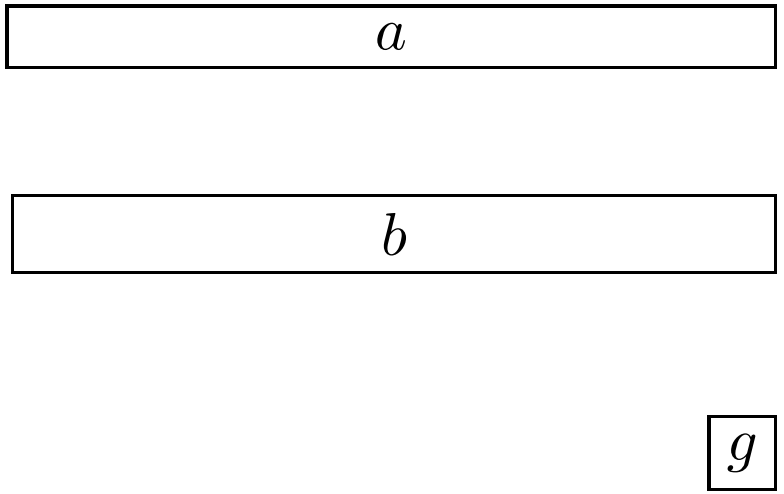
Pour k variables :

- $k-1$ produits pour « remonter » dans l'arbre (calculer $x_1x_2\cdots x_k$) ;
- $2k-2$ produits pour « descendre » dans l'arbre.

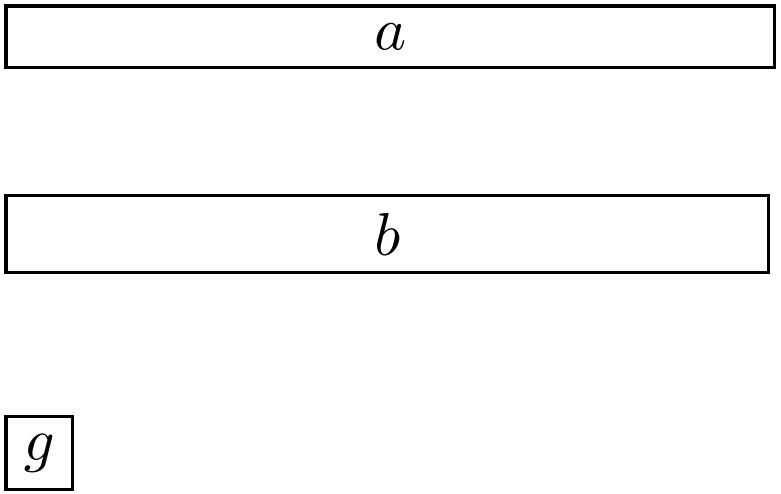
Analogues poids forts/faibles

poids forts	poids faibles
produit court haut	produit court bas
division classique	division de Hensel
multiplication modulaire	multiplication de Montgomery
pgcd classique	???

Pgcd 2-adique



pgcd classique



pgcd 2-adique

Pgcd 2-adique (2)

Soit $a = 7419669081$, $b = 2974121768$.

Pgcd classique :

$$-782730695a + 19527118 22b = 1$$

Division binaire généralisée : soit $\nu_2(a) < \nu_2(b)$,

$$r = a + q\frac{b}{2^{\nu_2(b)-\nu_2(a)}}$$

$$q = -\frac{a}{2^{\nu_2(a)}}\frac{2^{\nu_2(b)}}{b} \bmod 2^{\nu_2(b)-\nu_2(a)+1}$$

$$|q| < 2^{\nu_2(b)-\nu_2 a}$$

$$\nu_2(r) > \nu_2(b)$$

Pgcd 2-adique (3)

Exemple avec $a = 7419669081$, $b = 2974121768$:

$$r_0 = a = (1101110100011111100101010011001)_2$$

$$r_1 = b = (1011000101000101011111100101000)_2$$

$$r_2 = a - 5b/2^3 = (1010010110111001110111010111100000)_2$$

$$r_3 = b - 3r_2/2^2 = (-1000111010100010100110100000000)_2$$

$$r_4 = r_2 + 3r_3/2^3 = (1001100001011010100111111000000000)_2$$

$$r_5 = r_3 - r_4/2^1 = (-1101111110101011111011000000000000)_2$$

$$r_6 = r_4 + r_5/2^1 = (1100000011011111010010000000000000)_2$$

$$\dots$$

$$r_{12} = r_{10} + 7r_{11}/2^4 = (1000000000000000000000000)_2$$

Ce qu'il fallait retenir

$$n \log n \log\log n \ll M(n) \ll n^2$$

- Addition, soustraction : $O(n)$;
- Multiplication, division, racine carrée : $O(M(n))$;
- pgcd, conversion : $O(M(n) \log n)$;
- exponentiation modulaire (RSA) : $O(nM(n))$.