Odile Fleury
Loic Foissy
Alain Ninet

Agrégation
de
Mathématiques
Option C

Algèbre et calcul formel

2e édition

ellipses

Chapitre 1

Algorithmes et complexité

Le terme « algorithme » date du XIII^{e} siècle, où il était synonyme d'arithmétique pour les scientifiques arabes. Plusieurs étymologies sont envisagées, la plus répandue indiquant qu'il vient du nom du mathématicien arabe Al-Khwârizmî, qui vivait au temps du calife Almamoun, dans le premier tiers du IX^{e} siècle. Aujourd'hui, un algorithme désigne une série d'opérations ou d'instructions, destinées à résoudre un problème donné.
Nous allons d'abord voir comment on représente un nombre en base 2 et en machine, avant de parler d'algorithmes et de complexité, que nous illustrerons par plusieurs exemples classiques, à connaître pour passer l'agrégation.
On pourra trouver des informations supplémentaires sur le contenu de ce chapitre dans [BCG^{+}17], [Buc06], [Kob98] et [Sch04] pour ce qui concerne les algorithmes en général, [Cia94], [Dem09], [QSS00] et [Zem00] pour les algorithmes classiques.

1.1 Représentation en machine

Les processeurs actuels manipulent des bits, c'est-à-dire des éléments de $\{0,1\}$. La base logique pour travailler est donc la base 2, en binaire. Cependant, une machine ne peut stocker qu'un nombre fini de bits, donc de chiffres.
Une norme universelle a été instaurée, afin de définir un protocole de représentation des nombres réels : la norme ANSI/IEEE Standard 754 (American National Standards Institute, Institute of Electrical and Electronics Engineers, datant de 1985). Elle permet de créer des réels en machine et de leur associer 5 opérations élémentaires +, -, $\times$, /, $\sqrt{}$.

Nous allons voir dans cette partie comment est définie cette norme, et par extension comment l'utiliser pour stocker, manipuler et opérer avec des complexes, des grands entiers, des éléments de $\mathbb{Z}/n\mathbb{Z}$, des polynômes, des matrices, des éléments d'un corps fini.

Représentation des réels

Définition 1.1.1. Soit $B = (b_\ell, ..., b_{-k}) \in \{0,1\}^{k+\ell+1}$, $\ell \in \mathbb{N}$, $k \in \mathbb{N} \cup \{+\infty\}$ et $x \in \mathbb{R}^+$. On dit que B est une représentation binaire (ou en base 2) de x si

$$x = \sum_{j=0}^{\ell} b_j \, 2^j + \sum_{i=1}^{k} b_{-i} \, 2^{-i}.$$

On la note généralement

$$b_\ell ... b_0, b_{-1} ... b_{-k_2}.$$

On peut définir de manière similaire la représentation binaire d'un réel négatif.

Remarque 1.1.2. Il n'y a pas forcément unicité de la représentation. En effet, on a $1_2 = 0,11111\ldots_2$ car $\sum_{i=1}^{\infty} 2^{-i} = 2^{-1} \times \frac{1}{1-2^{-1}} = 1$.

Exemple 1.1.3.

1. $145 = 128 + 16 + 1 = 2^7 + 2^4 + 2^0$ donc 10010001_2 est une représentation binaire de 145.
2. $-23,6875 = -(16 + 4 + 2 + 1 + \frac{8+2+1}{16})$, ce qui s'écrit avec des puissances de 2 comme $-23,6875 = -(2^4 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4})$ donc $-10111,1011_2$ est une représentation binaire de x.

Définition 1.1.4. Soit $b \in \mathbb{N}$ et $m \in \mathbb{N}^*$. Soit x un réel. On dira que x est un flottant normalisé pour les paramètres b et m s'il existe :

- un entier $s \in \{0,1\}$,
- une famille $(E_j)_{0 \leqslant j \leqslant b} \in \{0,1\}^{b+1}$ (non tous nuls et non tous égaux à 1),
- une famille $(x_i)_{1 \leqslant i \leqslant m} \in \{0,1\}^m$

tels que $x = (-1)^s \left(1 + \sum_{j=1}^{m} x_j \times 2^{-j}\right) \times 2^e$ avec $e = E - (2^b - 1)$ et $E = \sum_{k=0}^{b} E_k \times 2^k$.

On appelle s le signe de x, e son exposant, E son exposant biaisé, $E - e$ le biais et la famille $(x_i)_{1 \leqslant i \leqslant m}$ la mantisse de x.

Remarque 1.1.5.
1. La représentation binaire de E est $E_b \dots E_0$ et $x = (-1)^s \times 1, x_1 x_2 \dots x_{m_2} \times 2^e$.
2. On a $1 \leqslant E \leqslant 2^{b+1} - 2$ et $-(2^b - 2) \leqslant e \leqslant 2^b - 1$ et il n'existe qu'un nombre fini de réels flottants normalisés. Le plus petit, en valeur absolue, est $2^{-(2^b-2)}$, le plus grand $1,1 \dots 1_2 \times 2^{2^b-1}$.
3. Le réel 0 n'est pas un flottant normalisé.
4. Les flottants normalisés sont stockés en machine sur $b + m + 1$ bits, sous la forme :
$$sE_b \dots E_0 x_1 \dots x_m.$$
5. Tout flottant normalisé est un réel de la forme $\pm \frac{a}{2^k}$ avec $a \in \mathbb{N}^*$ et $k \in \mathbb{N}$.

Remarque 1.1.6. La norme IEEE permet deux représentations machine :

— une sur 32 bits, dite simple précision : on prend $b = 7$ et $m = 23$, on représente donc les flottants normalisés sur 32 bits avec un biais de 127,

— une sur 64 bits, dite double précision : on prend $b = 11$ et $m = 52$, on représente donc les flottants normalisés sur 64 bits avec un biais de 1023.

Bien sûr, le nombre affiché à l'écran peut être tronqué. Le nombre stocké en mémoire contient souvent plus de chiffres significatifs que celui apparaîssant à l'écran, qui est lui en base 10.
En machine, par exemple sur 32 bits, un réel x est représenté :

1. Par sa valeur exacte s'il est un flottant normalisé sur 32 bits.
2. Par une valeur approchée si $2^{-126} < |x| < 1,1 \dots 1_2 \times 2^{127}$. Il y a plusieurs méthodes d'approximation, par exemple par le flottant normalisé soit immédiatement inférieur, soit immédiatement supérieur à x.
3. Par un signe quelconque, un exposant biaisé nul et une mantisse nulle si $x = 0$, c'est-à-dire par $s0 \dots 00 \dots 0$.
4. Par un signe quelconque, un exposant biaisé nul et une mantisse non nulle si x est dans l'underflow, c'est-à-dire si $0 < |x| < 2^{-126}$; il est stocké sous la forme $s0 \dots 0x_0 \dots x_{23}$.

5. Par un signe quelconque, un exposant biaisé de 255 et une mantisse nulle si x est dans l'overflow, c'est-à-dire si $|x| > 1,1\ldots1_2 \times 2^{127}$; il est stocké sous la forme $s1\ldots10\ldots0$.
6. Par un signe quelconque, un exposant biaisé de 255 et une mantisse non nulle si x correspond à une situation non représentable en machine, c'est-à-dire typiquement l'infini ; il est stocké sous la forme $s1\ldots1x_0\ldots x_{23}$.

Exemple 1.1.7. Soit x le réel dont l'écriture en norme IEEE 754 sur 32 bits est la suivante :

$$0\ \ 10001010\ \ 01101011000100000000000.$$

Il s'écrit :

$$x = (-1)^s \times 1,011010110001_2 \times 2^{E-127}$$

avec $s = 0$ et $E = 10001010_2 = 2^7 + 2^3 + 2^1 = 128 + 8 + 2 = 138$, et donc :

$$x = (1 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-12}) \times 2^{138-127} = 2904,5.$$

La norme IEEE impose également, pour chacune des 5 opérations élémentaires (mais pas pour **sin** ou **exp** par exemple), la façon de calculer l'arrondi correct (suivant le type d'arrondi choisi). C'est-à-dire que si l'on prend x et y deux réels, et qu'on note X et Y leur représentation en machine, le calcul de $x + y$ par la machine donnera l'arrondi de $x + y$ et pas l'arrondi de $X + Y$.
Il reste des problèmes, la norme n'étant pas parfaite. En effet, on manipule généralement des valeurs approchées, pas les valeurs réelles exactes.
On peut, par exemple, avoir le cas de trois flottants u, v et w tels que la représentation de $u + v$ et de $u + w$ soit la même que celle de u, mais que celle de $u + (v + w)$ soit différente.

Exemple 1.1.8. En se basant sur une simple précision, et en posant $u = 2^{-102}$, $v = w = 2^{-126}$, $u + v$ et $u + w$ seront représentés par u, alors que $u + (v + w)$ sera bien représenté par $u + v + w$. Dans le premier cas, v est trop petit par rapport à u et n'apparaît pas dans la mantisse de $u + v$.

Des problèmes peuvent apparaître lors de calculs de limites de suites convergentes par exemple. En général, les problèmes surviennent lorsque des calculs vont nous amener près de 0 ou de l'infini ou lorsque l'on fait des calculs sur des réels de tailles très différentes (somme d'un grand et d'un petit réel par exemple, le plus petit étant plutôt "absorbé" par le plus grand).

On peut également se poser la question de la représentation des nombres complexes, des grands entiers, des éléments de $\mathbb{Z}/n\mathbb{Z}$, des matrices, des polynômes ou des éléments d'un corps fini.

Autres représentations

Voyons maintenant comment représenter différents autres objets mathématiques usuels (la liste n'étant pas exhaustive) :

— Un réel, en norme IEEE 754, demandera 32 ou 64 bits pour être stocké sous la forme approchée d'un flottant normalisé, en simple ou double précision, comme on vient de le voir. Un complexe étant représenté par ses parties réelles et imaginaires - toutes deux réelles - il sera représenté en machine par 2 flottants normalisés.

— Pour de très grands entiers ($|n| \geqslant 2^{m+1}$), si l'on veut faire des opérations arithmétiques, on ne peut se permettre d'approximer. La solution est de stocker, dans une liste, l'écriture binaire de l'entier, chaque coefficient de l'écriture étant représenté par un flottant normalisé. Cette liste comportera exactement $k+1$ termes, avec k tel que $2^k \leqslant n < 2^{k+1}$, soit $k = \left\lfloor \dfrac{\ln n}{\ln 2} \right\rfloor$. Il faudra donc une liste de $\lfloor \log_2(n) \rfloor$ flottants normalisés pour représenter l'entier.

— Un élément x de $\mathbb{Z}/n\mathbb{Z}$ sera généralement représenté par deux entiers : l'entier n ainsi que l'unique entier k tel que $0 \leqslant k \leqslant n-1$ et $\overline{x} = \overline{k}$. Dans certaines situations, il peut aussi être intéressant d'utiliser l'unique entier k tel que $-n/2 < k \leqslant n/2$ et $\overline{x} = \overline{k}$, mais nous utiliserons en général la première représentation. La façon dont sont représentés ces deux entiers en machines dépendra de la taille de n, suivant qu'il s'agit d'un petit ou d'un grand entier.

— Une matrice dans un anneau $\mathbb{A}$ de n lignes et p colonnes sera représentée en machine par une liste de np éléments de $\mathbb{A}$. Remarquons que, pour les matrices creuses, d'autres moyens de stockage, moins gourmands en espace mémoire, existent, mais nous ne nous y intéresserons pas dans ce cours.

— Pour stocker en machine un polynôme de degré n à coefficients dans un anneau $\mathbb{A}$, il faudra prévoir une liste de $n+1$ éléments de $\mathbb{A}$. Là encore, le cas particulier d'un polynôme de grand degré mais ayant peu de coefficients non nuls conduirait à une autre représentation, plus économique.

— Les éléments d'un corps $\mathbb{F}_q$, avec $q = p^r$:
  — seront représentés comme dans $\mathbb{Z}/p\mathbb{Z}$ si $r = 1$,
  — seront représentés par un couple de polynômes (P, Q) de $\mathbb{F}_p[X]$, où Q est un polynôme de degré au plus $r - 1$ et P un polynôme irréductible de degré r si $r > 1$ (voir le chapitre sur les corps finis).

1.2 Algorithmes, complexité

Nous allons maintenant donner quelques notions d'algorithmique et de complexité. Le but n'est pas de voir la théorie générale de l'algorithmique de façon exhaustive, mais d'en donner une première approche, suffisante pour l'agrégation.

Définition 1.2.1. Un algorithme est une suite d'instructions qui, à partir d'un certain nombre de données de départ, va calculer un certain nombre de résultats.
Un algorithme se présente ainsi : on donne le but de l'algorithme, la liste des données en entrée et celle des résultats en sortie (avec éventuellement leur taille et leur type), puis le corps de l'algorithme (les instructions à effectuer). Idéalement, il est présenté de sorte à être facilement transposable dans un langage de programmation ou un logiciel (pour nous, SageMath).

En présence d'un nouvel algorithme, pensez à vérifier qu'il s'arrête en un temps fini (même long) et qu'il donne effectivement le résultat voulu.
On calcule également sa complexité afin de vérifier qu'il s'exécute en un temps et avec une capacité mémoire raisonnables.

Complexité

Il y a deux types de complexités : spatiale (l'espace mémoire nécessaire à l'exécution de l'algorithme) et temporelle (temps nécessaire à l'exécution de l'algorithme). Souvent, nous dirons simplement complexité lorsque nous parlerons de complexité temporelle. Deux cas peuvent se présenter :

— soit on peut calculer précisément la complexité $T(n)$ en fonction de $n \in \mathbb{N}$, la taille ou le nombre de données - éventuellement $T(n_1, n_2, \ldots, n_k)$ s'il y a plusieurs données,

— soit on ne le peut pas et on cherche une majoration du nombre d'opérations, c'est-à-dire une application f à valeurs dans $\mathbb{R}_+^*$ telle que $T(n) = \mathcal{O}(f(n))$ quand $n \to +\infty$. On rappelle que cela signifie qu'il

existe une constante $C > 0$ et $n_0 \in \mathbb{N}$ tels que, pour tout $n \geqslant n_0$, on a $T(n) \leqslant C\, f(n)$.

Suivant le cas de figure, on dit que la complexité est en T (si on a pu déterminer T) ou au plus en f. Le plus souvent, nous ne pourrons pas déterminer T et chercherons plutôt un majorant f.

Définition 1.2.2. Si $f(n) = n$, on dit que l'algorithme est linéaire.
Si $f(n) = n^2$, on dit que l'algorithme est quadratique.
Plus généralement, si $f(n) = n^r$ avec $r \geqslant 1$, on dit que l'algorithme est polynomial.
Si $n^r < f(n) < c^n$ pour tous $r \geqslant 1$ et $c > 1$, on dit que l'algorithme est sous-exponentiel.
Si $f(n) = c^n$ avec $c > 1$, on dit que l'algorithme est exponentiel.
Un bon algorithme, c'est-à-dire qui peut s'exécuter en un temps raisonnable, est polynomial (avec r de préférence pas trop grand).

La complexité spatiale (ou physique) est surtout intéressante pour des calculs manipulant de grosses matrices, des polynômes de grands degrés ou de grands nombres entiers :

— Un réel, en norme IEEE 754, demandera 32 ou 64 bits d'espace mémoire pour être stocké. Un complexe en demandera le double. La complexité spatiale est constante.

— Pour de très grands entiers, nous avons vu qu'il faut $\lfloor \log_2(n) \rfloor$ flottants normalisés pour représenter l'entier, la complexité spatiale sera donc en $\mathcal{O}(\log_2(n)) = \mathcal{O}(\ln(n))$.

— Un élément x de $\mathbb{Z}/n\mathbb{Z}$ étant représenté par deux entiers, il demandera le même ordre de place en mémoire qu'un seul entier, donc suivant les cas la complexité est constante si n est petit, en $\mathcal{O}(\ln(n))$ si n est grand.

— Une matrice dans un anneau $\mathbb{A}$, de n lignes et p colonnes, demandera à stocker une liste de np éléments de $\mathbb{A}$, sa complexité spatiale sera donc en $\mathcal{O}(npa)$), où a est la complexité spatiale d'un élément de $\mathbb{A}$. Par exemple, une matrice de grands entiers dont les coefficients sont majorés, en valeur absolue, par un entier M, aura une complexité spatiale en $\mathcal{O}(np\ln(M))$.

— Un polynôme de degré n à coefficients dans un anneau $\mathbb{A}$ demandera à stocker une liste de $n+1$ éléments de $\mathbb{A}$, sa complexité spatiale sera donc en $\mathcal{O}(na)$), où a est la complexité spatiale d'un élément de $\mathbb{A}$. Par exemple, un polynôme de grands entiers dont les coefficients sont ma-

jorés, en valeur absolue, par un entier M, aura une complexité spatiale en $\mathcal{O}(n \ln(M))$.

— Les éléments d'un corps $\mathbb{F}_q$, avec $q = p^r$ auront une complexité spatiale :

  — constante si p petit,

  — en $\mathcal{O}(r \ln(p)) = \mathcal{O}(\ln(p^r))$ si p grand.

Voyons maintenant, suivant les cas de figure, le coût des opérations usuelles, c'est-à-dire leur complexité temporelle.

Coût des opérations élémentaires

Chaque opération élémentaire sur des flottants normalisés, explicitée dans la norme ANSI/IEEE Standard 754, a un coût forfaitaire constant maximum, dépendant de la machine, du système d'exploitation et éventuellement du logiciel.

Lorsque l'on cherche la complexité d'un algorithme, on comptabilise trois types d'opérations :

— Exécution d'une opération élémentaire $(+, -, \times, /, \sqrt{.})$. Une addition et une soustraction ont le même temps d'exécution, majoré par une constante c_+. Une multiplication, une division ou une extraction de racine carrée ont un temps d'exécution majoré respectivement par une constante $c_\times$, $c_/$ et $c_{\sqrt{.}}$, avec $c_+ < c_\times < c_/ < c_{\sqrt{.}}$.

— Exécution d'un test : le temps nécessaire à cette opération peut être comparé à celui d'une addition. En effet, comparer a et b revient à étudier le signe de $a - b$, c'est-à-dire le premier bit de la représentation machine de $a - b$.

— Autre type d'opération dont on ne connaît pas le temps de calcul. On la fait apparaître comme un paramètre dans le calcul de la complexité.

Pour calculer ou majorer la complexité temporelle d'un algorithme, on calcule le nombre de ces opérations. On se contentera parfois d'estimer les plus nombreuses et/ou les plus coûteuses.

Coût des opérations élémentaires pour les grands entiers

Pour certaines applications comme la cryptographie, on est amené à manipuler de grands entiers, de l'ordre de $2^{1000} \simeq 10^{301}$, voire plus.