UFR Sciences
M2 Calcul Formel

Martin WEIMANN

Complexité et algorithmes élémentaires

Références:

- Bostan, Chyzak, Giusti, Lebreton, Lecerf, Salvy, Schost, *Algorithmes efficaces en Calcul Formel* (2017).
- Demazure, *Cours d'algèbre*, Cassini (1997).
- Foissy, Ninet, *Algèbre et Calcul Formel Agrégation de Mathématiques Option C* (2017).
- Saut Picard, *Cours de calcul formel: Algorithmes fondamentaux* (1999).
- Gathen and Gerhard, *Modern Computer Algebra*, Cambridge University Press (2003).
- Page de K. Belabas, *https://www.math.u-bordeaux.fr/ kbelabas/teach/Agreg/*.

1 Complexité d'un algorithme

La complexité d'un algorithme dépend de la taille de l'entrée (ou donnée). La taille de l'entrée est en général mesurée par un entier[1] $n \in \mathbb{N}$. Il ne faut pas confondre la complexité d'un algorithme avec la complexité du problème à résoudre, qui est par définition l'inf des complexités de tous les algorithmes résolvant ce problème. Par exemple, l'algorithme du pivot de Gauss a une complexité cubique, mais il existe des algorithmes plus rapides pour trigonaliser une matrice.

Complexité en temps, complexité en espace

- La complexité en temps d'un algorithme est une fonction de n qui mesure le temps de calcul pour une donnée de taille n.
- La complexité en espace (ou en mémoire) d'un algorithme est une fonction de n qui mesure l'espace mémoire utilisé lors des calculs sur une donnée de taille n.

Complexité du pire, complexité en moyenne

- La complexité du pire des cas donne une borne supérieure pour toutes les données de taille n. C'est le type d'estimation qui nous intéressera le plus souvent ici (on calculera également des temps de calcul expérimentaux en TP).
- La complexité en moyenne donne une moyenne sur l'ensemble des données de taille n (muni d'une mesure de probabilité spécifiée).

[1] Voire par une liste d'entiers. Par exemple, on peut mesurer la taille d'une matrice de taille $m \times p$ par l'entier $n = mp$ ou bien par le couple (m, p), plus précis en fonction du problème à résoudre.

Complexité binaire, complexité arithmétique

La complexité en temps est essentiellement le nombre d'opérations élémentaires multiplié par le temps d'une opération élémentaire (que l'on suppose traitée en temps constant). Pour avoir un indicateur indépendant de la machine utilisée, on estime en général le nombre d'opérations élémentaires, exprimé en fonction de la taille de l'entrée. Bien sûr, il est important de spécifier ce que l'on entend par taille de l'entrée et par opération élémentaire.

- La *taille* n d'un entier $a \in \mathbb{Z}$ est le nombre de bits nécessaires à son écriture binaire, i.e. $n = \lfloor \log_2(a) + 1 \rfloor + 1$ (le $+1$ pour le signe). Dans ce cas, une opération élémentaire sur a est une opération sur un bit. On parle de *complexité binaire*.
- Soit K un corps ou un anneau. La taille (arithmétique) de $P \in K[X]$ est $n = \deg(P) + 1$ et la taille d'une matrice $M \in \mathcal{M}_{m,p}(K)$ est $n = m \times p$. Une opération élémentaire est dans ce cas une opération arithmétique $+, -, \times, \div$ dans K (pas de division si K est un anneau)[2]. On parle de *complexité arithmétique* (ou *algébrique* chez certains auteurs). Pour estimer la complexité binaire, il faut prendre en compte également la taille des coefficients. Dans le cas important $K = \mathbb{Z}/N\mathbb{Z}$, on verra qu'il faut essentiellement multiplier par $\log(N)$ pour obtenir une complexité binaire (donc une complexité en temps réaliste). Si K est de cardinal infini, la taille des coefficients traités peut croître de manière exponentielle lors des calculs auquel cas la complexité arithmétique ne reflète que partiellement la complexité en temps. Cependant, estimer la complexité binaire dans $K[X]$ est en général très difficile, et on ne s'y penchera pas vraiment dans ce cours.

Anneaux et corps effectifs.

En calcul formel, on ne manipule les objets que de manière exacte, ou symbolique. Une structure algébrique (groupe, anneau, corps, etc.) est dite *effective* si on dispose:

- d'une structure de données pour en représenter les éléments ;
- d'algorithmes pour en effectuer les opérations et pour y tester l'égalité et autres prédicats.

Les anneaux $\mathbb{Z}$ et $\mathbb{Z}/N\mathbb{Z}$ (en particulier $\mathbb{F}_p$) sont effectifs. Le corps de fraction d'un anneau intègre est effectif. En particulier, $\mathbb{Q}$ est effectif. Si A est un anneau effectif, alors A^n, $\mathcal{M}_n(A)$ et $A[X_1, \ldots, X_n]$ sont effectifs, leurs éléments se représentant sous forme de listes. Si K est un corps effectif, alors les K-algèbres $K[X_1, \ldots, X_n]/(f_1, \ldots, f_r)$ sont effectives. En particulier, une extension algébrique d'un corps effectif est effective, par exemple $\mathbb{Q}(i)$, $\mathbb{Q}(\sqrt{2})$ et $\mathbb{F}_q$ sont effectifs. Par contre $\mathbb{R}$ ou $\mathbb{C}$ ne sont pas effectifs : ils nécessitent d'utiliser le calcul flottant pour représenter un réel, et on perd le test à zéro. On supposera toujours dans ce cours que l'anneau ou le corps de base est effectif.

Complexité asymptotique.

Il est difficile en général d'estimer la complexité exacte d'un algorithme. On préfère estimer la complexité asymptotique (pour $n \to +\infty$) : on s'intéresse essentiellement au terme dominant et on oublie en général la constante devant ce terme. Les notations de Landau "grand O" et "petit o" sont très pratiques.

Définition 1 Soient f et g deux fonctions de n.

- On dit que f est dominée par g, noté $f = O(g)$ (f est un "grand O" de g), s'il existe $n_0 \in \mathbb{N}$ et $c \in \mathbb{R}$ tels que
$$\forall\, n \geq n_0, \quad |f(n)| \leq c g(n).$$

[2] On néglige en général les tests d'égalité ou d'inégalités, les affectations, etc.

- On dit que f est négligeable devant g, noté $f = o(g)$ (f est un "petit o" de g), si

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0.$$

- On dit que f domine g, noté $f = \Omega(g)$ si $g = O(f)$.
- On dit que f est comparable à g, noté $f = \Theta(g)$, si $f = O(g)$ et $g = O(f)$. On a alors bien évidemment aussi $g = \Theta(f)$. Rem : f et g sont alors positives pour n grand.
- On dit que f est équivalente à g, noté $f \sim g$, si

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 1.$$

Définition 2 On dit que la complexité f d'un algorithme est (par ordre croissant):

- *logarithmique* si $f(n) = O(\log n)$ (cela entraîne qu'on ne lit pas toutes les données d'entrées qui sont en $O(n)$);
- *linéaire* si $f(n) = O(n)$;
- *quasi-linéaire* si $f(n) = O(n(\log n)^\alpha)$ pour un certain $\alpha \geq 0$, noté $f(n) = \tilde{O}(n)$;
- *quadratique* si $f(n) = O(n^2)$ et cubique si $f(n) = O(n^3)$;
- *polynomiale* si $f(n) = 2^{O(\log(n))}$, i.e. s'il existe $\alpha > 0$ (indépendant de n) tel que $f(n) = O(n^\alpha)$;
- *sous-exponentielle* si $f(n) = 2^{o(n)}$;
- *exponentielle* si $f(n) = 2^{O(n)}$.

Remarque 1 Soit g fonction de la variable n. Pour montrer que la complexité d'un algorithme est en $O(g)$ (problème de majoration), il faut montrer que c'est vrai pour toutes les (suites de) données de taille n. Pour montrer que c'est en $\Omega(g)$ (problème de minoration), il suffit de trouver une (suite de) donnée de taille n où il se comporte ainsi.

Diviser pour régner

Ce principe est bien souvent très efficace. Il consiste à réduire récursivement un problème de taille n en m sous-problèmes de tailles divisées par une constante p (le plus souvent $p = 2$) puis à recombiner les résultats. Un exemple classique est le tri fusion. Le coût de la recombinaison (et éventuellement du découpage préliminaire) est borné par une fonction T de la taille des entrées. Lorsque les entrées ont une taille inférieure à p, un autre algorithme de coût constant c (indépendant de n) est invoqué. Le coût total obéit alors souvent à une récurrence de la forme

$$C(n) \leq \begin{cases} T(n) + mC\left(\lfloor n/p \rfloor\right) & \text{si} \quad n \geq p \\ c \quad \text{sinon} \end{cases} \tag{1}$$

Le coût total dépend alors fortement du coût T des recombinaisons. On a le résultat général suivant (qui n'est pas à connaître par coeur).

Lemme 1 Soit C une fonction vérifiant (1). On suppose T croissante et telle qu'il existe $1 \leq q \leq r$ pour lesquels

$$qT(n/p) \leq T(n) \leq rT(n/p) \quad \forall\ n >> 0. \tag{2}$$

On a alors

$$C(n) = \begin{cases} O(T(n)) & \text{si} \quad q > m \\ O(T(n)\log(n)) & \text{si} \quad q = m \\ O(T(n)n^{\log_p(m/q)}) & \text{si} \quad q < m \end{cases} \tag{3}$$

On notera que l'hypothèse (2) est vérifiée en particulier pour les fonctions usuelles de la forme $T(n) = n^\alpha \log^\beta(n)$ avec $\alpha > 0$.

Exercices Section 1.

Exercice 1 Justifier que $\mathbb{Z}$ et $\mathbb{Z}/N\mathbb{Z}$ sont des anneaux effectifs puis justifier que les corps finis $\mathbb{F}_q$ sont effectifs.

Exercice 2

1. Montrer que si $f = O(cg)$ pour $c > 0$, alors $f = O(g)$.
2. Montrer que si $f = \Theta(cg)$ pour $c > 0$, alors $f = \Theta(g)$.
3. Montrer que $f \sim g$ si et seulement si $f = g + o(g)$.
4. Suppose g positive. Montrer que si $f \sim g$ avec g positive, alors $f = \Theta(g)$.

Exercice 3

1. Soit $P \in \mathbb{R}[x]$ un polynôme de degré k. Montrer que $|P(n)| = \Theta(n^k)$.
2. Soit $k, n \in \mathbb{N}$. Montrer que $\sum_{i=0}^{n} i^k = \Theta(n^{k+1})$ (utiliser une récurrence et considérer $\sum_{i=1}^{n+1} i^{k+1}$).
3. Montrer que $\log(n!) = \Theta(n\log(n))$ (considérer les $n/2$ derniers facteurs pour la minoration)

Exercice 4 Montrer que l'algorithme naïf de la multiplication de deux matrices carrées de taille $n \times n$ est $O(n^3)$. Il existe cependant des algorithmes plus rapides en $O(n^{2,3})$. C'est un problème ouvert important de connaître la complexité de ce problème (c'est au moins $O(n^2)$, taille de la sortie).

2 Exponentiation, addition, multiplication

2.1 Exponentiation rapide

C'est probablement l'opération la plus fondamentale en calcul formel puisqu'elle concerne les groupes. Soit A un groupe (que l'on suppose multiplicatif pour simplifier) et $x \in A$. Le calcul naïf de l'exponentiation x^n requiert $n-1$ multiplications dans A. Fort heureusement, une méthode récursive "diviser pour régner" permet d'accélérer considérablement ce calcul (fondamental en calcul formel).

Théorème 1 (Exponentiation rapide) Soit A un anneau et $x \in A$. On peut calculer x^n avec $O(\log(n))$ multiplications dans A.

Exercice 5 Ecrire en pseudo-code un algorithme récursif d'exponentiation rapide.

2.2 Addition et multiplication naïves

Théorème 2 (Opérations sur les entiers) Soient a et b deux entiers de taille au plus n (donc n bits au plus pour écrire a et b en écriture binaire). On peut calculer :

1. $a+b$ et $a-b$ en $O(n)$ opérations binaires.
2. $a \times b$ en $O(n^2)$ opérations binaires.

Théorème 3 (Opérations sur les polynômes) Soit K un corps et $a, b \in K[X]$ deux polynômes de degrés au plus n. On peut calculer :

1. $a+b$ et $a-b$ en $O(n)$ opérations arithmétiques dans K
2. $a \times b$ en $O(n^2)$ opérations arithmétiques dans K.

Remarque 2 L'addition des polynômes est élémentaire (addition terme à terme). L'addition des entiers (écrits en base 2) est plus délicate du fait de l'existence de retenues. Ce principe général rend les algorithmes sur $\mathbb{Z}$ plus délicats que sur $K[X]$ quoique souvent basés sur des méthodes analogues.

Exercice 6 Montrer que le nombre d'additions et multiplications dans K nécessaires pour multiplier deux polynôme de degrés respectifs n et m par l'algorithme de multiplication naïf est précisément $2mn+m+n+1$, donc une complexité plus précise $O(mn)$. Cette précision est bien sûr importante si $m << n$.

2.3 Multiplication de Karatsuba : diviser pour régner

C'est un algorithme récursif de multiplication dans $K[X]$ de complexité sous-quadratique, basé sur le principe "diviser pour régner" : on découpe un problème de taille n en sous problèmes de taille $n/2$ que l'on traite récursivement.

Soient à multiplier deux polynômes a et b de degrés au plus $n-1$ avec $n = 2^k$ (pour simplifier). On écrit alors

$$a = a_0 + a_1 X^{n/2}, \quad b = b_0 + b_1 X^{n/2} \tag{4}$$

où $a_0, a_1, b_0, b_1 \in K[X]$ sont de degré au plus $n/2 - 1 = 2^{k-1} - 1$, de sorte que

$$ab = a_0 b_0 + (a_0 b_1 + a_1 b_0) X^{n/2} + a_1 b_1 X^n.$$

Le principe de l'algorithme de Karatsuba consiste à calculer $a_0 b_0$ et $a_1 b_1$ (récursivement), puis à utiliser ensuite l'égalité

$$a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1.$$

On perd 3 additions par rapport au calcul naïf de $a_0 b_1 + a_1 b_0$, mais le gain d'une multiplication va au final entraîner une amélioration de la complexité.

Théorème 4 L'algorithme de Karatsuba permet de calculer le produit de deux polynômes dans $K[X]$ de degré au plus $n-1$ avec $O(n^{\log_2(3)}) \approx O(n^{1.58})$ opérations dans K.

On obtient donc une complexité sous-quadratique. Pour donner une idée, pour $n = 1000$, on a $n^{\log_2(3)} \approx 50\,000$ alors que la multiplication naïve requiert environ $n^2 = 1\,000\,000$ opérations.

Exercice 7 Ecrire en pseudo-code l'algorithme de Karatsuba et montrer que son coût $C(n)$ vérifie l'inégalité $C(n) \leq 4n + 3C(n/2)$. En déduire l'estimation $C(n) = O(n^{\log_2(3)})$ en utilisant la fonction auxiliaire $D(q) = C(2^q)/3^q$. Retrouver ce résultat à l'aide du Lemme 1.

Remarque 3 Lorsque $n-1$ n'est pas de la forme 2^k, on complète les polynômes avec des coefficients nuls pour se ramener au cas précédent. Puisque le premier entier plus grand que $n-1$ de la forme 2^k est au plus égal à $2(n-2)$, la complexité est au plus multipliée par 2, donc reste en $O(n^{1.59})$.

Remarque 4 L'algorithme de Karatsuba se généralise dans $\mathbb{Z}$ en écrivant les entiers en base 2. Il faut cependant gérer cette fois les retenues.

2.4 Multiplication rapide (quasi-linéaire)

Théorème 5 (Multiplication rapide) Il existe des algorithmes de multiplication rapides dans $\mathbb{Z}$ et $K[X]$ de complexité quasi-linéaire $\tilde{O}(n)$.

Ces algorithmes ont donc une complexité quasi-optimale en la taille de l'entrée et la sortie (sous quelques hypothèses raisonnables sur K). Ils sont basés sur le principe d'évaluation/interpolation (pour les polynômes), sur la transformation de Fourier rapide, et sur des approches récursive "diviser pour régner". Voir par exemple [BCGLLSS, Section 2.4]. Certains textes de l'agrégation de maths portent sur ce sujet.

Remarque 5 Un algorithme récent de multiplication rapide dans $\mathbb{Z}$ (Harley et van der Hoeven, 2019) a atteint la borne théorique optimale $O(n \log n)$. C'est un résultat théorique remarquable mais pour l'instant peu utilisable en pratique (il faudrait traiter des entrées de tailles colossales pour qu'ils aient un intérêt pratique).

2.5 Fonctions de multiplication

Pour garder de la souplesse sur le choix de l'algorithme de multiplication utilisé dans un anneau A effectif (e.g. $A = \mathbb{Z}$ ou $A = K[X]$), on utilise le concept de *fonction de multiplication*. Une telle fonction $\mathrm{M}_A : A \to A$ doit vérifier :

- On peut multiplier deux éléments de A de taille au plus n avec $\mathrm{M}_A(n)$ opérations (binaires si $A = \mathbb{Z}$ ou arithmétiques si $A = K[X]$).
- On a $\mathrm{M}_A(n) + \mathrm{M}_A(m) \leq \mathrm{M}_A(n+m)$ pour tout $n, m \in \mathbb{N}$. En particulier, $k\,\mathrm{M}_A(n) \leq \mathrm{M}_A(kn)$.

Cette seconde condition permet d'avoir un contrôle sur le coût des algorithmes récursifs. Elle impose que M_A croît au moins linéairement.

3 Division euclidienne

3.1 Cas des polynômes

Soit K un corps et $a, b \in K[X]$ deux polynômes. Effectuer la division euclidienne de a par b consiste à calculer $q, r \in K[X]$ tels que $a = qb + r$ et $\deg(r) < \deg(b)$. Ce couple est unique (exo).

Théorème 6 Soit $a, b \in K[X]$ de degrés respectifs n et m, avec $n \geq m$ et le coefficient dominant de b non nul. La division euclidienne de a par b selon les puissances décroissantes effectue $(2m+1)(n-m+1)$ opérations $(+, -, \times)$ et une division dans K, soit $O(m(n-m))$ opérations arithmétiques (au plus quadratique en la taille de l'entrée).

Exercice 8 Dérouler l'algorithme classique de division euclidienne avec $a = X^4 + 3X^3 + X^2 - 2X$ et $b = X^2 - X + 1$.

Exercice 9 Ecrire l'algorithme en question et prouver le théorème 6.

Exercice 10 Justifier que si A est un anneau et si le coefficient dominant de $b \in A[X]$ est inversible, alors la division euclidienne par b est encore définie dans $A[X]$.

3.2 Cas des entiers

Soient $a, b \in \mathbb{Z}$. Il existe un unique couple $q, r \in \mathbb{Z}$ tels que $a = qb + r$ avec $r < b$. On appelle division euclidienne de a par b le calcul de q, r.

Remarque 6 On note souvent $q = \text{Quo}(a, b)$ et $r = \text{Rem}(a, b)$ (ou $r = a \mod b$ par abus). On a aussi la notation plus compacte $(q, r) = \text{QuoRem}(a, b)$.

L'algorithme suivant est celui de la division euclidienne apprise à l'école :

Entrée: $a, b \in \mathbb{N}$, avec $b \neq 0$.
Sortie: $q, r \in \mathbb{N}$ tels que $a = qb + r$, $r < b$.

- $q = 0$, $r = a$;
- Tant que $r > b$ faire:
  - Trouver le plus grand entier n tq $10^n b \leq r$
  - Trouver le plus grand chiffre c tq $10^n cb \leq r$
  - $q = q + 10^n c$
  - $r = r - 10^n cb$
- Retourner q, r.

Exercice 11 Dérouler l'algorithme de division avec $a = 2538$ et $b = 32$, en posant en parallèle la division scolaire.

Théorème 7 Soient $a, b \in \mathbb{Z}$ de tailles respectives n et m, avec $n \geq m$. L'algorithme de division euclidienne ci-dessus calcule $(q, r) = \text{QuoRem}(a, b)$ avec $O(m(n - m))$ opérations binaires (donc quadratique en la taille de l'entrée).

Preuve. Bien que l'idée de l'algorithme et la complexité soit analogue au cas des polynômes, l'existence des retenues complique les choses.

L'algo est correct. Notons $q_0 = 0$, $r_0 = a$ et notons avec un indice i les entiers n, c, r, q calculés à la i-ème itération. Ainsi, on a pour $i \geq 1$

$$q_i = q_{i-1} + 10^{n_i} c_i, \qquad r_i = r_{i-1} - 10^{n_i} c_i b$$

d'où l'on déduit $r_{i-1} + q_{i-1} b = r_i + q_i b$. Il s'ensuit par induction que $a = r_0 + q_0 b = r_i + q_i b$ pour tout i. Comme l'algo termine à un certain indice t (cf ci-dessous) – i.e. $r_t < b$ –, on obtient bien $(q_t, r_t) = \text{QuoRem}(a, b)$.

L'algo termine. Montrons que l'algo termine après un nombre logarithmique d'itérations. On considère pour cela la suite des n_i, qui par construction valent $n_i = \lfloor \log_{10}(r_{i-1}/b) \rfloor$ $(i \geq 1)$. On a

$$r_i = r_{i-1} - 10^{n_i} b c_i \quad \text{avec} \quad 10^{n_i} b(c_i + 1) > r_{i-1},$$

(la seconde inégalité par définition de c_i), d'où l'on déduit $r_i < 10^{n_i} b$. En divisant par b et en passant au $\log_{10}$, on en déduit $\lfloor \log_{10}(r_i/b) \rfloor < n_i$, d'où il suit que $n_{i+1} < n_i$. Donc l'algorithme termine après au plus $n_1 = \lfloor \log_{10}(a/b) \rfloor$ itérations.

Complexité (esquisse, en représentation décimale). Quitte à précalculer tous les produits bc, avec $0 \leq c \leq 9$, le calcul de tous les c_i coûte $O(\log(b))$ (car c a un seul digit). Le calcul de n_i coûte $O(1)$ opération ($\approx$ connaître le nombre de digits de r_{i-1}). Le calcul de q_i est en temps constant $O(1)$ (second terme a un seul digit non nul) et le calcul de r_i coûte $O(\log(b))$ car $10^n bc$ a $\approx \log(b)$ digits non nuls (la gestion des retenues est cependant délicate). On obtient un coût de $O(\log(b))$ à chaque itération, soit une complexité totale de $O(\log(b)\log(q)) = O(m(n-m))$. □

Remarque 7 Plus généralement, la division euclidienne existe dans tout anneau euclidien, par exemple dans l'anneau des entiers de Gauss $\mathbb{Z}[i]$ muni du stathme $\phi(a+ib) = a^2 + b^2$ (on rappelle au passage que Euclidien ⇒ Principal ⇒ Factoriel). Le principe de l'algorithme de division classique est similaire au cas des entiers et des polynômes.

3.3 Division rapide

Tout comme la multiplication, la division euclidienne est un élément central du calcul formel, et de nombreux autres algorithmes ont été proposés. En utilisant l'itération de Newton (qui permet entre autre de calculer rapidement l'inverse d'un élément), on peut montrer que la division euclidienne se réduit essentiellement au coût d'une multiplication. Voir par exemple [BCGLLSS, Section 4.2].

Théorème 8 Soit $A = \mathbb{Z}$ ou $A = K[X]$. Il existe des algorithmes de division euclidienne rapide dans $\mathbb{Z}$ et $K[X]$, de complexité $O(\mathrm{M}_A(n))$ en la taille n de l'entrée.

En utilisant la multiplication rapide $\mathrm{M}_A(n) = \tilde{O}(n)$, on obtient ainsi des algorithmes de divisions quasi-optimaux (quasi-linéaires en la taille de l'entrée et la sortie).

3.4 Application à l'écriture en base b

Soit $a, b \in \mathbb{N}$ avec $b \geq 2$. L'entier n s'écrit de manière unique sous la forme

$$a = r_k b^k + \cdots + r_1 b + r_0$$

avec $k \in \mathbb{N}$, $0 \leq r_i < b$ et $r_k \neq 0$ (si $a < 0$, on stocke le signe séparément). C'est ce que l'on appelle *l'écriture en base* b de a, notée $(r_k, \ldots, r_0)_b$. Les coefficients s'obtiennent simplement comme la suite (inversée) des restes de divisions euclidiennes itérées:

Baseb(a, b):

- $(q_0, r_0) = \mathrm{QuoRem}(a, b)$
- $i = 0$
- Tant que $q_i > 0$, faire :
  - $(q_{i+1}, r_{i+1}) = \mathrm{QuoRem}(q_i, b)$
  - $i = i + 1$
- Retourner $(r_i, \ldots, r_1, r_0)$.

Exercice 12 Prouver que cet algorithme est correct. Combien de divisions non triviales ?

Exercice 13 Ecrire une version récursive.

Exercice 14 Montrer que l'écriture d'un entier $a \in \mathbb{N}$ en base b est unique.

Exercice 15 Calculer l'écriture de 128 en base 3.

Exercice 16 Ecrire un algorithme d'écriture en base b qui se passe de la division euclidienne.

Exercice 17 (Evaluation) Ecrire un algorithme d'évaluation qui étant donné $(r_k, \ldots, r_0)_b$, calcule la valeur de l'entier n correspondant. Estimer la complexité.

Plus généralement, on peut définir l'écriture en base b dans tout anneau euclidien. L'algorithme précédent se généralise alors *mutatis mutandi*. Regardons de plus près la complexité dans le cas des polynômes.

3.5 Complexité de l'écriture en base b (cas des polynômes).

Soit K un corps et $a, b \in K[X]$, b non constant. Dans ce cas, l'écriture de a en base b s'écrit

$$a = \sum_{i=0}^{k} r_i b^i, \quad \deg(r_i) < \deg(b),\ r_k \neq 0.$$

On appelle également cette écriture le *développement b-adique* de a ou le *développement de Taylor généralisé* (le cas important $b = X - x_0$ correspond au développement de Taylor usuel d'une fonction au point x_0).

Théorème 9 L'algorithme **Baseb** calcule le développement b-adique d'un polynôme $a \in K[X]$ de degré n avec $O(n^2)$ opérations dans K (avec la division euclidienne usuelle de $K[X]$).

Remarque 8 On obtient une complexité binaire analogue dans le cas de deux entiers $a, b \in \mathbb{Z}$ de taille n.

Exercice 18 Prouver ce théorème.

Développement en base b rapide. Là encore, on peut atteindre une meilleure complexité en utilisant une approche "diviser pour régner" et la division rapide.

Soit $n = \deg(a)$ et $m = \deg(b)$. Soit k l'unique entier tel que $(k-1)m \leq n < km$, de sorte que a admet pour développement b-adique $a = \sum_{i=0}^{k-1} r_i b^i$. On peut supposer que k est une puissance de 2 (quitte à ajouter artificiellement des coefficients nuls au développement b-adique de a). La stratégie repose sur les égalités

$$a = b^{k/2} q + r, \quad r = \sum_{i=0}^{k/2-1} r_i b^i, \quad q = \sum_{i=0}^{k/2-1} r_{i+k/2} b^i.$$

La première égalité correspond à la division euclidienne de a par $b^{k/2}$ (en effet, $\deg(r) < k \deg(b)/2 = \deg(b^{k/2})$) et les deux dernières égalités correspondant au développement b-adique de r et q. On obtient ainsi une stratégie "diviser pour régner" facile à mettre en oeuvre.

Théorème 10 (écriture en base b rapide) Soit $a \in K[X]$ et soit M une fonction de multiplication dans $K[X]$. On suppose que la division euclidienne dans $K[X]$ coûte $\mathrm{M}(n)$ (Théorème 8). On peut calculer le développement b-adique d'un polynôme $a \in K[X]$ de degré au plus n avec $O(\mathrm{M}(n)\log(n))$ opérations dans K, soit $\tilde{O}(n)$ opérations si on utilise la multiplication rapide.

Remarque 9 Le théorème 10 admet son analogue dans le cas des entiers.

Exercice 19 Ecrire un tel algorithme et prouver le théorème.

3.6 Application au calcul modulaire

Soit A un anneau euclidien (donc unitaire, commutatif, intègre et principal). Soit $a \in A$. En supposant l'anneau A effectif, l'anneau quotient A/aA est lui aussi effectif. En effet, l'existence d'une division euclidienne dans A permet de faire du *calcul modulaire* : il suffit de choisir comme représentant d'une classe $x + aA \in A/aA$ le reste de la division euclidienne de x par a. Ceci permet en particulier de faire des calculs rapides dans les corps finis et les corps de nombres.

Théorème 11 (Coût du calcul modulaire) Soit $P \in K[X]$ de degré n. Une addition dans $K[X]/(P)$ coûte $O(n)$ opérations dans K et une multiplication coûte $O(\mathrm{M}(n))$ opérations dans K.

Preuve. On représente $a, b \in K[X]/(P)$ comme des polynômes de degrés $< n$. L'addition se fait terme à terme. Pour multiplier on retourne $\mathrm{Rem}(ab, P)$. Le coût découle du Théorème 8. □

Remarque 10 On a un résultat analogue pour les anneaux $\mathbb{Z}/N\mathbb{Z}$. La preuve est similaire (l'addition est plus délicate du fait des retenues).

Exercice 20 Ecrire un algorithme d'exponentiation rapide dans $\mathbb{Z}/N\mathbb{Z}$. Complexité ?

Exercice 21 Ecrire un algorithme de multiplication dans $\mathbb{Q}(\sqrt[3]{2})$.

Exercice 22 Ecrire un algorithme de multiplication dans le corps fini $\mathbb{F}_4$.