COURS 6

Récurrences linéaires à coefficients polynomiaux : n-ième terme, n premiers termes

Résumé

Le calcul du n-ième terme d'une suite donnée par une récurrence linéaire à coefficients polynomiaux intervient fréquemment en combinatoire et pour calculer des troncatures de séries, ainsi que comme étape clé dans des algorithmes de recherche de solutions polynomiales d'équations fonctionnelles linéaires et dans un algorithme de factorisation déterministe d'entiers. Pour une récurrence d'ordre r dont les coefficients ont degré au plus d, l'algorithme naïf par déroulement de la récurrence a complexité arithmétique $O(rdn)$ et complexité binaire $O\big(rn^2 I(d\log n)\big)$. Ces complexités sont optimales pour le calcul de tous les n premiers termes. Pour le calcul du n-ième terme seul, la technique des pas de bébés et pas de géants atteint une meilleure complexité arithmétique en $O\big(r^2 M(\sqrt{dn}\,)\log dn + r^\omega M(\sqrt{dn}\,)\big)$; la méthode du scindage binaire fournit une complexité binaire quasi-optimale en $O\big(r^\omega I(dn\log n)\log n\big)$.

Le chapitre 4 a montré comment calculer le n-ième terme d'une suite donnée par une récurrence à coefficients contants en complexité arithmétique en $O(\log n)$. Dans ce chapitre, nous nous attaquons au cadre plus général des récurrences à coefficients polynomiaux. Les algorithmes n'ont pas une aussi bonne complexité que pour des coefficients constants, mais cette fois encore, il existe des algorithmes plus efficaces que le simple déroulement de la récurrence.

La situation se distingue de celle des problèmes et algorithmes présentés jusqu'ici : les idées qui permettent le calcul du n-ième terme en bonne complexité arithmétique ne sont pas les mêmes que pour abaisser la complexité binaire. Ainsi, l'algorithme par pas de bébés et pas de géants de la section 2 donne un gain en racine de n sur la complexité arithmétique, par rapport à la méthode naïve. Ce gain est typique de la technique. Quant à l'algorithme par scindage binaire de la section 3, il n'abaisse en rien la complexité arithmétique, mais améliore pourtant la complexité binaire jusqu'à la rendre quasi-linéaire en la taille de sa sortie.

Nous terminons cette introduction par un peu de terminologie. Les solutions u de suites de la forme

$$p_r(n)u_{n+r} + \cdots + p_0(n)u_n = 0, \qquad (n \in \mathbb{N}), \tag{1}$$

pour des polynômes p_i sont dites *polynomialement récursives* ou en abrégé *P-récursives*. Dans le cas spécifique d'ordre 1, une suite u est dite *hypergéométrique*. Remarquons que la définition se récrit alors sous la forme suivante, plus facile à mémoriser :

$$\frac{u_{n+1}}{u_n} = -\frac{p_0(n)}{p_1(n)}.$$

Autrement dit, le quotient de deux termes successifs de la suite est donné par l'évaluation d'une fraction rationnelle fixée, sauf peut-être en un nombre fini de valeurs de n. Les suites P-récursives et encore plus les suites hypergéométriques joueront un rôle important dans des chapitres ultérieurs.

1. Calcul naïf de $n!$ et de suites P-récursives

La définition de la factorielle comme produit,

$$n! = 1 \times 2 \times \cdots \times n,$$

montre que $n!$ peut être calculé en $n-1$ opérations arithmétiques.

L'estimation de la complexité binaire de cette méthode repose sur la *formule de Stirling* :

$$\log n! = n \log n - n \log e + \frac{1}{2} \log n + O(1), \qquad n \to \infty.$$

En particulier, nous retiendrons l'équivalence $\log n! \sim n \log n$ qui donne la taille de $n!$. La k-ième étape du produit multiplie $k!$ par $k+1$, et donc un entier de taille $\log k! = O(k \log n)$ avec un entier de taille $\log(k+1) = O(\log n)$. Ce produit déséquilibré coûte donc moins de $ckI(\log n)$ opérations binaires pour une certaine constante c. Le calcul complet de la factorielle par la méthode naïve effectue donc moins de

$$\sum_{k=1}^{n} ckI(\log n) = O\big(n^2 I(\log n)\big)$$

opérations binaires.

Ces complexités arithmétique et binaire sont quasi-optimales pour le calcul conjoint des nombres $1!$, $2!$, …, $n!$.

Pour une suite P-récursive donnée par une récurrence de la forme (1), le calcul de u_{n+r} à partir des r valeurs précédentes de la suite demande $O(rd)$ opérations arithmétiques pour l'évaluation des polynômes (par exemple par le schéma de Horner) puis $O(r)$ opérations pour effectuer la combinaison linéaire des u_{n+i}. Le calcul de u_n à partir des r premières valeurs de la suite demande donc $O(rdn)$ opérations arithmétiques.

La complexité binaire par cette méthode découle de nouveau essentiellement de la croissance de u_n. Pour estimer celle-ci, il est agréable de faire apparaître u_n comme le quotient v_n/w_n des suites définies par les récurrences

$$v_{n+r} + p_{r-1} v_{n+r-1} \cdots + p_0(n) v_n = 0, \qquad p_r(n) w_{n+r} = w_{n+r-1}, \qquad (n \in \mathbb{N}),$$

et les conditions initiales $v_i = u_i$ et $w_i = 1$ quand $0 \le i < r$. Par une vision matricielle, v_n est donné comme première coordonnée du vecteur V_n de la suite vectorielle définie par la récurrence du premier ordre

$$V_{n+1} = A(n) V_n \quad \text{avec} \quad A(n) = \frac{1}{p_r(n)} \begin{pmatrix} 0 & p_r(n) & & & \\ & 0 & p_r(n) & & \\ & & \ddots & \ddots & \\ & & & 0 & p_r(n) \\ -p_0(n) & & \dots & & -p_{r-1}(n) \end{pmatrix}.$$

La matrice $A(n)$ se développe asymptotiquement sous la forme $A_\delta n^\delta + A_{\delta-1} n^{\delta-1} + \dots$, pour des matrices constantes A_i et un entier δ entre $-d$ et d. Il s'ensuit que

sa norme est asymptotiquement équivalente à Cn^δ pour une certaine constante C, après tout choix de norme sur les matrices $r \times r$. Les majorations

$$|v_n| \leq |V_n| \leq |A(n)| \dots |A(1)||U_0| \leq C^n (n!)^\delta \leq C^n (n!)^d$$

fournissent la borne $O(dn \log n)$ sur la taille $\log |v_n|$. Par le même raisonnement, la taille du dénominateur w_n est du même ordre, si bien que par le même type d'argument que pour la factorielle, la complexité binaire du calcul naïf de u_n est $O\big(n^2 I(d \log n)\big)$.

2. Pas de bébés et pas de géants

On sait que le n-ième terme d'une récurrence linéaire à coefficients constants peut être calculé en complexité arithmétique $O(\log n)$. Dans le cas des coefficients polynomiaux, les choses se compliquent : à ce jour, on ne connaît pas d'algorithme polynomial en $\log n$. L'exemple typique en est le calcul du n-ième terme de la factorielle $u_n = n!$, qui vérifie la récurrence à coefficients polynomiaux $u_{n+1} = (n+1)u_n$ pour $n \geq 0$. Une solution efficace exploite le théorème 1 du chapitre 5. Elle utilise la technique des *pas de bébés et pas de géants* et requiert un nombre d'opérations arithmétiques en $\sqrt{n}$ (à des facteurs logarithmiques près). Pour simplifier la présentation, supposons que n est un carré parfait. L'idée de l'algorithme est de poser

$$P(X) = (X+1)(X+2)\dots(X+n^{1/2}),$$

afin d'obtenir la valeur de u_n à l'aide de l'équation

$$u_n = \prod_{j=0}^{n^{1/2}-1} P\big(jn^{1/2}\big). \tag{2}$$

Cette égalité suggère la procédure suivante :

1. *Pas de bébés* : Calculer les coefficients de P. Ceci peut être fait en $O\big(\mathsf{M}(\sqrt{n})\log n\big)$ opérations arithmétiques (en construisant un arbre binaire de feuilles $X+i$, voir le chapitre 5).
2. *Pas de géants* : Évaluer P sur les points 0, $\sqrt{n}$, $2\sqrt{n}$, …, $(\sqrt{n}-1)\sqrt{n}$ et retrouver la valeur de u_n à l'aide de l'équation (2). En utilisant le théorème 1 du chapitre 5, ceci se fait en $O\big(\mathsf{M}(\sqrt{n})\log n\big)$ opérations arithmétiques.

Le coût total de cet algorithme est de $O\big(\mathsf{M}(\sqrt{n})\log n\big)$ opérations arithmétiques. Si la FFT est utilisée pour la multiplication des polynômes, le gain par rapport à la méthode directe est de l'ordre de $\sqrt{n}$, à des facteurs logarithmiques près, typique pour la technique des pas de bébés et pas de géants.

Ce résultat se généralise au problème du calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux. À cette fin, le polynôme à considérer est maintenant le polynôme matriciel

$$P(X) = A(X+m)\dots A(X+2)A(X+1),$$

pour $m = (n/d)^{1/2}$. Le produit $A(n)\dots A(1)$ est alors le produit de $(dn)^{1/2}$ évaluations de P, lequel a degré $(dn)^{1/2}$. L'algorithme obtient ces évaluations matricielles en réalisant successivement les évaluations multipoints des r^2 coordonnées de la matrice P.

EXERCICE 1. Généraliser ce résultat au calcul du Nième terme d'une récurrence d'ordre quelconque.

EXERCICE 2. étant donné un polynôme $f \in k[X]$ de degré 2 et un entier $N \geq 0$, quel est le nombre d'opérations dans k nécessaires pour déterminer le coefficient de X^N du polynôme f^N ? (Indication : La solution directe consiste à calculer tous les coefficients u_n de f^N modulo X^{N+1} par exponentiation binaire. Son coût est de $O\big(\mathsf{M}(N)\big)$ opérations arithmétiques. Une approche plus rapide repose sur le fait que les u_n satisfont une récurrence à coefficients polynomiaux d'ordre 2.)

EXERCICE 3. Imaginez un algorithme qui teste si une équation différentielle à coefficients dans $\mathbb{Z}[X]$ admet une solution polynomiale de degré au plus N, en $O(\mathsf{M}(\sqrt{N})\log(N))$ opérations bits.

2.1. Factorisation déterministe des entiers. Supposons que nous devons factoriser un entier N. Tester tous les diviseurs plus petits que $\sqrt{N}$ a un coût linéaire en $\sqrt{N}$ (dans ce paragraphe, par coût on entend complexité bit). Afin d'accélérer ce calcul, Strassen [3] propose de rassembler tous les entiers plus petits que $\sqrt{N}$ en $\sqrt[4]{N}$ blocs, chacun contenant $\sqrt[4]{N}$ entiers consécutifs. Soit c de l'ordre de $\sqrt[4]{N}$ et notons $f_0 = 1 \cdots c \bmod N$, $f_1 = (c+1)\cdots(2c) \bmod N$, ..., $f_{c-1} = (c^2-c+1)\cdots(c^2) \bmod N$. Si les valeurs $f_0, \ldots, f_{c-1}$ sont connues, alors il devient facile de déterminer un facteur de N, en prenant des pgcd de $f_0, \ldots, f_{c-1}$ avec N. Ainsi, la principale difficulté est de calculer les valeurs f_i.

Pour ce faire, on prend $R = \mathbb{Z}/N\mathbb{Z}$ et F le polynôme $(X+1)\cdots(X+c) \in R[X]$, qu'on évalue en les points $0, c, 2c, \ldots, c(c-1)$ en $O(\mathsf{M}(c)\log c)$ opérations de R. Par le Théorème 1 et puisque c est d'ordre $\sqrt[4]{N}$, la complexité totale est de $O(\mathsf{M}(\sqrt[4]{N})\log N)$ opérations modulo N, soit $O(\mathsf{M}(\sqrt[4]{N})\,\mathsf{M}(\log N)\log N)$ opérations bits. Notons qu'on ne connaît pas de meilleur algorithme déterministe ; l'existence d'un tel algorithme est un grand problème ouvert.

3. Scindage binaire

3.1. Cas de la factorielle. Pour exploiter la multiplication rapide, l'idée consiste à équilibrer les produits en calculant récursivement $P(a,b) = (a+1)(a+2)\cdots b$ par

$$P(a,b) = P(a,m)P(m,b) \quad \text{où} \quad m = \left\lfloor \frac{a+b}{2} \right\rfloor.$$

Appelons $C(a,b)$ le coût binaire du calcul de $P(a,b)$. Il résulte immédiatement de la méthode que ce coût vérifie l'inégalité

$$C(a,b) \leq C(a,m) + C(m,b) + I\big(\log P(a,m), \log P(m,b)\big).$$

Sous l'hypothèse raisonnable que la complexité de la multiplication d'entiers est croissante avec la taille des entiers, le coût du calcul de $P(a,m)$ est inférieur au coût du calcul de $P(m,b)$, d'où la nouvelle inégalité

$$C(a,b) \leq 2C(m,b) + I\big(P(m,b)\big).$$

À ce stade, le lemme « diviser pour régner » n'est pas suffisant pour conclure, mais l'utilisation de l'inégalité précédente donne les inégalités successives

$$\begin{aligned} C(0,n) &\le 2C(n/2,n) + I\big(\log P(n/2,n)\big) \\ &\le 4C(3n/4,n) + 2I\big(\log P(3n/4,n)\big) + I\big(\log P(n/2,n)\big) \\ &\le \cdots \\ &\le 2^k C(n-2^{-k}n,n) + 2^k I\big(\log P(n-2^{-k}n,n)\big) + \cdots + I\big(\log P(n/2,n)\big), \end{aligned}$$

pour tout entier positif k. L'entier $P(n-2^{-k}n,n)$ est le produit de $2^{-k}n$ facteurs de taille bornée par $\log n$; il a donc pour taille $2^{-k}n\log n$. Par sous-additivité de la fonction I, la dernière inégalité devient

$$C(0,n) \le 2^k C(n/2^k,n) + kI(n\log n).$$

En choisissant finalement d'arrêter la récursion lorsque $n-2^{-k}n$ et n diffèrent d'au plus un, ce qui impose k de l'ordre de $\log n$, on aboutit à la borne

$$C(0,n) \le O(\log n) + I(n\log n)\log n$$

donc à une complexité binaire dans $O\big(I(n\log n)\log n\big)$.

Si la multiplication utilisée est la FFT, cette complexité s'écrit $O(n\log^3 n\log\log n)$; si la multiplication est d'exposant de n strictement plus grand que 1, elle s'écrit $O\big(I(n\log n)\big)$.

3.2. Récurrences d'ordre 1. La factorielle de la section précédente suit la récurrence $u_{n+1} = (n+1)u_n$. On considère ici tout d'abord les solutions de récurrences de la forme

$$u_{n+1} = p(n)u_n, \qquad p \in \mathbb{Z}[X].$$

Si p a degré d, $\log p(n)$ est dans $O(d\log n)$, si bien que la taille de u_n est $O(dn\log n)$.

De même, pour toute récurrence de la forme

$$u_{n+1} = \frac{p(n)}{q(n)}u_n, \qquad p,q \in \mathbb{Z}[X],$$

on peut pour calculer u_n appliquer séparément la même technique de scindage binaire sur le numérateur et le dénominateur. Si d est le maximum des degrés de p et q, le calcul produit deux entiers de taille $O(dn\log n)$ en un nombre d'opérations binaires en $O\big(I(dn\log n)\log n\big)$. Ensuite, si le dénominateur est non nul, la méthode de Newton permet d'effectuer la division finale en $O\big(I(dn\log n)\big)$ opérations binaires.

3.3. Calcul de $e = \exp(1)$. Le point de départ est la suite (e_n) donnée par

$$e_n = \sum_{k=0}^{n} \frac{1}{k!}.$$

Cette suite converge vers e. Plus précisément, il est classique que le terme de reste dans $e - e_n$ se majore par une série géométrique de sorte que

$$0 \le e - e_n \le \frac{1}{n\,n!}.$$

Pour calculer N décimales de e par cette série, il suffit de rendre $\frac{1}{n\,n!}$ inférieur à 10^{-N}. Il s'ensuit qu'il suffit de prendre N de sorte que $n\,n!$ et 10^N soient du

même ordre, c'est-à-dire d'avoir N proportionnel à $n \log n$. Il suffit donc de prendre $n = O(N/\log N)$ termes dans la série.

La suite $(e_n)_{n\geq 0}$ vérifie $e_n - e_{n-1} = 1/n!$ et donc

$$n(e_n - e_{n-1}) = e_{n-1} - e_{n-2}.$$

Cette récurrence se récrit

$$\begin{pmatrix} e_n \\ e_{n-1} \end{pmatrix} = \frac{1}{n} \underbrace{\begin{pmatrix} n+1 & -1 \\ n & 0 \end{pmatrix}}_{A(n)} \begin{pmatrix} e_{n-1} \\ e_{n-2} \end{pmatrix} = \frac{1}{n!} A(n)A(n-1)\cdots A(2) \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Le calcul du produit de matrices peut alors être effectué par scindage binaire. Le calcul de e_N par ce procédé demande donc $O(I(N \log N) \log N)$ opérations binaires.

Pour calculer n décimales de e, la première étape ci-dessus construit deux entiers de taille $O(N \log N) = O(n)$ en $O(I(n) \log n)$ opérations binaires. Il faut ensuite effectuer une division qui ne demande que $O(I(n))$ opérations par l'algorithme de Newton. L'ensemble du calcul est donc quasi-optimal. (En outre, le $\log n$ n'est pas présent pour des multiplications comme celle de Karatsuba dont l'exposant de complexité est supérieur à 1.)

Un autre exemple d'application est donné dans les notes.

3.4. Suites polynomialement récursives.

DEFINITION 1. Une suite $(a_n)_{n\geq 0}$ d'éléments d'un anneau commutatif unitaire $\mathbb{A}$ est appelée suite polynomialement récursive (ou P-récursive) si elle vérifie une récurrence de la forme

$$p_d(n)a_{n+d} + p_{d-1}(n)a_{n+d-1} + \cdots + p_0(n)a_n = 0, \qquad n \geq 0,$$

où les p_i sont des polynômes de $\mathbb{A}[X]$.

Dans la suite on fait l'hypothèse que le coefficient de tête p_d ne s'annule pas sur les entiers $0, \ldots, N-d$, où N est l'indice du terme maximal que l'on souhaite calculer. On note D une borne sur le degrés des polynômes p_i et, par h une borne sur la taille de leurs coefficients lorsque ceux-ci sont entiers.

THÉORÈME 1. *Avec les notations de la definition 1, le calcul de $a_0, \ldots, a_N$ peut être effectué en $O(dNM(D)/D)$ opérations arithmétiques. Dans le cas où $p_i \in \mathbb{Z}[X]$, $i = 0, \ldots, d$, la complexité binaire de la méthode naïve est en $O(dN^2 I(hD \log N))$ et celle du scindage binaire pour calculer a_N est en $O(d^\omega I(hDN \log N) \log N)$.*

Comme précédemment, cette dernière complexité descend à $O(d^\omega I(hDN \log N))$ si l'exposant de la complexité I est supérieur à 1.

EXERCICE 4. Vérifier les formules de ce théorème.

Notes

L'algorithme par pas de bébés et pas de géants a été introduit par Strassen [3] et généralisé dans [1] au problème du calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux.

L'exercice 2 est inspiré de [2, Pb. 4].

Le même raisonnement se généralise au calcul des sommes convergentes de suites hypergéométriques. En particulier, c'est ainsi que les systèmes de calcul formel calculent rapidement π par une formule découverte en 1989 par les frères Chudnovsky :

$$\frac{1}{\pi} = \frac{12}{C^{3/2}} \sum_{n=0}^{\infty} \frac{(-1)^n (6n)!(A+nB)}{(3n)!n!^3 C^{3n}}$$

où $A = 13591409$, $B = 545140134$ et $C = 640320$.

Bibliographie

[1] Chudnovsky (D. V.) and Chudnovsky (G. V.). – Approximations and complex multiplication according to Ramanujan. In *Ramanujan revisited*, pp. 375–472. – Academic Press, Boston, MA, 1988.

[2] Flajolet (Philippe) and Salvy (Bruno). – The Sigsam challenges : Symbolic asymptotics in practice. *SIGSAM Bulletin*, vol. 31, n° 4, December 1997, pp. 36–47.

[3] Strassen (V.). – Einige Resultate über Berechnungskomplexität. *Jber. Deutsch. Math.-Verein.*, vol. 78, n° 1, 1976/77, pp. 1–8.