Arithmétique Algorithmique

http://www.math.univ-lyon1.fr/~roblot/ens.html

Partie II

Arithmétique rapide

Opérations de base sur les entiers longs

Entiers longs. Soit $w \geq 1$, on encode l'entier N en utilisant des nombres de w bits (nombres entiers entre 0 et $2^w - 1$). On écrit

$$N = (-1)^s \sum_{i=0}^{l} a_i 2^{wi} \quad \text{avec } s \in \{0,1\} \text{ et } a_i \in \{0, \ldots, 2^w - 1\}.$$

On représente N par $(s \cdot 2^{w-1} + l + 1, a_0, \ldots, a_l)$.
Cette représentation est unique si $a_l \neq 0$ et de taille $O_w(\log(N))$.

Exemple. -1 est representé par $(2^{w-1} + 1, 1)$.

Restriction. On doit avoir $-2^{w \cdot 2^{w-1}-1} + 1 \leq N \leq 2^{w \cdot 2^{w-1}-1} - 1$.

Taille maximale pour 32 bits : 68 719 476 735 bits $\approx$ 8 GB

Taille maximale pour 64 bits : $5,9 \cdot 10^{20}$ bits $\approx 6,8 \cdot 10^{10}$ GB

Opérations de base sur les entiers longs

Addition de deux entiers de même signe (et de même longueur).

$$N = (-1)^s \sum_{i=0}^{l} a_i 2^{wi} \quad \text{et} \quad M = (-1)^s \sum_{i=0}^{l} b_i 2^{wi}$$

Algorithme.

(1) Faire $t_0 \leftarrow 0$

(2) Pour $i = 0$ à l, faire
$c_i \leftarrow a_i + b_i + t_i$, $t_{i+1} \leftarrow 0$
Si $c_i \geq 2^w$, alors faire $c_i \leftarrow c_i - 2^w$, $t_{i+1} \leftarrow 1$

(3) Si $t_{l+1} = 1$ alors poser $c_{l+1} \leftarrow 1$ et retourner $(s + l + 2, c_0, \ldots, c_{l+1})$, sinon retourner $(s + l + 1, c_0, \ldots, c_l)$.

Complexité. $O(l) = O(\log N)$ opérations sur des mots de w bits.
De ce point de vue, l'algorithme est optimal.

Opérations de base sur les entiers longs

Addition de deux entiers de signes différents (et de même longueur).

$$N = \sum_{i=0}^{l} a_i 2^{wi} \quad \text{et} \quad M = -\sum_{i=0}^{l} b_i 2^{wi} \quad \textbf{avec} \quad N + M \geq 0$$

Algorithme.

(1) Faire $t_0 \leftarrow 0$

(2) Pour $i = 0$ à l, faire

Si $a_i \geq b_i + t_i$, alors faire

$c_i \leftarrow a_i - b_i - t_i$, $t_{i+1} \leftarrow 0$,

sinon faire

$c_i \leftarrow 2^w + a_i - b_i - t_i$ et $t_{i+1} \leftarrow 1$

(3) Retourner $(l + 1, c_0, \ldots, c_l)$. [non réduit]

Remarques. Complexité $O(l)$ optimale aussi.
Tester si $N + M \geq 0$ se fait en temps $O(l)$ aussi.
Possibilité de faire un algorithme sans l'hypothèse $N + M \geq 0$.

Opérations de base sur les entiers longs

Multiplication de deux entiers (de même longueur).

$$N = (-1)^s \sum_{i=0}^{l} a_i 2^{wi} \quad \text{et} \quad M = (-1)^t \sum_{i=0}^{l} b_i 2^{wi}$$

Algorithme.

(1) Pour $i = 0$ à l, faire $T_i \leftarrow (a_i \cdot |M|) \cdot 2^{wi}$,

(2) Retourner $(-1)^{s+t} \sum_{i=0}^{l} T_i$.

Remarques. La multiplication de l'entier long $|M|$ par un entier de w bits se fait en $O(l)$ opérations.
La multiplication par 2^{wi} est juste un décalage.
La somme finale se fait avec l'algorithme précédent.

Complexité. $O(l^2)$ opérations sur des mots de w bits.
Complexité non optimale !

Polynômes à coefficients dans $\mathbb{Z}/2^w\mathbb{Z}$

Représentation. Le polynôme

$$A(X) = a_l X^l + \cdots + a_0 \in \mathbb{Z}/2^w\mathbb{Z}[X]$$

est représenté par $(l+1, a_0, \ldots, a_l)$ avec $l \leq 2^w - 1$.

Polynômes et entiers longs.

$$(-1)^s \sum_{i=0}^{l} a_i 2^{wi} \quad \leftrightarrow \quad \left(s, \sum_{i=0}^{l} a_i X^i \right)$$

Ce n'est pas un "morphisme" car il n'y a pas de retenue dans les calculs sur les polynômes de $\mathbb{Z}/2^w\mathbb{Z}[X]$.

Avantages. Les algorithmes sur les polynômes de $\mathbb{Z}/2^w\mathbb{Z}[X]$ sont plus faciles à énoncer que les algorithmes sur les entiers longs. On peut transformer un tel algorithme en un algorithme sur les entiers longs de même complexité.

Polynômes à coefficients dans $\mathbb{Z}/2^w\mathbb{Z}$

Division euclidienne.

$A(X) = a_l X^l + \cdots + a_0$ par $B(X) = b_m X^m + \cdots + b_0$ avec $b_m = 1$ et $l \geq m$.

Algorithme.

(1) Poser $R \leftarrow A$

(2) Pour $i = l - m$ à 0 (décroissant), faire
  Si $\deg R = m + i$, alors
    poser $q_i \leftarrow \mathrm{CD}(R)$, $R \leftarrow R - q_i \cdot B \cdot X^i$,
  sinon faire $q_i \leftarrow 0$

(3) Retourner $Q(X) = \sum_{i=0}^{l-m} q_i X^i$ et R

Remarques. $\mathrm{CD}(R)$ renvoie le coefficient dominant de R.
La division n'est pas possible si b_m n'est pas inversible (cf. "pseudo-division").
Complexité en $O(l^2)$.
Cas des entiers longs plus complexes à cause des quotients partiels (cf. 90 divisé par 19)

Multiplication rapide par la méthode de Karatsuba

Degré 1. Le calcul

$$(aX+b)(cX+d) = acX^2 + (ad+bc)X + bd$$

nécessite 4 multiplications et 1 addition.

Autre méthode, on calcule ac, bd, $u = (a+b)(c+d)$ et on a

$$ad + bc = u - ac - bd$$

Coût : 3 multiplications et 4 additions.

Diviser pour régner. Pour deux polynômes $A(X)$ et $B(X)$ de degré $2n$, on pose

$$A(X) = A_1(X)\cdot X^n + A_0(X) \quad \text{et} \quad B(X) = B_1(X)\cdot X^n + B_0(X)$$

avec $\deg A_i, \deg B_i \leq n$ et on calcule

$$A\cdot B = A_1B_1\cdot X^{2n} + \big((A_0+A_1)(B_0+B_1) - A_0B_0 - A_1B_1\big)\cdot X^n + A_0B_0$$

Pour le calcul de A_1B_1, A_0B_0 et $(A_0+A_1)(B_0+B_1)$, on utilise la même méthode de manière récursive.

Multiplication rapide par la méthode de Karatsuba

Complexité d'un algorithme "diviser pour régner"

Soient T et f deux fonctions de $\mathbb{N}$ dans $\mathbb{N}$, vérifiant la relation

$$T(n) = a\,T(n/b) + f(n)\,,$$

avec $a \geq 1$ et $b > 1$.
Alors, on a

$$T(n) \in \begin{cases} O(n^{\log_b(a)}) & \text{si } f(n) \in O(n^{\log_b(a)-\varepsilon}) \text{ pour un } \varepsilon > 0 \\ O(n^{\log_b(a)}\log(n)) & \text{si } f(n) \in O(n^{\log_b(a)}) \\ O(f(n)) & \text{si } f(n) \in O(n^{\log_b(a)+\varepsilon}) \text{ pour un } \varepsilon > 0 \\ & \text{et } af(n/b) \leq cf(n) \text{ pour } n \gg 0 \text{ et } c < 1 \end{cases}$$

Multiplication rapide par la méthode de Karatsuba

Diviser pour régner. Pour deux polynômes $A(X)$ et $B(X)$ de degré $2n$, on pose $A(X) = A_1(X) \cdot X^n + A_0(X)$ et $B(X) = B_1(X) \cdot X^n + B_0(X)$ avec $\deg A_i, \deg B_i \leq n$ et on calcule

$$A \cdot B = A_1 B_1 \cdot X^{2n} + \big((A_0 + A_1)(B_0 + B_1) - A_0 B_0 - A_1 B_1\big) \cdot X^n + A_0 B_0$$

Donc on a

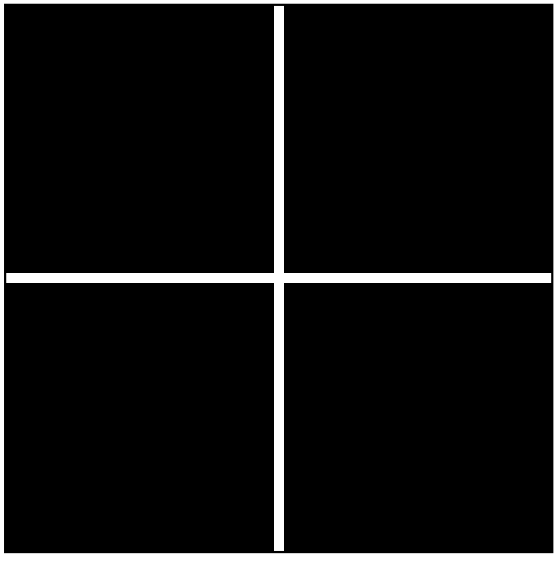
$$T(2n) = 3T(n) + 8n$$

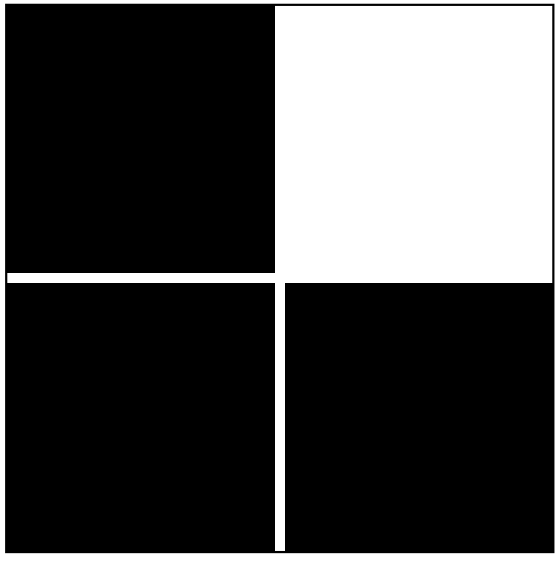
Complexité de l'algorithme de Karatsuba

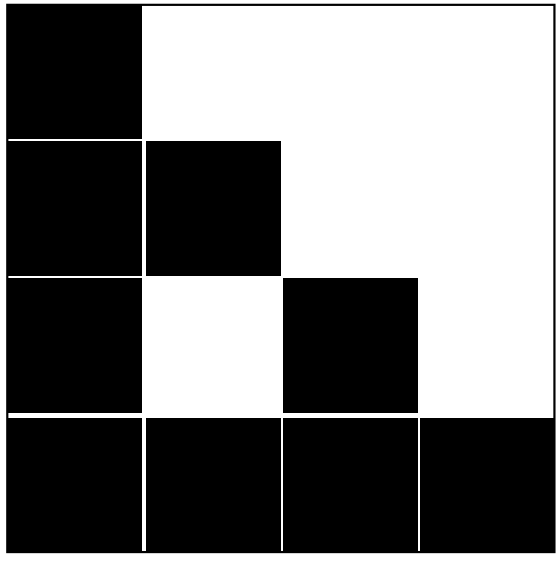
L'algorithme de Karatsuba calcule le produit de deux polynômes dans $\mathcal{R}[X]$ et de degré $\leq n$ en $O(n^{\log_2 3})$ multiplications dans $\mathcal{R}$.
L'algorithme de Karatsuba calcule le produit de deux entiers de taille 2^{wn} en $O(n^{\log_2 3})$ multiplications entre des entiers de w bits.

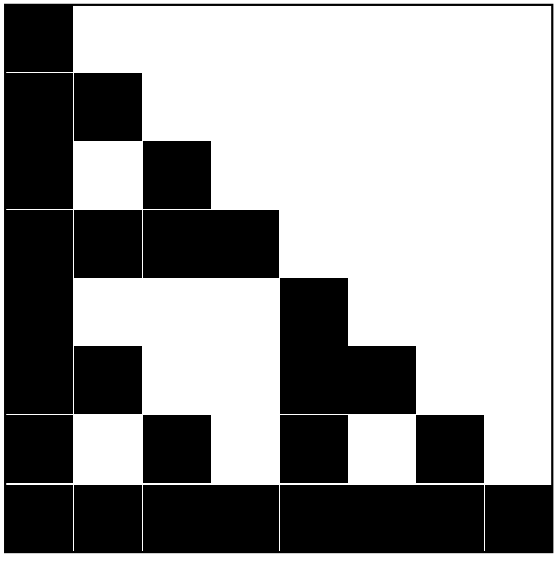
Remarque. $\log_2 3 \approx 1.585$

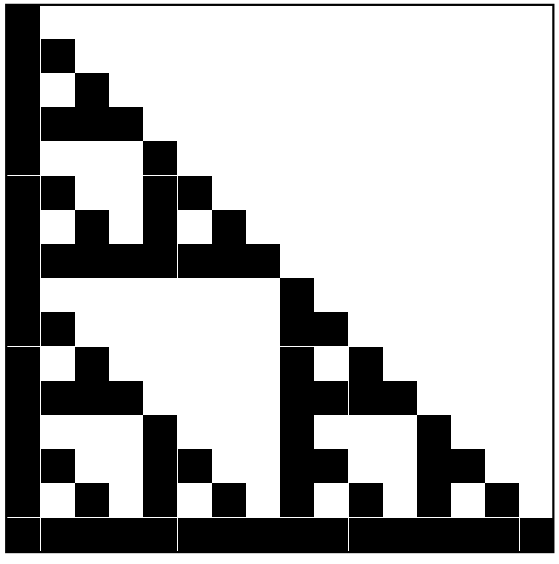
Multiplication rapide par la méthode de Karatsuba











Tend vers
une fractale
de dimension $\log_2 3$

Multiplication rapide par FFT : idée de départ

Multiplication pour interpolation. On cherche à multiplier deux polynômes A et B de degré $\leq n$.

Le résultat $C = A \cdot B$ est un polynôme de degré $\leq 2n$ et donc complétement déterminé par $2n + 1$ valeurs.

Les valeurs de C sont faciles à calculer : $C(x) = A(x) \cdot B(x)$.

Problèmes. Il faut quand même calculer beaucoup de valeurs de C, et, surtout reconstruire un polynôme à partir de ses valeurs ce qui peut être un problème coûteux.

Solution. On utilise les racines de l'unité et la transformée de Fourier discrète !

Remarque. FFT = Fast Fourier Transform

Multiplication rapide par FFT : transformée de Fourier discrète

Racine de l'unité. On suppose que l'on travaille dans un anneau $\mathcal{R}$ (unitaire, commutatif) qui contient ω une racine primitive de l'unité d'ordre $n \geq 2$, c'est-à-dire $\omega^n = 1$ et $\omega^m \neq 1$ pour $1 \leq m < n$.

DFT. Pour $A \in \mathcal{R}[X]$, on pose

$$\mathrm{DFT}_\omega(A) = \big(A(1), A(\omega), \ldots, A(\omega^{n-1})\big) \in \mathcal{R}^n$$

C'est un morphisme d'anneau (où l'addition et la multiplication dans $\mathcal{R}^n$ se font composante par composante) de noyau l'idéal engendré par $X^n - 1$.

On note $\mathcal{R}_n[X]$ l'ensemble des polynômes de $\mathcal{R}[X]$ et de degré $< n$. Alors la restriction de DFT à $\mathcal{R}_n[X]$ est injective.

Multiplication rapide par FFT : transformée de Fourier discrète

Convolution. Pour $A, B \in \mathcal{R}_n[X]$, on pose

$$A * B = A \cdot B \mod X^n - 1 \in \mathcal{R}_n[X]$$

Proposition

L'ensemble $\mathcal{R}_n[X]$ avec l'addition classique et la convolution pour multiplication est un anneau isomorphe (par DFT) à $\mathcal{R}^n$.

Explicitement, pour $A, B \in \mathcal{R}_n[X]$, on a

$$\begin{aligned} \mathsf{DFT}_\omega(A * B) &= \mathsf{DFT}_\omega(A) \cdot \mathsf{DFT}_\omega(B) \\ \mathsf{DFT}_\omega(A + B) &= \mathsf{DFT}_\omega(A) + \mathsf{DFT}_\omega(B) \end{aligned}$$

Remarque. $(\mathcal{R}_n[X], +, *) \cong \mathcal{R}[X]/(X^n - 1)$

Multiplication rapide par FFT : matrices de Vandermonde

Définition. Pour $\alpha \in \mathcal{R}$, on définit

$$V_\alpha = (\alpha^{(i-1)(j-1)})_{1\leq i,j\leq n} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \alpha^{n-1} & \alpha^{2(n-1)} & \cdots & \alpha^{(n-1)^2} \end{pmatrix}$$

Calcul de DFT et DFT^{-1}

Soit $A \in \mathcal{R}_n[X]$, on écrit $A = a_{n-1}X^{n-1} + \cdots + a_0$. On a alors

$$\mathrm{DFT}_\omega(A) = (a_0, a_1, \ldots, a_{n-1}) \cdot V_\omega$$

Pour $(v_0, \ldots, v_{n-1}) \in \mathcal{R}^n$, on pose

$$(a_0, a_1, \ldots, a_{n-1}) = (v_0, \ldots, v_{n-1}) \cdot V_\omega^{-1}$$

Alors $\mathrm{DFT}_\omega(a_{n-1}X^{n-1} + \cdots + a_0) = (v_0, \ldots, v_{n-1})$.

Multiplication rapide par FFT : matrices de Vandermonde

Théorème

$$V_\omega \cdot V_{\omega^{-1}} = n \cdot \mathrm{Id}_n$$

Preuve. On a $V_\alpha = \left(\alpha^{(i-1)(j-1)}\right)_{i,j=1}^{n}$ et donc l'entrée i, j de la matrice $V_\omega \cdot V_{\omega^{-1}}$ est

$$\sum_{k=1}^{n} \omega^{(i-1)(k-1)} \omega^{-(k-1)(j-1)} = \sum_{k=1}^{n} \omega^{(k-1)(i-j)}$$

Mais, si $n \nmid i - j$ (i.e $i \neq j$) alors ω^{i-j} est une racine primitive de l'unité $\neq 1$ et d'ordre divisant n et donc

$$\sum_{k=1}^{n} (\omega^{i-j})^k = \begin{cases} n & \text{si } i = j, \\ 0 & \text{sinon} \end{cases} \qquad \square$$

Corollaire. $V_\omega^{-1} = n^{-1} V_{\omega^{-1}}$ (si n est inversible dans $\mathcal{R}$).

Multiplication rapide par FFT : premier essai

Méthode. Soit deux polynômes $A, B \in \mathcal{R}[X]$ avec $\deg(AB) < n$, alors

$$\begin{aligned} A \cdot B = A * B &= \mathsf{DFT}_w^{-1}(\mathsf{DFT}_w(A * B)) \\ &= \mathsf{DFT}_w^{-1}(\mathsf{DFT}_w(A) \cdot \mathsf{DFT}_w(B)) \\ &= \frac{1}{n}\mathsf{DFT}_{w^{-1}}(\mathsf{DFT}_w(A) \cdot \mathsf{DFT}_w(B)) \end{aligned}$$

Algèbre linéaire. Multiplication d'une matrice $n \times n$ par un vecteur de dimension n se fait en $O(n^2)$ opérations dans $\mathcal{R}$, et donc la méthode ci-dessus donne un algorithme en $O(n^2)$ opérations dans $\mathcal{R}$.

FFT. Il faut pouvoir calculer plus rapidement la fonction DFT et donc pour cela utiliser plutôt la transformée de Fourier rapide (FFT).

Multiplication rapide par FFT : Transformée de Fourier rapide

Hypothèses. $\mathcal{R}$ contient une racine primitive de l'unité d'ordre $n = 2^k$ et 2 est inversible dans $\mathcal{R}$.

Diviser pour régner. Pour $A(X) \in \mathcal{R}_n[X]$, on calcule les restes des divisions euclidiennes

$$R_\pm(X) = A(X) \mod X^{n/2} \pm 1.$$

Les polynômes R_+ et R_- sont de degré $< n/2 = 2^{k-1}$ et peuvent être calculés très facilement par la formule (en posant $m = n/2$)

$$\begin{aligned} a_{2m-1}X^{2m-1} + \cdots + a_m X^m + a_{m-1}X^{m-1} + \cdots + a_0 \mod (X^m \mp 1) \\ = (a_{2m-1} \pm a_{m-1})X^{m-1} + \cdots + (a_m \pm a_0). \end{aligned}$$

Multiplication rapide par FFT : Transformée de Fourier rapide

Rappel. $R_\pm(X) = A(X) \mod X^{n/2} \pm 1$

Suite. Pour $0 \leq l \leq n/2$, on a

$$A(\omega^{2l}) = R_-(\omega^{2l}) \quad \text{et} \quad A(\omega^{2l+1}) = R_+(\omega^{2l+1})$$

puisque $\omega^{ln} = 1$ et $\omega^{n/2} = -1$.

Comme ω^2 est une racine primitive de l'unité d'ordre $n/2$, le calcul de $\mathrm{DFT}_\omega(A)$ revient à calculer

$$\mathrm{DFT}_{\omega^2}(R_-) \quad \text{et} \quad \mathrm{DFT}_{\omega^2}(\tilde{R}_+)$$

où $\tilde{R}_+(X) = R_+(\omega X)$.

Multiplication rapide par FFT : Transformée de Fourier rapide

Algorithme.

Soit $A(X) = \sum_{i=0}^{n-1} a_i X^i \in \mathcal{R}_n[X]$ avec $n = 2^k$, calcule $\mathrm{DFT}_\omega(A)$.

(1) Si $n = 1$ alors retourner a_0

(2) Faire

$$R_- \leftarrow \sum_{i=0}^{n/2} (a_i + a_{i+n/2}) X^i \quad \text{et} \quad \tilde{R}_+ \leftarrow \sum_{i=0}^{n/2} (a_i - a_{i+n/2}) \omega^i X^i$$

(3) Calculer $\mathrm{DFT}_{\omega^2}(R_-)$ et $\mathrm{DFT}_{\omega^2}(\tilde{R}_+)$ par cet algorithme (récursif)

(4) Retourner
$(R_-(1), \tilde{R}_+(1), R_-(\omega^2), \tilde{R}_+(\omega^2), \ldots R_-(\omega^{n-2}), \tilde{R}_+(\omega^{n-2}))$

Complexité. On a $T(n) = 2T(n/2) + n$ pour les additions et $T(n) = 2T(n/2) + n/2$ pour les multiplications, et donc le coût de l'algorithme est $O(n \log n)$.

Multiplication rapide par FFT : Anneau avec FFT

Complexité de l'algorithme de multiplication par FFT

Soit $\mathcal{R}$ un anneau contenant les racines primitives de l'unité d'ordre 2^k pour tout $k \geq 1$.
Alors, il est possible de multiplier deux polynômes de $\mathcal{R}[X]$ de degré $< n$ en $O(n \log n)$ opérations de $\mathcal{R}$.

Et sinon ? Si $\mathcal{R}$ ne contient pas une racine de l'unité d'ordre 2^k, mais cependant 2 est inversible dans $\mathcal{R}$, on travaille plutôt dans

$$\mathcal{D} = \mathcal{R}[T]/(T^{2^{k-1}} + 1)$$

qui contient $\mathcal{R}$ et dans lequel $\omega = \bar{T}$ est une racine primitive de l'unité d'ordre 2^k.

Problème. Travailler dans $\mathcal{D}$ revient à travailler dans $\mathcal{R}[T]$ et on tourne en rond...

Multiplication rapide par FFT : Schönhage et Strassen

Plongements. On pose $s = 2^{\lfloor k/2 \rfloor}$ et $t = 2^{\lceil k/2 \rceil}$, ainsi on a $n = st$, et $t = s$ ou $2s$.

On pose $\mathcal{D} = \mathcal{R}[T]/(T^{2s}+1)$ et on note $\omega = \bar{T} \in \mathcal{D}$, c'est une racine de l'unité d'ordre $4s$.

Pour $A \in \mathcal{R}_n[X]$, on écrit sous la forme

$$A(X) = \sum_{i=0}^{t-1} A_i(X) \cdot X^{si}$$

avec $A_i \in \mathcal{R}_s[X]$. Puis, on définit

$$\tilde{A}(X) = \sum_{i=0}^{t-1} A_i(\omega) \cdot X^i \in \mathcal{D}_t[X] \quad \text{(Note : } t \simeq \sqrt{n}\text{)}.$$

Réciproque. On peut reconstruire A à partir de $\tilde{A}$ en remplaçant X par X^s puis ω par X.

Multiplication rapide par FFT : Schönhage et Strassen

Exemples. On considère

$$A(X) = -7X^{24} + 2X^{23} - 3X^{22} - 2X^{21} - X^{19} + 4X^{18} + 5X^{17} \\ - 6X^{16} - 7X^{15} - X^{14} + 4X^{13} - 5X^{12} - 2X^{11} - X^{10} + 4X^9 + 7X^8 \\ + 5X^7 - X^6 + 5X^5 - 7X^3 + 7X^2 - 2X + 2$$

Donc on prend $k = 5$, donc $s = 4$ et $t = 8$. On écrit

$$A(X) = -7X^{24} + (2X^3 - 3X^2 - 2X)X^{20} + (-X^3 + 4X^2 + 5X - 6)X^{16} \\ + (-7X^3 - X^2 + 4X - 5)X^{12} + (-2X^3 - X^2 + 4X + 7)X^8 \\ + (5X^3 - X^2 + 5X)X^4 + (-7X^3 + 7X^2 - 2X + 2)$$

et ainsi on a

$$\tilde{A}(X) = -7X^6 + (2\omega^3 - 3\omega^2 - 2\omega)X^5 + (-\omega^3 + 4\omega^2 + 5\omega - 6)X^4 \\ + (-7\omega^3 - \omega^2 + 4\omega - 5)X^3 + (-2\omega^3 - \omega^2 + 4\omega + 7)X^2 \\ + (5\omega^3 - \omega^2 + 5\omega)X + (-7\omega^3 + 7\omega^2 - 2\omega + 2)$$

Multiplication rapide par FFT : Schönhage et Strassen

Produit. Pour $A, B \in \mathcal{R}[X]$ avec $\deg(AB) < n$. On pose $\tilde{H} = \tilde{A}\tilde{B}$ et on note $H \in \mathcal{R}[X]$, le polynôme reconstruit à partir de $\tilde{H}$ en remplaçant X par X^s et ω par X.

Lemme

$$A(X)B(X) = H(X)$$

Preuve. On écrit $A(X) = \sum_{i=0}^{t-1} A_i(X) \cdot X^{si}$ et $B(X) = \sum_{i=0}^{t-1} B_i(X) \cdot X^{si}$, d'où

$$\tilde{A}(X)\,\tilde{B}(X) = \sum_{i=0}^{2t-2} \sum_{j=0}^{i} A_j(\omega) B_{i-j}(\omega) X^i.$$

Or $\deg(A_j B_{i-j}) < 2s$ donc il n'y a pas de "simplification" dans $\mathcal{D}$, et en remplaçant X par X^s et ω par X, on obtient

$$\sum_{i=0}^{2t-2} \sum_{j=0}^{i} A_j(X) B_{i-j}(X) X^{si} = A(X)\,B(X). \quad \square$$

Multiplication rapide par FFT : Schönhage et Strassen

Description. Pour multiplier deux polynômes $A, B \in \mathcal{R}[X]$, on prend k minimal tel que $\deg(AB) < n = 2^k$. On pose $s = 2^{\lfloor k/2 \rfloor}$ et $t = 2^{\lceil k/2 \rceil}$.

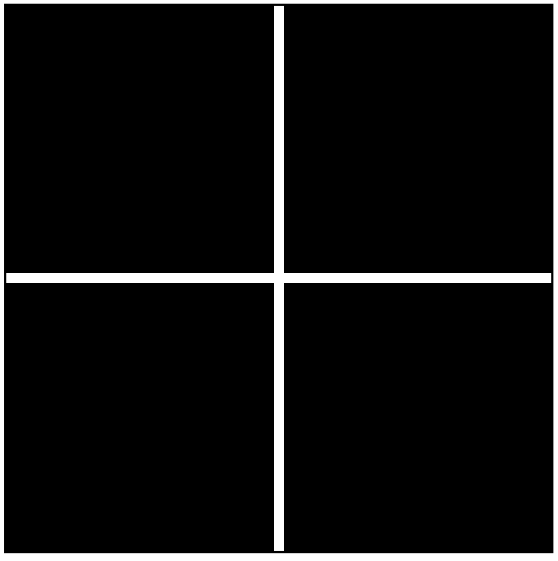
On calcule les polynômes $\tilde{A}$ et $\tilde{B}$ par la méthode expliquée ci-dessus, et on calcule le produit $\tilde{A}\tilde{B}$ dans $\mathcal{D}_t[X]$ (par FFT !) pour en déduire le résultat de AB.

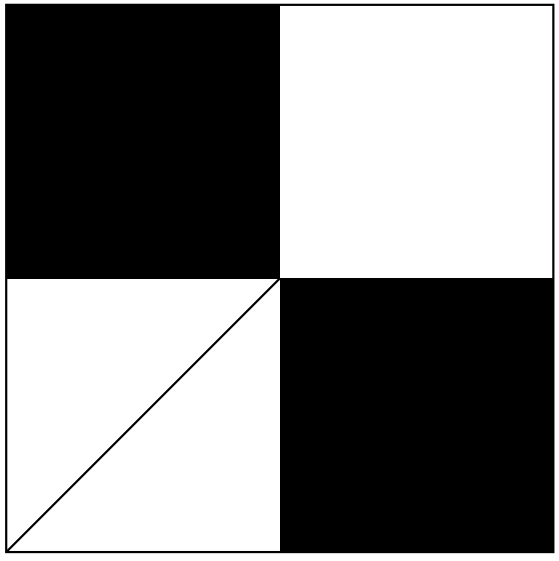
Pour calculer dans $\mathcal{D} = \mathcal{R}[T]/(T^{2s}+1)$, on utilise la même méthode récursivement (avec un degré de taille $\sqrt{n}$).

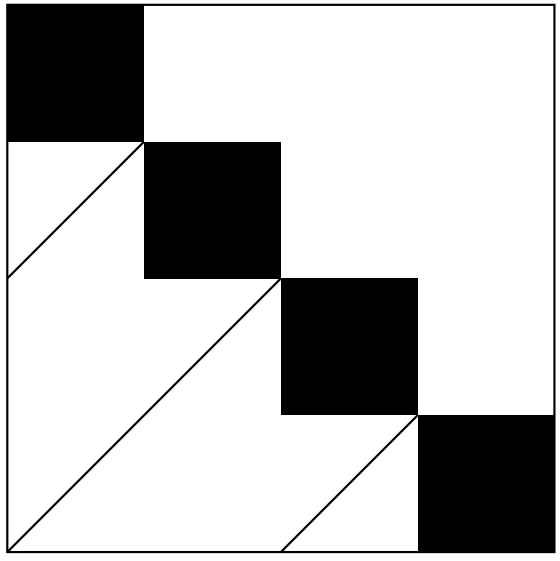
Complexité de l'algorithme de Schönhage et Strassen

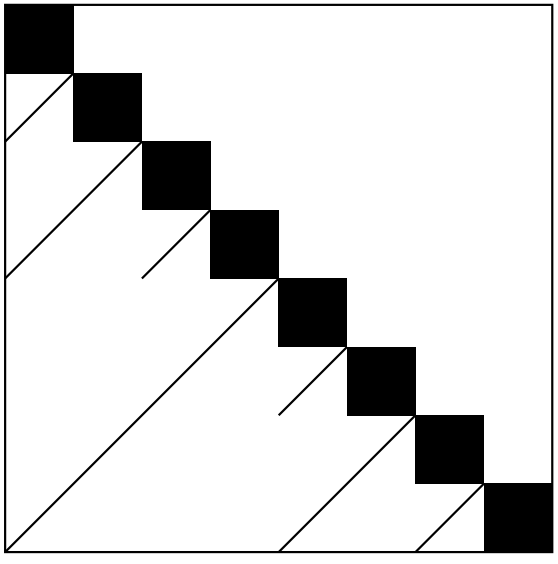
L'algorithme de Schönhage et Strassen calcule le produit de deux polynômes de $\mathcal{R}[X]$ de degré $< n$ en $O(n \log n \log\log n)$ opérations dans $\mathcal{R}$.
L'algorithme de Schönhage et Strassen calcule le produit de deux entiers de taille 2^{wn} en $O(n \log n \log\log n)$ opérations sur des entiers de w bits.

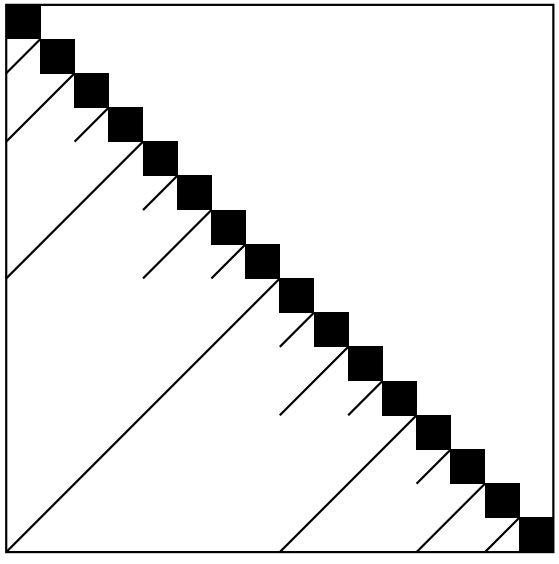
Multiplication rapide par FFT











Tend vers
une fractale
de dimension 1

Division euclidienne rapide par la méthode de Newton

Problème. On note $\deg A = n$ et $\deg B = m$ avec $n \geq m$ et on suppose que B est unitaire.

Il existe deux uniques polynômes Q et R dans $\mathcal{R}[X]$ tels que

$$A(X) = B(X) \cdot Q(X) + R(X) \qquad (*)$$

avec $\deg R < \deg B$.

L'algorithme classique calcule Q et R en $O(n^2)$ opérations dans $\mathcal{R}$.

Remarque. Si on connaît Q, on peut retrouver R en une multiplication et une addition de polynômes à coefficients dans $\mathcal{R}$. Donc il suffit de calculer Q.

Division euclidienne rapide par la méthode de Newton

Polynômes réciproques. Pour $P \in \mathcal{R}[X]$ et $k \geq \deg P$, on note

$$\mathrm{Rec}_k(P(X)) = X^k P(1/X) \in \mathcal{R}[X].$$

On remplace X par $1/X$ dans $(*)$ et on multiplie par X^n

$$X^n A\Big(\frac{1}{X}\Big) = \Big(X^m B\Big(\frac{1}{X}\Big)\Big) \cdot \Big(X^{n-m} Q\Big(\frac{1}{X}\Big)\Big) + X^{n-m+1}\Big(X^{m-1} R\Big(\frac{1}{X}\Big)\Big)$$

d'où

$$\mathrm{Rec}_n(A) \equiv \mathrm{Rec}_m(B) \cdot \mathrm{Rec}_{n-m}(Q) \pmod{X^{n-m+1}},$$

et finalement

$$\mathrm{Rec}_{n-m}(Q) = \mathrm{Rec}_n(A) \cdot \mathrm{Rec}_m(B)^{-1} \mod X^{n-m+1}.$$

Remarque. On peut facilement reconstruire Q à partir de $\mathrm{Rec}_{n-m}(Q)$.

Division euclidienne rapide par la méthode de Newton

Exemple. Prenons $\mathcal{R} = \mathbb{F}_{11}[X]$. On considère les deux polynômes

$$A(X) = 9X^5 + 10X^3 + 7X^2 + X + 10 \text{ et } B(X) = X^3 + 6X^2 + 6X + 9$$

Et donc

$$\begin{aligned} \mathrm{Rec}_5(A) &= 10X^5 + X^4 + 7X^3 + 10X^2 + 9 \\ \mathrm{Rec}_3(B) &= 9X^3 + 6X^2 + 6X + 1 \end{aligned}$$

On a $n - m + 1 = 3$ et on trouve (cf. plus loin) que

$$(9X^3 + 6X^2 + 6X + 1)(8X^2 + 5X + 1) \equiv 1 \pmod{X^3}$$

Donc

$$\mathrm{Rec}_2(Q) = 5X^2 + X + 9, \quad \text{d'où} \quad Q(X) = 9X^2 + X + 5$$

et

$$R(X) = A(X) - B(X) \cdot Q(X) = 6X + 9.$$

Division euclidienne rapide par la méthode de Newton

Reformulation. Soit $F(X) \in \mathcal{D}[X]$ avec $F(0) = 1$ et soit $l \geq 1$.

On veut trouver $G(X) \in \mathcal{D}[X]$ tel que $F(X) \cdot G(X) \equiv 1 \pmod{X^l}$.

Inversion par la méthode de Newton

On définit une suite de polynômes $G_i \in \mathcal{D}[X]$ par $G_0(X) = 1$ et, pour $i \geq 0$

$$G_{i+1}(X) = 2G_i(X) - F(X) \cdot G_i(X)^2 \mod X^{2^{i+1}}.$$

Alors, pour tout $i \geq 0$, on a $F(X) \cdot G_i(X) \equiv 1 \pmod{X^{2^i}}$.

Preuve. Clair pour $i = 0$. Par récurrence, on a, pour $i \geq 1$

$$1 - F \cdot G_{i+1} \equiv 1 - F \cdot (2G_i - F \cdot G_i^2) \equiv 1 - 2\,F \cdot G_i + (F \cdot G_i)^2$$
$$\equiv (1 - F \cdot G_i)^2 \equiv 0 \pmod{X^{2^{i+1}}}. \quad \square$$

Division euclidienne rapide par la méthode de Newton

Exemple. On calcule l'inverse de $F(X) = 9X^3 + 6X^2 + 6X + 1$ modulo X^3 dans $\mathbb{F}_{11}[X]$.

On a

$$\begin{aligned}
G_0(X) &= 1 \\
G_1(X) &= 2 \cdot 1 - (9X^3 + 6X^2 + 6X + 1) \cdot 1 \mod X^2 \\
&= 5X + 1 \\
G_2(X) &= 2 \cdot (5X + 1) - (9X^3 + 6X^2 + 6X + 1) \cdot (5X + 1)^2 \mod X^4 \\
&= 10X + 2 - (10X^3 + 3X^2 + 5X + 1) \\
&= X^3 + 8X^2 + 5X + 1
\end{aligned}$$

Donc la réponse est

$$8X^2 + 5X + 1.$$

Division euclidienne rapide par la méthode de Newton

Algorithme. Division euclidienne de A par B avec $\deg A \geq \deg B$

(1) Poser $m \leftarrow \deg A - \deg B$

(2) Calculer $R_B \leftarrow \mathrm{Rec}_{\deg B}(B)$ et poser $G \leftarrow 1$, $i \leftarrow 0$

(3) Tant que $2^i < m+1$, faire

$$i \leftarrow i+1 \quad \text{puis} \quad G \leftarrow 2G - R_B \cdot G^2 \mod X^{2^i}.$$

(4) Poser $R_Q \leftarrow \mathrm{Rec}_{\deg A}(A) \cdot G \mod X^{m+1}$

(5) Poser $Q \leftarrow \mathrm{Rec}_m(R_Q)$, $R \leftarrow A - B \cdot Q$ et retourner (Q, R)

Complexité de la division par la méthode de Newton

La méthode de Newton permet d'effectuer la division euclidienne de deux polynôme de degré $< n$ en $O(M(n))$ où $M(n)$ est le coût d'une multiplications entre deux polynômes dans $\mathcal{R}_n[X]$.
La méthode de Newton permet d'effectuer la division euclidienne de deux entiers de taille 2^{wn} en $O(n \log n \log\log n)$ opérations sur des entiers de w bits.