COURS 1

Introduction

Le calcul formel calcule des objets mathématiques exacts. Ce cours « Algorithmes efficaces en calcul formel » explore deux directions : la calculabilité et l'efficacité. En particulier, de nombreuses questions portant sur les objets fondamentaux que sont les entiers, les polynômes et les séries admettent une réponse en complexité quasi-optimale.

Le choix des sujets couverts par le cours est guidé par un objectif simple : montrer comment des calculs d'intégrales ou de sommes de fonctions ou de suites spéciales de la combinatoire ou de la physique mathématique peuvent être abordés algorithmiquement. Les techniques reposent sur des calculs d'élimination et nous dédions de ce fait une grande place à l'étude effective des systèmes polynomiaux (bases standard, résolution géométrique) où ces questions sont classiques. Le souhait d'aboutir à une bonne complexité nous amène à traiter dans une première partie du cours l'algorithmique de base du calcul formel du point de vue de l'efficacité. Ce chapitre introductif présente rapidement le calcul formel et les notions de complexité, tels qu'ils seront développés dans l'ensemble du cours.

1. Décider, calculer

1.1. Fondements logiques. D'une certaine manière, le calcul formel est fondé sur une contrainte d'origine logique.

THÉORÈME 1 (Richardson). *Dans la classe des expressions obtenues à partir de* $\mathbb{Q}(x)$, π, $\log 2$ *par les opérations* $+, -, \times$ *et la composition avec* $\exp$, $\sin$ *et* $|\cdot|$, *le test d'équivalence à 0 est indécidable.*

Autrement dit, il n'existe pas d'algorithme permettant pour toute expression de cette classe de déterminer en temps fini si elle vaut 0 ou non. Plus généralement tout test d'égalité peut bien entendu se ramener à tester l'égalité à zéro dès que la soustraction existe. Cette limitation de nature théorique explique la difficulté et parfois la frustration que rencontrent les utilisateurs débutants des systèmes de calcul formel face à des fonctions de « simplification », qui ne peuvent être qu'heuristiques.

Pour effectuer un calcul, il est pourtant souvent crucial de déterminer si des expressions représentent 0 ou non, en particulier pour évaluer une fonction qui possède des singularités (comme la division). L'approche du calculateur formel expérimenté consiste à se ramener autant que faire se peut à des opérations d'un domaine dans lequel le test à zéro est décidable. Le calcul formel repose ainsi de manière naturelle sur des constructions algébriques qui préservent la décidabilité du test à 0. En particulier, les opérations courantes sur les vecteurs, matrices, polynômes, fractions rationnelles, ne nécessitent pas d'autre test à 0 que celui des coefficients.

1.2. Structures de base. Les objets les plus fondamentaux sont assez faciles à représenter en machine de manière exacte. Nous considérons tour à tour les plus importants d'entre eux, en commençant par les plus basiques.

Entiers machine. Les entiers fournis par les processeurs sont des entiers modulo une puissance de 2 (le nombre de bits d'un mot machine, typiquement 32 ou 64). Ils sont appelés des *entiers machine*. Les opérations rendues disponibles par le processeur sont l'addition, la soustraction, la multiplication et parfois la division. La norme ANSI du langage C fournit au programmeur la division et le modulo pour ces entiers, c'est-à-dire que le compilateur implante ces opérations si le processeur ne le fait pas.

Entiers. Pour manipuler des entiers dont la taille dépasse celle d'un mot machine, il est commode de les considérer comme écrits dans une base B assez grande :

$$N = a_0 + a_1 B + \cdots + a_k B^k.$$

L'écriture est unique si l'on impose $0 \leq a_i < B$. (Le signe est stocké séparément.) Ces nombres peuvent être stockés dans des tableaux d'entiers machine. Les objets obtenus sont des entiers de taille arbitraire appelés parfois *bignums*.

L'addition et le produit peuvent alors être réduits à des opérations sur des entiers inférieurs à B^2, au prix de quelques opérations de propagation de retenue. Le choix de B dépend un peu du processeur. Si le processeur dispose d'une instruction effectuant le produit de deux entiers de taille égale à celle d'un mot machine, renvoyant le résultat dans deux mots machines, alors B pourra être pris aussi grand que le plus grand entier tenant dans un mot machine. Sinon, c'est la racine carré de ce nombre qui sera utilisée pour B.

Entiers modulaires. Les calculs avec des polynômes, des fractions rationnelles ou des matrices à coefficients entiers souffrent souvent d'une maladie propre au calcul formel : la croissance des expressions intermédiaires. Les entiers produits comme coefficients des expressions intervenant lors du calcul sont de taille disproportionnée par rapport à ceux qui figurent dans l'entrée et dans la sortie.

Exemple 1. Voici le déroulement typique du calcul du plus grand diviseur commun (pgcd) de deux polynômes à coefficients entiers par l'algorithme d'Euclide :

$$P_0 = 7x^5 - 22x^4 + 55x^3 + 94x^2 - 87x + 56,$$
$$P_1 = 62x^4 - 97x^3 + 73x^2 + 4x + 83,$$
$$P_2 = \mathrm{rem}(P_0, P_1) = \frac{113293}{3844}x^3 + \frac{409605}{3844}x^2 - \frac{183855}{1922}x + \frac{272119}{3844},$$
$$P_3 = \mathrm{rem}(P_1, P_2) = \frac{18423282923092}{12835303849}x^2 - \frac{15239170790368}{12835303849}x + \frac{10966361258256}{12835303849},$$
$$P_4 = \mathrm{rem}(P_2, P_3) = -\frac{216132274653792395448637}{44148979404824831944178}x - \frac{631179956389122192280133}{88297958809649663888356},$$
$$P_5 = \mathrm{rem}(P_3, P_4) = \frac{20556791167692068695002336923491296504125}{3639427682941980248860941972667354081}.$$

Chaque étape calcule le reste (noté rem pour *remainder*) de la division euclidienne des deux polynômes précédents. Les coefficients de ces polynômes intermédiaires

font intervenir des entiers qui croissent de manière exponentielle, alors que le résultat recherché est 1.

Les entiers modulaires remédient à ce problème de deux manières. D'une part, pour un calcul de décision, de dimension, ou de degré, l'exécution de l'algorithme sur la réduction de l'entrée modulo un nombre premier donne un algorithme probabiliste répondant à la question. Cette technique peut aussi servir de base à un algorithme déterministe lorsque les nombres premiers pour lesquels la réponse est fausse peuvent être maîtrisés. C'est le cas du pgcd : en évitant les premiers qui divisent les coefficients de tête des deux polynômes, le degré du pgcd modulaire est le même que le degré du pgcd exact.

D'autre part, les entiers modulaires sont utilisés dans les algorithmes reposant sur le théorème des restes chinois. Ce théorème indique qu'un entier inférieur au produit de nombres premiers $p_1 \cdots p_k$ peut être reconstruit à partir de ses réductions modulo $p_1, \ldots, p_k$. Lorsqu'une borne sur la taille du résultat est disponible, il suffit d'effectuer le calcul modulo suffisamment de nombres premiers (choisis assez grands pour que leur nombre soit faible et assez petits pour que les opérations tiennent dans un mot machine), pour ensuite reconstruire le résultat, court-circuitant de la sorte toute croissance intermédiaire.

Rationnels. Les rationnels peuvent être stockés comme des paires où numérateur et dénominateur sont des entiers de taille arbitraire. Les opérations d'addition et de multiplication se réduisent aux opérations analogues sur les entiers et le test d'égalité à zéro se réduit au test d'égalité à 0 sur le numérateur. L'implantation d'un calcul de plus grand dénominateur commun (pgcd) permet aussi de réduire ces fractions et donne une forme normale où le test d'égalité est facile.

Vecteurs et matrices. Une fois donnée une représentation exacte pour des coefficients, il est facile de construire des vecteurs ou matrices comme des tableaux, ou plus souvent comme des tableaux de pointeurs sur les coefficients. Les opérations de produit par un scalaire, de produit de matrices ou de produit d'une matrice par un vecteur se réduisent aux opérations d'addition et de multiplication sur les coefficients. Il en va de même de la recherche de noyau ou d'inverse de matrices.

Polynômes et fractions rationnelles. Les polynômes peuvent être stockés de plusieurs manières, et la meilleure représentation dépend des opérations que l'on souhaite effectuer. Pour un polynôme en une variable, les choix principaux sont :

- la représentation dense : comme pour les entiers, le polynôme est représenté comme un tableau de (pointeurs sur les) coefficients ;
- la représentation creuse : le polynôme est représenté comme une liste de paires (coefficient, exposant) généralement triée par les exposants.

Ces représentations peuvent être utilisées récursivement pour stocker des polynômes multivariés. De même que pour les entiers, les fractions rationnelles sont représentées par des paires de polynômes et la réduction est possible dès lors qu'un pgcd est disponible. Les opérations d'addition, produit, division euclidienne, pgcd, se réduisent aux additions et multiplications sur les coefficients.

Ces constructions sont possibles dès que les coefficients sont disponibles. Il est donc possible par exemple de manipuler des polynômes dont les coefficients sont des rationnels, des entiers modulaires, ou des matrices.

Séries tronquées. Les séries tronquées

$$\sum_{k=0}^{N} a_k x^k + O(x^{N+1})$$

se représentent pratiquement comme des polynômes. La différence principale apparaît lors du produit : les coefficients des termes d'exposant au moins $N+1$ n'ont pas besoin d'être calculés, ni stockés. Cette structure de données joue un rôle très important non seulement pour des calculs d'approximations, mais aussi, comme on le verra dans le cours, comme une représentation *exacte*. En voici trois exemples importants qui seront abordés dans le cours :

1. Une fraction rationnelle dont les numérateurs et dénominateurs ont degré borné par d peut être reconstruite à partir d'un développement en série à l'ordre $2d+1$. Cette représentation joue ainsi un rôle clé dans le calcul efficace de la division euclidienne de polynômes (Cours 3) ; de suites récurrentes linéaires, comme le calcul rapide du 10 000$^{\text{e}}$ nombre de Fibonacci (Cours 4) ; le calcul du polynôme minimal d'une matrice creuse (Cours 27).
2. Un polynôme en deux variables peut être reconstruit à partir du développement en série d'une solution. L'efficacité de la résolution de systèmes polynomiaux par la méthode de la *résolution géométrique* abordée au cours 20 repose de manière cruciale sur cette opération, qui doit être effectuée rapidement.
3. Il est possible de reconstruire une équation différentielle linéaire à coefficients polynomiaux à partir du développement en série d'une solution et de bornes sur l'ordre et le degré des coefficients. De façon analogue, il est possible de reconstruire une récurrence linéaire à coefficients polynomiaux à partir des premiers termes d'une de ses solutions.

1.3. Équations comme structures de données. Une fois construits les objets de base que sont les polynômes, les séries ou les matrices, il est possible d'aborder des objets mathématiques construits *implicitement*. Ainsi, il est bien connu qu'il n'est pas possible de représenter toutes les solutions de polynômes de haut degré par radicaux, mais de nombreuses opérations sur ces solutions sont aisées en prenant le polynôme lui-même comme structure de données. Ce point de vue permet d'étendre le domaine d'application du calcul formel pourvu que des algorithmes soient disponibles pour effectuer les opérations souhaitées (typiquement addition, multiplication, multiplication par un scalaire, test d'égalité) par manipulation des équations elles-mêmes.

Nombres algébriques. C'est ainsi que l'on nomme les solutions de polynômes univariés. Les opérations d'addition et de multiplication peuvent être effectuées à l'aide de résultants (Cours 10). Ceux-ci peuvent être calculés efficacement à l'aide de séries (Cours 3). La division s'obtient par l'algorithme d'Euclide (Cours 10), et le test à

zéro se déduit du pgcd. Par exemple, il est possible de prouver assez facilement une identité comme

$$\frac{\sin\frac{2\pi}{7}}{\sin^2\frac{3\pi}{7}} - \frac{\sin\frac{\pi}{7}}{\sin^2\frac{2\pi}{7}} + \frac{\sin\frac{3\pi}{7}}{\sin^2\frac{\pi}{7}} = 2\sqrt{7}$$

une fois que l'on reconnaît qu'il s'agit d'une égalité entre nombres algébriques.

Systèmes polynomiaux. Vu l'importance des systèmes polynomiaux, une grande partie du cours leur sera consacrée (Cours 13 à 21). En considérant un système de polynômes, les questions naturelles qui peuvent être résolues sont l'existence de solutions, la dimension de l'espace des solutions (qui indique s'il s'agit d'une surface, d'une courbe, ou de points isolés), le degré, ou le calcul d'une paramétrisation de l'ensemble des solutions.

Il est également possible d'éliminer une ou des variables entre des polynômes. Cette opération peut s'interpréter géométriquement comme une projection. Dans le cas le plus simple, elle permet de calculer un polynôme s'annulant sur les abscisses des intersections de deux courbes. Une autre application est l'implicitisation, qui permet par exemple de calculer une équation pour une courbe donnée sous forme paramétrée.

Équations différentielles linéaires. Cette structure de données permet de représenter de nombreuses fonctions usuelles (exponentielle, fonctions trigonométriques et trigonométriques hyperboliques, leurs réciproques) ainsi que de nombreuses fonctions spéciales de la physique mathématique (fonctions de Bessel, de Struve, d'Anger, . . ., fonctions hypergéométriques et hypergéométriques généralisées), ainsi bien sûr que de multiples fonctions auxquelles n'est pas attaché un nom classique. Les opérations d'addition et de produit sont effectuées par des variantes noncommutatives du résultant qui se ramènent à de l'algèbre linéaire élémentaire (Cours 12). Le test à zéro se ramène à tester l'égalité d'un nombre fini de conditions initiales.

Ainsi, des identités élémentaires comme $\sin^2 x + \cos^2 x = 1$ sont non seulement facilement prouvables algorithmiquement, mais elles sont également calculables, c'est-à-dire que le membre droit se calcule à partir du membre gauche. Les relations étroites entre équations différentielles linéaires et récurrences linéaires — les séries solutions des unes ont pour coefficients les solutions des autres — amènent aux mêmes réponses algorithmiques à des questions sur des suites. Par exemple, l'identité de Cassini sur les nombres de Fibonacci

$$F_{n+2}F_n - F_{n+1}^2 = (-1)^{n+1}, \qquad n \geq 0$$

est exactement du même niveau de difficulté que $\sin^2 x + \cos^2 x = 1$.

Systèmes d'équations différentielles et de récurrences linéaires. Ces systèmes sont aux équations ce que les systèmes polynomiaux sont aux polynômes en une variable. Les mêmes opérations sont disponibles. En particulier, l'élimination s'étend dans ce cadre en introduisant des algèbres d'opérateurs adaptés (Cours 25). Une application très importante de cette élimination, la *création téléscopique*, permet de calculer

automatiquement des sommes et des intégrales définies. Ainsi,

$$\sum_{k=0}^{n}\left(\sum_{j=0}^{k}\binom{n}{j}\right)^3 = n2^{3n-1}+2^{3n}-3n2^{n-2}\binom{2n}{n},$$

$$\sum_{n=0}^{\infty} H_n(x)H_n(y)\frac{u^n}{n!} = \frac{\exp\left(\frac{4u(xy-u(x^2+y^2))}{1-4u^2}\right)}{\sqrt{1-u^2}},$$

$$\frac{1}{2}J_0(x)^2+J_1(x)^2+J_2(x)^2+\cdots=\frac{1}{2},$$

$$\int_{-1}^{+1}\frac{e^{-px}T_n(x)}{\sqrt{1-x^2}}\,dx=(-1)^n\pi I_n(p),$$

$$\int_0^{+\infty} xe^{-px^2}J_n(bx)I_n(cx)\,dx=\frac{1}{2p}\exp\left(\frac{c^2-b^2}{4p}\right)J_n\left(\frac{bc}{2p}\right),$$

$$\int_0^{+\infty} xJ_1(ax)I_1(ax)Y_0(x)K_0(x)\,dx=-\frac{\ln(1-a^4)}{2\pi a^2},$$

$$\sum_{k=0}^{n}\frac{q^{k^2}}{(q;q)_k(q;q)_{n-k}}=\sum_{k=-n}^{n}\frac{(-1)^kq^{(5k^2-k)/2}}{(q;q)_{n-k}(q;q)_{n+k}},$$

formules qui mettent en jeu diverses fonctions spéciales ou polynômes orthogonaux classiques, peuvent être prouvées automatiquement. Les algorithmes correspondants seront décrits au cours 26.

Ainsi, les exemples ci-dessus illustrent bien la manière dont le calcul formel parvient à effectuer de nombreux calculs utiles dans les applications malgré l'indécidabilité révélée par le théorème de Richardson.

2. Calculer rapidement

En pratique, la calculabilité n'indique que la faisabilité. Il faut disposer d'algorithmes efficaces et d'une bonne implantation pour pouvoir effectuer des calculs de grande taille. La première partie de ce cours (Cours 2 à 12) est consacrée aux algorithmes efficaces sur les structures de base du calcul formel. L'efficacité sera mesurée par la théorie de la complexité et nous ferons ressortir des principes récurrents dans la conception d'algorithmes efficaces.

Exemple 2. Pour donner une idée de ce que veut dire rapidement, voici ce qui peut être calculé en *une minute* avec le système Magma sur une machine de bureau d'aujourd'hui[1], en notant $\mathbb{K}$ le corps $\mathbb{Z}/p\mathbb{Z}$ à p éléments, $p = 67108879$ étant un nombre premier de 26 bits (dont le carré tient sur un mot machine) :

1. Entiers :
   - produit de deux entiers avec 200 000 000 de chiffres ;
   - factorielle de 4 000 000 (environ 25 000 000 de chiffres) ;
   - factorisation d'un entier de 45 chiffres (150 bits).
2. Polynômes dans $\mathbb{K}[x]$:

[1] Ce texte est écrit en 2006. La machine a un processeur AMD 64 à 2,2 GHz et une mémoire de 2 Go ; le système d'exploitation est linux.

- produit de deux polynômes de degré 8 000 000 (plus d'un an avec la méthode naïve) ;
- pgcd et résultant de deux polynômes de degré 200 000 ;
- factorisation d'un polynôme de degré 2 000.

3. Polynômes dans $\mathbb{K}[x, y]$:
   - résultant de deux polynômes de degré total 100 (sortie de degré 10 000) ;
   - produit et somme de deux nombres algébriques de degré 450 (sortie de degré 200 000) ;
   - factorisation d'un polynôme de degré 500 en deux variables.
4. Matrices :
   - déterminant d'une matrice $3\,500 \times 3\,500$ à coefficients dans $\mathbb{K}$;
   - polynôme caractéristique d'une matrice $2\,000 \times 2\,000$ à coefficients dans $\mathbb{K}$;
   - déterminant d'une matrice 700×700 dont les coefficients sont des entiers 32 bits.

Ces exemples montrent qu'il est relativement aisé de calculer avec des objets de taille colossale (mais pas avec les algorithmes naïfs), et donnent envie d'une mesure de complexité des différents algorithmes permettant d'expliquer, voire de prédire, les différences de tailles atteintes pour ces questions.

2.1. Mesures de complexité. Pour bien définir la complexité, il faut se donner : un modèle de machine ; les opérations disponibles sur cette machine ; leur coût unitaire. La complexité en espace mesure la mémoire utilisée par l'exécution de l'algorithme, et la complexité en temps, la somme des coûts unitaires des opérations effectuées par l'algorithme. Dans ce cours, nous ne nous intéresserons pas à la complexité en espace.

Machine RAM. Le modèle que nous utiliserons est celui de la *Random Access Machine* (RAM). Dans ce modèle, un programme lit et écrit des entiers sur deux bandes différentes et utilise un nombre arbitraire de registres entiers pour ses calculs intermédiaires. Les opérations élémentaires (l'assembleur de la machine) sont la lecture, l'écriture (sur bande ou en registre), l'addition, la soustraction, le produit, la division et trois instructions de saut : saut inconditionnel, saut si un registre est nul et saut si un registre est positif. Un point technique est que le programme ne fait pas partie des données, il n'est donc pas modifiable.

Complexité binaire ou arithmétique. Nous considérerons deux mesures de complexité :

1. Dans la complexité *binaire*, les bandes d'entrée et de sortie ainsi que les registres ne peuvent stocker que des *bits* (0 ou 1). La mesure de complexité des algorithmes opérant sur une telle machine tient compte de la taille des entiers manipulés et modélise précisément le temps de calcul.
2. Dans la complexité *arithmétique*, les opérations sur les entiers ont coût unitaire. Cette mesure modélise précisément le temps de calcul pour des calculs sur des entiers modulaires ou sur les flottants machine. Nous étendrons cette mesure au cas où les objets manipulés ne sont pas des entiers, mais plus

généralement des éléments d'un corps k donné et nous mesurerons alors la complexité en nombre d'opérations arithmétique dans k.

EXEMPLE 3. Le calcul de $n!$ par la méthode naïve requiert n opérations arithmétiques et $O(n^2 \log^2 n)$ opérations binaires. Nous verrons au cours 4 qu'il est possible d'abaisser ce coût à seulement $O(n^{1/2} \log n)$ opérations arithmétiques, et $O(n \log^3 n)$ opérations binaires. Les algorithmes rapides permettant d'atteindre ces complexités fournissent le meilleur algorithme connu de factorisation déterministe d'entiers et des algorithmes très efficaces pour le calcul de millions de décimales de π, $\log 2$ et de nombreuses autres constantes.

Taille. Un algorithme et une structure de données sont généralement dotés d'une notion naturelle de taille et il s'agit d'étudier le coût de l'algorithme en fonction de cette taille. Pour simplifier, il est souvent commode de considérer le comportement asymptotique de ce coût lorsque la taille tend vers l'infini. Il est important de comprendre que la complexité d'un problème n'a de sens qu'une fois la structure de donnée fixée pour l'entrée comme pour la sortie.

Par exemple, pour les polynômes, le choix de la représentation dense mène à mesurer la complexité par rapport au degré, alors que le choix de la représentation creuse met en avant le nombre de monômes. Pour la factorisation, la complexité est polynomiale en le degré, mais exponentielle en le nombre de monômes, dans le cas le pire.

Cas le pire, cas moyen. La complexité dans le cas le pire est le maximum des complexités pour toutes les entrées d'une taille donnée. C'est celle que nous étudierons. Il est souvent utile de considérer aussi la complexité en moyenne, lorsque l'on peut mettre une mesure sur l'ensemble des entrées de taille bornée. Pour la plupart des algorithmes que nous étudierons dans la première partie de ce cours, il n'y a pas de différence importante entre les deux. Ce n'est plus le cas en revanche pour la complexité des algorithmes sur les systèmes polynomiaux (où la notion de complexité en moyenne est avantageusement remplacée par celle de complexité dans le cas générique).

Bornes inférieures. La recherche de bornes inférieures de complexité est très difficile. Par exemple, à l'heure actuelle on ne sait pas prouver que la multiplication de matrices est nécessairement plus coûteuse qu'un nombre borné d'additions. Dès qu'il est possible de montrer que tous les bits de l'entrée doivent être pris en compte, la somme de la taille de l'entrée et de la taille de la sortie est une borne inférieure sur la complexité. En effet, dans le modèle RAM, chacune des écritures et des lectures prend une opération. Si N est cette borne inférieure, l'algorithme sera dit *quasi-optimal* lorsque sa complexité sera bornée par $O(N \log^k N)$ pour un $k \geq 0$ arbitraire. L'essentiel de la première partie du cours consistera à rechercher des algorithmes quasi-optimaux pour les opérations de base sur les structures de données fondamentales.

2.2. La notation $O(\cdot)$. Nous utilisons la notation $O(\cdot)$ pour exprimer une borne sur la complexité des algorithmes. La signification précise de la notation

$$f(n) = O(g(n)), \qquad n \to \infty$$

est qu'il existe $K > 0$ et $A > 0$ tels que pour tout $n > A$, f et g soient liés par l'inégalité

$$|f(n)| \leq K|g(n)|.$$

Lorsque plusieurs paramètres interviennent dans l'analyse de la complexité, il faut absolument préciser lequel tend vers l'infini pour que cette notation ait un sens. Si plusieurs paramètres tendent vers l'infini, soit ils sont liés par des inégalités qui seront précisées, soit la définition ci-dessus s'étend avec une constante K qui ne dépend d'aucun des paramètres.

La notation $O(\cdot)$ intervient aussi dans ce cours pour représenter la troncature des séries. L'expression

$$f(x) := g(x) + O(x^N)$$

signifiera que le polynôme ou la série g est tronqué après son N^{e} terme et que le résultat, un polynôme, est stocké dans f.

2.3. Diviser pour régner. Le principe le plus important de la conception d'algorithmes efficaces est le paradigme « diviser pour régner ». Il consiste à résoudre un problème en le réduisant à un certain nombre m d'entrées de taille divisées par p (le plus souvent $p = 2$) puis à recombiner les résultats. Le coût de la recombinaison et éventuellement du découpage préliminaire est borné par une fonction f de la taille des entrées. Le coût total dépend de la croissance de f par rapport à N. Un cadre commode pour nos applications est résumé dans le lemme suivant.

Lemme 1 (« Diviser pour régner »). *S'il existe $q > 1$ tel que pour tout réel $x \geq 1$, la relation $f(x) \geq qf(x/p)$ soit vérifiée, alors pour tout $m \geq 1$ et $x \geq 1$, toute solution de l'inégalité*

$$C(x) \leq f(x) + mC(x/p) \quad \textit{pour } x > p \qquad \textit{et} \qquad C(x) \leq c \quad \textit{pour } x \leq p$$

vérifie

$$C(x) \leq cm^{\lceil \log_p x \rceil} + \begin{cases} \frac{1}{1-\frac{m}{q}} f(x) & \textit{si } q > m, \\ f(x)\lceil \log_p x \rceil & \textit{si } q = m, \\ \frac{x^{\log_p (m/q)}}{1-\frac{q}{m}} f(x) & \textit{si } q < m. \end{cases}$$

La notation $\lceil x \rceil$ désigne l'entier k tel que $k - 1 < x \leq k$, et $\log_p x$ représente le logarithme en base p de x, c'est-à-dire $\log x / \log p$.

En pratique, la fonction f représente le coût d'un algorithme et n'est souvent définie que pour des arguments entiers. Dans ce cas, la complexité de l'algorithme appliqué à une entrée de taille x sera souvent bornée par la complexité de l'algorithme lorsqu'il est appliqué à des entrées de taille supérieure, et on prendra la puissance de 2 immédiatement supérieure pour pouvoir appliquer le lemme. Comme cette puissance est à moins d'un facteur 2 de x, les estimations perdent au pire un facteur q.

DÉMONSTRATION. En appliquant plusieurs fois l'inégalité, il vient

$$\begin{aligned} C(x) &\leq f(x) + mC\left(\frac{x}{p}\right), \\ &\leq f(x) + mf\left(\frac{x}{p}\right) + m^2 C\left(\frac{x^2}{p^2}\right), \\ &\leq f(x) + mf\left(\frac{x}{p}\right) + m^2 f\left(\frac{x}{p^2}\right) + \cdots + m^{\lceil \log_p x \rceil} C\left(\frac{x}{p^{\lfloor \log_p x \rfloor}}\right). \end{aligned}$$

L'inégalité vérifiée par f entraîne par récurrence

$$f\left(\frac{x}{p^k}\right) \leq q^{-k} f(x).$$

La somme considérée est donc majorée par

$$cm^{\lceil \log_p x \rceil} + f(x)\left(1 + \frac{m}{q} + \left(\frac{m}{q}\right)^2 + \cdots + \left(\frac{m}{q}\right)^{\lceil \log_p x \rceil - 1}\right).$$

Soit S la somme entre parenthèses. Si $q > m$, la série géométrique S est convergente, d'où le résultat. Si $q = m$, tous les termes de S sont égaux à un et leur nombre est $\lceil \log_p x \rceil$. Enfin, si $q < m$, on se ramène encore à une série géométrique en réécrivant S sous la forme

$$\left(\frac{m}{q}\right)^{\lceil \log_p x \rceil - 1}\left(1 + \frac{q}{m} + \left(\frac{q}{m}\right)^2 + \cdots + \left(\frac{q}{m}\right)^{\lceil \log_p x \rceil - 1}\right).$$

□

EXEMPLE 4. Pour trier un tableau de N éléments, le tri fusion utilise un « diviser pour régner » : chaque moitié du tableau est triée récursivement, donc $p = m = 2$, et les deux moitiés sont ensuite fusionnées en au plus N comparaisons, donc $q = 2$. Si N est une puissance de 2 $\lceil \log_2 N \rceil = \log_2 N$ et la complexité est donc d'au plus $N \log N + N$ comparaisons pour trier une liste de taille N.

EXEMPLE 5. La complexité de l'algorithme de Karatsuba du cours 2 s'obtient avec $m = 3$, $p = 2$ et f linéaire (donc $q = p$).

3. Organisation du cours

Le diagramme de la Figure 1 montre les dépendances entre les cours.(Un cours est au-dessous d'un autre lorsqu'il utilise des notions qui y ont été présentées.)

Notes

Les références générales sur les algorithmes du calcul formel sont deux livres : celui de von zur Gathen et Gerhard [10] et celui, plus élémentaire, de Geddes, Czapor et Labahn [5]. La complexité est également utilisée comme fil conducteur dans le livre plus difficile de Bürgisser, Clausen et Shokrollahi [3]. Une bonne introduction à la seconde partie du cours, sur les systèmes polynomiaux, est le livre de Cox, Little et O'Shea [4]. De premiers éléments pour aborder la partie 3, sur les équations différentielles et les récurrences linéaires, sont donnés dans le livre $A = B$ de Petkovšek, Wilf et Zeilberger [7].

Le théorème de Richardson [8] s'applique à des fonctions. Pour des constantes, l'approche la plus récente [9] réduit le test à zéro à une conjecture de théorie des

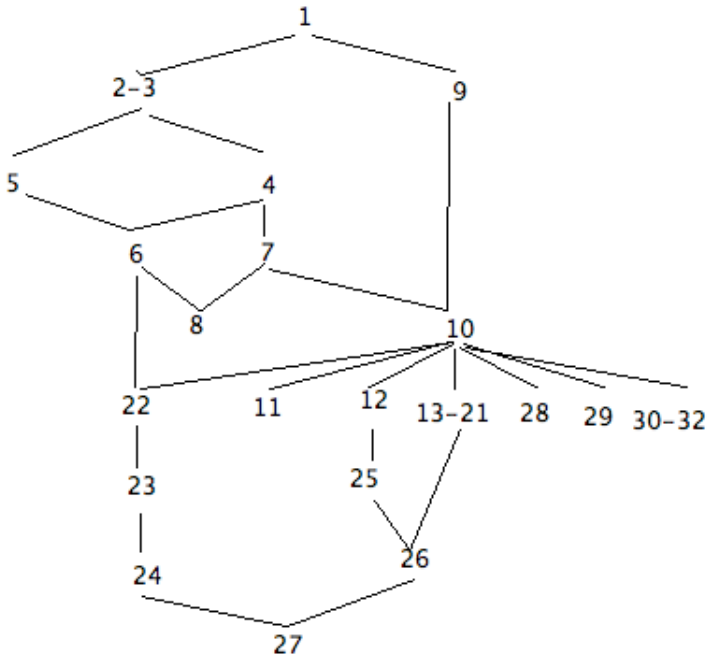

Fig. 1.

nombres due à Schanuell qui exprime que les seules relations entre exponentielles et logarithmes sont celles qui découlent des formules d'addition et de multiplication.

L'implantation d'une arithmétique efficace pour les entiers longs (bignums) est un travail très délicat. Une des meilleures arithmétiques disponibles est fournie par GMP, le *Gnu Multiprecision Package* [6]. Elle est le résultat d'un travail de nombreuses années, qui comporte une partie importante de code assembleur consacré à la multiplication sur chacun des processeurs produits dans une période récente. Les entiers de GMP sont ceux qui sont utilisés dans Maple pour les grandes tailles. D'autres entiers très efficaces sont implantés dans le système Magma.

Les différents modèles de complexité (machine RAM, machine de Turing, *straight-line program*, ...) sont bien présentés par Aho, Hopcrof et Ulmann dans [1].

Bibliographie

[1] Aho (Alfred V.), Hopcroft (John E.), and Ullman (Jeffrey D.). – *The design and analysis of computer algorithms*. – Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1974, x+470p. Addison-Wesley Series in Computer Science and Information Processing.

[2] Beck (Matthias), Berndt (Bruce C.), Chan (O-Yeat), and Zaharescu (Alexandru). – Determinations of analogues of Gauss sums and other trigonometric sums. *International Journal of Number Theory*, vol. 1, n° 3, 2005, pp. 333–356.

[3] Bürgisser (Peter), Clausen (Michael), and Shokrollahi (M. Amin). – *Algebraic complexity theory*. – Springer-Verlag, Berlin, 1997, *Grundlehren der Mathematischen Wissenschaften*, vol. 315, xxiv+618p.

[4] Cox (David), Little (John), and O'Shea (Donal). – *Ideals, varieties, and algorithms*. – Springer-Verlag, New York, 1996, second edition, xiv+536p.

[5] Geddes (Keith O.), Czapor (Stephen R.), and Labahn (George). – *Algorithms for Computer Algebra*. – Kluwer Academic Publishers, 1992.

[6] Granlund (Torbjörn). – *GNU Multiple Precision Arithmetic Library*. – http://swox.com/gmp, 2006.

[7] Petkovšek (Marko), Wilf (Herbert S.), and Zeilberger (Doron). – $A = B$. – A. K. Peters, Wellesley, MA, 1996, xii+212p.

[8] Richardson (Daniel). – Some undecidable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, vol. 33, n° 4, 1968, pp. 514–520.

[9] Richardson (Daniel). – How to recognize zero. *Journal of Symbolic Computation*, vol. 24, n° 6, 1997, pp. 627–645.

[10] von zur Gathen (Joachim) and Gerhard (Jürgen). – *Modern computer algebra*. – Cambridge University Press, New York, 1999, xiv+753p.