MULTIPLICATION RAPIDE : KARATSUBA ET FFT

1. Introduction

La multiplication est une opération élémentaire qu'on utilise évidemment très souvent, et la rapidité des nombreux algorithmes qui l'utilisent dépend fortement de la rapidité de la multiplication.

Si l'on utilise la méthode classique, pour des polynômes, ou des entiers, le coût en temps est quadratique. Nous présentons ici deux méthodes de multiplication rapide sur les polynômes : l'algorithme de Karatsuba, puis un algorithme utilisant la transformée de Fourier rapide, dite *FFT*. Ce dernier algorithme est particulièrement efficace, puisqu'il rend le coût en temps de la multiplication quasi-linéaire. Plus précisément, ce coût, pour des polynômes à coefficients dans $\mathbb{C}$ de degré inférieur à $n > 1$, est en $O(n\log(n))$ opérations sur $\mathbb{C}$. Pour des polynômes de $A[x]$, où A est un anneau quelconque, il est en $O(n\log(n)\log(\log(n)))$ opérations sur A.

Nous nous limitons ici aux polynômes, mais ces méthodes conduisent également à des algorithmes de multiplication rapide des entiers.

L'algorithme de Karatsuba n'utilise que des notions élémentaires. La FFT utilise la notion d'anneau quotient, et la notion de racine primitive de l'unité dans $\mathbb{C}$, et plus généralement dans un anneau quelconque (cette dernière notion est définie dans le texte).

2. Algorithme de multiplication de Karatsuba

Soit A un anneau commutatif unitaire. Soient $f(x) = \sum f_i x^i$ et $g(x) = \sum g_i x^i$ deux polynômes de $A[x]$ de degré inférieur ou égal à n. Le calcul classique du produit demande au plus n^2 multiplications pour les $f_i g_j$, et $(n-1)^2$ additions pour les $\sum_{i+j=k} f_i g_j$. Par exemple, le produit $(ax+b)(cx+d) = acx^2+(ad+bc)x+bd$ utilise quatre multiplications et une addition. Il y a une méthode simple pour faire mieux. On calcule ac, bd, $u = (a+b)(c+d)$, et $ad + bc = u - ac - bd$, avec trois multiplications et quatre additions. Le total des opérations est porté à sept, mais une application récursive de la méthode va diminuer le coût total quand n est grand.

On suppose que n est une puissance de 2. Soit $m = n/2$. On écrit $f = F_1 x^m + F_0$ et $g = G_1 x^m + G_0$. Suivant l'idée expliquée ci-dessus pour $n = 1$, on écrit :

$$fg = F_1 G_1 x^n + ((F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1)x^m + F_0 G_0.$$

Donc la multiplication de f et g utilise trois multiplications de polynômes de degré inférieur ou égal à m et quelques additions de polynômes.

Ainsi, si $C(n)$ est le coût en nombre d'opérations sur A de l'algorithme, on a :

$$C(n) \leq 3C(n/2) + O(n).$$

On en déduit par récurrence que $C(n)$ est en $O(n^{\log(3)})$, donc en $O(n^{1.59})$. C'est donc, au moins asymptotiquement, une bonne amélioration par rapport à la méthode classique.

Pour implémenter cet algorithme, on peut utiliser des tableaux pour représenter des polynômes, ou bien utiliser les fonctions sur les polynômes de maple, et donc rentrer les polynômes en temps que tel. Alors, pour obtenir F_0 et F_1, on peut par exemple utiliser `F1=quo(f,x^m,'F0')`.

3. Transformée de Fourier discrète sur $\mathbb{C}$

Nous allons discuter un algorithme de multiplication de polynômes de $\mathbb{C}[x]$ dans lequel le nombre d'opérations sur $\mathbb{C}$ est quasi-linéaire.

Soit n un entier > 1 et $\omega \in \mathbb{C}$ une racine primitive $n^{\text{ème}}$ de l'unité. Nous allons identifier un polynôme $f = \sum_{j=0}^{n-1} f_j x^j$ de degré strictement inférieur à n avec le vecteur $(f_0, \ldots, f_{n-1}) \in \mathbb{C}^n$ de ses coefficients. Au lieu de travailler dans $\mathbb{C}[x]$, on travaille donc dans $\mathbb{C}^n$.

Définition 3.1. (1) *L'application linéaire*

$$\begin{array}{rccc} DFT_\omega & : & \mathbb{C}^n & \longrightarrow & \mathbb{C}^n \\ & & f & \longrightarrow & \hat{f} = \left(\sum_{k=0}^{n-1} f_k, \sum_{k=0}^{n-1} f_k\omega^k, \ldots, \sum_{k=0}^{n-1} f_k\omega^{(n-1)k}\right) \end{array}$$

est appelée la transformée de Fourier discrète.

(2) *Le produit de* convolution *de $f = (f_0, \ldots, f_{n-1})$ et $g = (g_0, \ldots, g_{n-1})$ dans $\mathbb{C}^n$ est le vecteur de $\mathbb{C}^n$*

$$f *_n g = \left(\sum_{j+k \equiv l \ mod \ n} f_j g_k \right)_{l \in \{0,\ldots,n-1\}}.$$

On notera $f *_n g = f * g$ si aucune confusion n'est possible dans le contexte.

Remarquons que si $f = (f_1, \ldots, f_{n-1})$ et $F = \sum_{k=0}^{n-1} f_k x^k$, alors $\hat{f} = (F(1), F(\omega), \ldots, F(\omega^{n-1})$.

Remarquons aussi que cette notion de convolution correspond à la multiplication dans $\mathbb{C}[x]/(x^n - 1)\mathbb{C}[x]$. En effet, si $f = (f_1, \ldots, f_{n-1})$, $g = (g_1, \ldots, g_{n-1})$ $F = \sum_{k=0}^{n-1} f_k x^k$, $G = \sum_{k=0}^{n-1} g_k x^k$, $h = f *_n g = (h_1, \ldots h_{n-1})$ et $H = \sum_{k=0}^{n-1} h_k x^k$, alors

$$H \equiv FG \bmod x^n - 1.$$

On peut démontrer que ce produit de convolution est compatible avec DFT_ω, c'est-à-dire que

$$DFT_\omega(f * g) = DFT_\omega(f).DFT_\omega(g),$$

où le produit . sur $\mathbb{C}^n$ est effectué composante par composante. En fait, DFT_ω définit un isomorphisme d'algèbres de $(\mathbb{C}^n, +, *_n)$ dans $(\mathbb{C}^n, +, .)$. La matrice de cet isomorphisme est la matrice de Vandermonde V_ω de $(1, \omega, \ldots, \omega^{n-1})$

$$V_\omega = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{pmatrix}.$$

Comme 1, ω, ..., ω^{n-1} sont deux à deux distincts, la matrice V_ω est inversible. Son inverse donne la matrice permettant de calculer le polynôme d'interpolation en les points 1, ω, ..., ω^{n-1}, c'est-à-dire, les valeurs a_i étant fixées pour $i \in \{0, \ldots n-1\}$, de calculer l'unique polynôme f de degré inférieur ou égal à $n-1$ tel que $f(\omega^i) = a_i$ pour tout i. Le théorème suivant donne l'inverse de V_ω.

Théorème 3.2. *Si $\omega \in \mathbb{C}$ est une racine primitive $n^{ème}$ de l'unité, alors ω^{-1} est aussi une racine primitive $n^{ème}$ de l'unité et $V_\omega V_{\omega^{-1}} = nI$, où I est la matrice identité de taille n.*

Revenons à notre problème, qui est de multiplier deux polynômes F et G. On suppose que $\deg(FG) \leq n-1$. La transformée de Fourier discrète revient à évaluer F et G sur n points, ce qui suffit pour les définir. Dans cette représentation, la multiplication est rapide à calculer, puisque pour tout i dans $\{0 \ldots n-1\}$, $FG(\omega^i) = F(\omega^i)G(\omega^i)$. Cela nous ramène à n multiplications dans $\mathbb{C}$. Ensuite, comme on connaît le produit FG sur n points, et comme $\deg(FG) \leq n-1$, on peut reconstituer FG. Ici, on peut utiliser l'application inverse de DFT_ω, c'est-à-dire $\frac{1}{n} DFT_{\omega^{-1}}$.

Voyons maintenant comment calculer rapidement la transformée de Fourier discrète de façon rapide. On peut utiliser pour cela l'algorithme *transformée de Fourier rapide*, ou *FFT*.

On suppose que $n = 2^k$, où $n \in \mathbb{N}$. Pour évaluer F en 1, ... , ω^{n-1}, on divise d'abord F par $x^{n/2} - 1$ et $x^{n/2} + 1$:

$$F = q_0(x^{n/2} - 1) + r_0 = q_1(x^{n/2} + 1) + r_1,$$

pour q_0, r_0, q_1, r_1 dans $\mathbb{C}[x]$ de degré inférieur strictement à $n/2$. Nous n'avons besoin que de calculer r_0 et r_1. Pour cela, on remarque que si $F = F_1 x^{n/2} + F_0$, alors $x^{n/2} - 1$ divise $F - F_0 - F_1$, et donc $r_0 = F_0 + F_1$. De même, $r_1 = F_0 - F_1$. On n'a donc besoin que de n opérations dans $\mathbb{C}$ pour le calcul de r_0 et de r_1. De plus,

$$F(\omega^{2l}) = r_0(\omega^{2l}) \text{ et } F(\omega^{2l+1}) = r_1(\omega^{2l+1})$$

pour tout $l \in \{0, \ldots, n/2\}$. Il reste à évaluer r_0 en les puissances paires de ω et r_1 en les puissances impaires de ω. Or, ω^2 est une racine primitive

$(n/2)^{\text{ème}}$ de l'unité. Soit $r'_1 = r_1(\omega x)$, on est ramené à calculer DFT_{ω^2} sur r_0 et r'_1. Cette remarque permet un traitement récursif du problème.

Soit maintenant $T(n)$ le nombre d'opérations nécessaires à l'algorithme FFT. Alors $T(n) = 2T(n) + O(n)$. Ce qui mène à $T(n) = O(nlog(n))$.

Soit S est le nombre d'opérations nécessaires à la multiplication, on trouve également que $S(n) = O(nlog(n))$.

4. Transformée de Fourier discrète sur un anneau quelconque

On a donc vu que la FFT permet d'accélérer notablement la multiplication dans $\mathbb{C}[x]$. La seule propriété de $\mathbb{C}$ que nous avons utilisée pour cela est que $\mathbb{C}$ contient pour tout n les racines $n^{\text{ème}}$ de l'unité. Nous voudrions utiliser le même procédé dans le cas de $A[x]$, dans le cas où A est un anneau quelconque.

Si A contient les racines $n^{\text{ème}}$ de l'unité, et si n est inversible dans A, la même méthode s'applique. Sinon, il est possible d'ajouter à l'anneau des racines de l'unité. Dans ce qui suit, A est un anneau commutatif et unitaire dans lequel n est inversible.

Définition 4.1. *On dit que A contient une racine primitive $n^{ème}$ de l'unité s'il existe ω dans A tel que*

1. $\omega^n = 1$
2. *Pour tout diviseur premier l de n, $\omega^{n/l} - 1$ n'est pas un diviseur de 0. C'est-à-dire, pour tout $a \in A \setminus \{0\}$, $a(\omega^l - 1) \neq 0$.*

Exemples. $A = \mathbb{C}$, $A = \mathbb{F}_q$ où q est une puissance d'un nombre premier et où n divise $q-1$. Si $A = \mathbb{Z}/(2^{2^k}+1)\mathbb{Z}$, et si $n = 2^{k+1}$, alors 2 est une racine primitive $n^{\text{ème}}$ de l'unité dans A.

Proposition 4.2. *Si ω est une racine primitive $n^{ème}$ de l'unité dans A, alors pour tout $l \in \{1, \ldots, n\}$, $\omega^l - 1$ n'est pas un diviseur de 0. De plus :*

$$\sum_{j=0}^{n-1} \omega^{lj} = 0.$$

Grâce à cette proposition, on peut montrer que l'algorithme décrit dans le paragraphe précédent pour $\mathbb{C}$ s'applique également pour un anneau A contenant une racine primitive $n^{\text{ème}}$ de l'unité.

On ne suppose plus maintenant que A contient des racines primitives $n^{\text{ème}}$ de l'unité, mais seulement que 2 est inversible dans A. On peut encore appliquer une méthode similaire. L'idée est de rajouter les racines de l'unité qui manquent.

On suppose que n est une puissance de 2 et on pose $n = 2^k$. On veut multiplier entre eux deux polynômes F et G à coefficients dans A tels que

$\deg(FG \leq n)$. Soient $m = 2^{\lfloor k/2 \rfloor}$ et $t = 2^{\lceil k/2 \rceil}$. On définit les polynômes F_0, ... , F_{t-1}, G_0, ... , G_{t-1} de degrés strictement inférieur à m tels que

$$F = \sum_{j=0}^{t-1} F_j x^{mj} \text{ et } G = \sum_{j=0}^{t-1} G_j x^{mj}.$$

On pose alors :

$$F' = \sum_{j=0}^{t-1} F_j(x) y^j \text{ et } G' = \sum_{j=0}^{t-1} G_j(x) y^j,$$

de sorte que $F(x) = F'(x, x^m)$ et $G(x) = G'(x, x^m)$. Le degré en y de $F'G'$ est strictement inférieur à $2t \leq 4m$. De plus, les coefficients de $F'G'$ sont des polynômes en x de degré strictement inférieur à $2m$, de telle sorte qu'il suffit de connaître leur image dans $D = A[x]/(x^{2m} + 1)A[x]$.

On peut démontrer que l'image ω de x dans D est une racine primitive $4m^{\text{ème}}$ de l'unité dans D. On se retrouve donc dans la situation précédente. Une analyse de l'algorithme montrerait que coût en nombre d'opérations est en $O(nlog(n)log(log(n)))$ opérations dans A (mais il est déconseillé de se lancer dans cette analyse).

5. Suggestions

Plusieurs affirmations sont données sans preuve. Le candidat est invité à les démontrer.

Il pourra illuster sur des exemples certains des algorithmes proposés, puis les implémenter et les commenter.

On peut aussi réfléchir à des applications de ces algorithmes.