Chapitre 1

Cours

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 sous le 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 dans une base quelconque et en machine, avant de parler d'algorithmes et de complexité, que nous illustrerons par un exemple classique. On pourra trouver des informations supplémentaires sur le contenu de ce chapitre dans [QSS00] ou dans [Sch04].

1.1 Représentation des réels

Nous allons parler de représentation de nombres (entiers, rationnels ou réels) en base b et en machine. En machine, n'importe quel réel (en fait, n'importe quel caractère) est représenté par des 0 et des 1 (des bits), donc en base 2.

Réels en base b

Commençons par donner la définition de la représentation d'un réel en base b, b étant un entier supérieur ou égal à 2.

Définition 1.1.1 *Soit $x \in \mathbb{R}$. Soit $k \in \mathbb{N}$, $s = \pm 1$, $(n_j)_{0 \leq j \leq k}$ et $(m_j)_{j \in \mathbb{N}^*}$ deux familles d'entiers de $\{0, \ldots, b-1\}$. On dit que k, s et les deux familles forment une représentation de x en base b si $s \times x = \sum_{j=0}^{k} n_j \, b^j + \sum_{j=1}^{+\infty} m_j \, b^{-j}$. On le note alors $x = s \times n_k \ldots n_0, m_1 m_2 \ldots_b$.*

Exemples : Étudions les représentations binaires de 145 et $-23,6875$. On a :

$$145 = 128 + 16 + 1 = 2^7 + 2^4 + 2^0$$

donc 10 010 001 est une représentation binaire de 145. De la même façon,

$$\begin{aligned} 23,6875 &= 16 + 4 + 2 + 1 + \frac{11}{16} = 16 + 4 + 2 + 1 + \frac{8+2+1}{16} \\ &= 2^4 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4} \end{aligned}$$

donc $-10\ 111,1011$ est une représentation binaire de $-23,6875$.

Remarque : Dans le cas où la famille des $(m_j)_{j\in\mathbb{N}^*}$ serait finie (constante nulle à partir d'un certain rang l), on note :

$$s\ x = n_p \dots n_0, m_1 m_2 \dots m_{l\ b}.$$

Remarque : Cette écriture a toujours un sens. En effet, la série $\displaystyle\sum_{j=1}^{+\infty} m_j\ b^{-j}$ est convergente (majorée par 1) pour toute base b et toute suite $(m_j)_{j\in\mathbb{N}^*}$ avec $0 \le m_j \le b-1$:

$$0 \le \sum_{j=1}^{+\infty} m_j\ b^{-j} \le (b-1)\ \sum_{j=1}^{+\infty} b^{-j} = (b-1) \times \frac{1}{b}\ \frac{1}{1-\frac{1}{b}} = 1.$$

Voyons le cas particulier des entiers, qui peuvent s'écrire de façon unique avec une famille $(m_j)_{j\in\mathbb{N}^*}$ constante nulle.

Proposition 1.1.2 *Soit $b \in \mathbb{N}$, $b \ge 2$. Soit $n \in \mathbb{Z}^*$.*
Il existe un unique $k \in \mathbb{N}$, un unique $s = \pm 1$ et une unique famille $\{n_0, \dots, n_k\}$ d'entiers dans $\{0, 1, \dots, b-1\}$, $n_k \neq 0$, tels que :

$$n = s \times n_k\ n_{k-1}\ \dots\ n_{0\ b} = s \times \sum_{j=0}^{k} n_j\ b^j.$$

Preuve :
L'existence est obtenue facilement par divisions successives de n par b. L'unicité de s est immédiate par considération de signes.

L'unicité des autres termes se montre ainsi : soit $n = s\ \displaystyle\sum_{j=0}^{k} n_j\ b^j = s\ \sum_{j=0}^{l} m_j\ b^j$.

On suppose par exemple que $k \le l$.

Alors, on a nécessairement $\sum_{j=0}^{k}(n_j - m_j)\ b^j = m_{k+1}\ b^{k+1} + \ldots + m_l\ b^l$.
De plus, le terme de gauche est en valeur absolue strictement plus petit que b^{k+1} ; le terme de droite est supérieur à b^{k+1} ou nul. Les deux termes doivent donc être nuls, ce qui implique $k = l$ et $\sum_{j=0}^{k}(m_j - n_j)\ b^j = 0$.
Par le même genre de raisonnement et par une récurrence rétrograde sur j, on montre que pour tout $0 \leq j \leq k$ on a $n_j - m_j = 0$.

■

On a donc une seule façon d'écrire un entier non nul sans chiffre après la virgule, quelle que soit la base.
Cependant, si on considère les entiers comme des rationnels, et qu'on suppose possible l'existence de chiffres après la virgule en écriture en base b, l'unicité n'est plus de mise. Par exemple, si l'on prend une base quelconque, on a :

$$1 = 1_b \qquad \text{et} \qquad 1 = 0, ccc\ldots_b$$

avec $c = b - 1$. La première écriture est triviale, la seconde signifie que :

$$\sum_{j=1}^{+\infty} c\ b^{-j} = \frac{c}{b}\ \frac{1}{1 - \frac{1}{b}} = 1.$$

De façon plus générale, on a la proposition suivante :

Proposition 1.1.3 *Soit $\frac{p}{q}$ une fraction rationnelle irréductible positive telle que $0 < p < q$, $q \in \mathbb{N}^*$. Il existe une base b dans laquelle cette fraction admet deux représentations distinctes.*

Preuve : Il suffit de reprendre la remarque précédente sur l'écriture de 1 et de voir que, si l'on se place en base q, la fraction peut s'écrire $0, p_q$ ou $0, (p-1)(q-1)(q-1)\ldots_q$.

■

En fait, on peut montrer que les rationnels sont les seuls réels pouvant s'écrire sous plusieurs formes dans certaines bases. Un irrationnel n'aura qu'une écriture possible, quelle que soit la base choisie.
Cependant, bien que n'étant pas unique, l'écriture d'un rationnel en base b vérifie certaines règles données dans la proposition suivante.

Proposition 1.1.4 *Soit $b \in \mathbb{N}$, $b \geq 2$. Soit p et q deux entiers premiers entre eux, $q \in \mathbb{N}^*$ et $0 \leq p \leq q-1$. Alors :*

1. *soit il existe $k > 0$ et une famille finie $m_1, \ldots, m_k$ dans $\{0, \ldots, b-1\}$ tels que $\frac{p}{q} = \sum_{j=1}^{k} m_j \; b^{-j}$. On note alors $0, m_1 \ldots m_{k\ b}$ une représentation de $\frac{p}{q}$ en base b,*
2. *soit il existe une famille périodique $(m_j)_{j \in \mathbb{N}^*}$ dans $\{0, \ldots, b-1\}$, de période $T \in \mathbb{N}^*$ à partir d'un rang k, telle que $\frac{p}{q} = \sum_{j=1}^{+\infty} m_j \; b^{-j}$. On note alors $0, m_1 \ldots m_{k-1} \underline{m_k \ldots m_{k+T-1}} \ldots_b$ une représentation de $\frac{p}{q}$ en base b.*

Remarque : On verra une réciproque de cette proposition en exercice.

Preuve : On fait la démonstration en construisant la suite $(m_j)_j$ ainsi : posons $r_0 = p$ et définissons par récurrence deux suites m_j et r_j, $j \in \mathbb{N}^*$, par : m_j est le quotient et r_j le reste de la division euclidienne de br_{j-1} par q.
On vérifie aisément que $0 \leq m_j \leq b-1$ (car $0 \leq r_j \leq q-1$ pour tout j). De plus, on peut distinguer deux cas :

1. Ou il existe k tel que $r_k = 0$, et dans ce cas, pour tout $j > k$, on a $m_j = r_j = 0$.
2. Ou tous les r_j sont non nuls. Alors, puisque ce sont les restes d'une division par q, on a $0 < r_j < q$ et $r_j \in \mathbb{N}$: les r_j prennent un nombre fini de valeurs identiques, il existe donc k et T tels que $r_{k-1} = r_{k+T-1}$. On peut alors montrer facilement par récurrence que pour tout $j \geq k$, $r_j = r_{j+T}$ et $m_j = m_{j+T}$.

Il reste à vérifier qu'on obtient bien une représentation de $\frac{p}{q}$, grâce à la formule (qu'on montre par récurrence) :

$$\frac{p}{q} = \sum_{j=1}^{k} m_j \; b^{-j} + \frac{r_k}{q} \; b^{-k} \qquad \forall \; k \in \mathbb{N}^*.$$

■

La démonstration est constructive : elle donne une façon de construire une écriture de n'importe quel rationnel. Mais comme on l'a vu, cette écriture n'est pas forcément unique.

Représentation en machine

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

Définition 1.1.5 *Soit x un réel. On dira que x est un flottant normalisé s'il existe :*
- *un entier $s \in \{0,1\}$,*
- *une famille $(E_j)_{0\leq j\leq 7} \in \{0,1\}^8$ (non tous nuls et non tous égaux à 1),*
- *une famille $(x_i)_{1\leq i\leq 23} \in \{0,1\}^{23}$*

tels que

$$x = (-1)^s \ 1, x_1 x_2 \dots x_{23\ 2} \times 2^e$$

avec $E = E_7 \dots E_{0\ 2}$ et $e = E - 127$.
On appelle s le signe de x, e son exposant, E son exposant biaisé, $E - e$ le biais (ici, 127) et la famille $(x_i)_{1\leq i\leq 23}$ la mantisse de x.

Remarque : On a $1 \leq E \leq 254$ et $-126 \leq e \leq 127$.
Si $x > 0$, $s = 0$, et si $x < 0$, $s = 1$.

Remarque : Il n'existe qu'un nombre fini de réels flottants normalisés. Le plus petit, en valeur absolue, est 2^{-126}, le plus grand $1,1\dots1_2 \times 2^{127}$, soit 340 282 346 638 528 859 811 704 183 484 516 925 440.
Le réel 0 n'est pas un flottant normalisé.

Ces flottants sont les seuls réels représentables en machine par cette norme sous 32 bits sous forme exacte. Ils y sont représentés de la manière suivante :

$$s\ E_7 \dots E_0\ x_1 \dots x_{23}.$$

En machine, 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 $2^{-126} < |x| < 1,1\dots1_2 \times 2^{127}$. Il y a plusieurs méthodes d'approximation, par exemple par le flottant normalisé soit immédiatement inférieur, soit immédiatement supérieur, soit le plus proche de x.

3. Par un signe quelconque, un exposant biaisé nul et une mantisse nulle si $x = 0$, c'est-à-dire par $s0\ldots00\ldots0$.
4. Par un signe quelconque, un exposant biaisé nul et une mantisse non nulle quelconque (par exemple des 1) si x est dans l'underflow, c'est-à-dire si $0 < |x| < 2^{-126}$; il est stocké sous la forme $s0\ldots0x_0\ldots x_{23}$.
5. Par un signe quelconque, un exposant biaisé de 255 et une mantisse nulle quelconque (par exemple des 1) 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 un nombre non représentable en machine, c'est-à-dire typiquement l'infini ; il est stocké sous la forme $s1\ldots1x_0\ldots x_{23}$.

Exemples :
Le flottant normalisé dont l'écriture en norme IEEE 754 est :

$$0 \quad 10001010 \quad 01101011000100000000000$$

correspond au réel

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

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

$$x = (1 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-12}) \times 2^{138-127} = 2\ 904,5.$$

Le réel $x = 165,375$ s'écrit en binaire :

$$165 = 128 + 32 + 4 + 1 = 2^7 + 2^5 + 2^2 + 2^0 = 10\ 100\ 101_2,$$

$$0,375 = \frac{375}{1000} = \frac{3}{8} = \frac{1}{4} + \frac{1}{8} = 2^{-2} + 2^{-3} = 0,011_2.$$

On a donc l'écriture binaire de x : $x = 10\ 100\ 101,011_2 = 1,0100101011_2 \times 2^7$.
Comme la mantisse de x compte moins de 23 chiffres et que son exposant est compris entre -126 et 127, on en déduit que x est un flottant normalisé.
Son exposant biaisé sera $E = 7 + 127 = 134$. Il ne reste plus qu'à écrire E en binaire : $E = 134 = 128 + 4 + 2 = 2^7 + 2^2 + 2^1 = 10\ 000\ 110_2$. On en déduit donc l'écriture de x (positif) en norme IEEE :

$$0 \quad 10000110 \quad 01001010110000000000000.$$

La norme IEEE permet également une représentation machine sur 64 bits. Le principe est le même, avec un exposant biaisé sur 11 bits, un biais de 1023

et une mantisse sur 52 bits. On parle alors de double précision, contre de la simple précision sur 32 bits. On notera que désormais, la plupart des ordinateurs utilisent la double précision.

Bien sûr, le nombre affiché à l'écran est en général tronqué. Le nombre stocké en mémoire contient plus de chiffres significatifs que celui apparaissant à l'écran.

La norme IEEE impose également, pour chacune de ces 5 opérations élémentaires, l'arrondi correct (suivant le type d'arrondi choisi). C'est-à-dire que si 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 $x + y$, qui ne sera pas nécessairement $X + Y$.
Il reste des problèmes, la norme n'étant pas parfaite. En effet, on manipule malgré tout 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 $u + v$ soit la même que celle de u, celle de $u + w$ celle de u également, mais que celle de $u + (v + w)$ soit différente. Ainsi, le calcul de $(u + v) + w$ ne donnera pas le même résultat que $u + (v + w)$.

Exemple : $u = 2^{-120}$, $v = w = 2^{-127}$.

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.

1.2 Algorithmes

Nous allons maintenant donner quelques notions d'algorithmique. Le but n'est pas de voir une théorie générale de l'algorithmique, mais d'en donner une première approche intuitive, suffisante pour l'agrégation. Nous allons donner une définition pratique, non théorique, d'un algorithme, puis nous verrons l'exemple du calcul des racines d'un polynôme de degré 2.

Définition 1.2.1 *Un algorithme est une suite d'instructions qui, depuis un certain nombre de données de départ, va permettre d'obtenir 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).

```
a : réel ; b : réel ; c : réel ;
x1 : complexe ; x2 : complexe ;
/*x1 et x2 sont les racines du polynôme aX^2+bX+c*/

d ← b ∧2−4ac ;
Si (d<0) Alors
    x1 ← −b\(2a)+i√(−d)\(2a) ;
    x2 ← −b\(2a)−i√(−d)\(2a) ;
Sinon
    Si (d=0) Alors
        x1 ← −b\(2a) ;
        x2 ← x1 ;
    Sinon
        x1 ← −b\(2a)+√d\(2a) ;
        x2 ← −b\(2a)−√d\(2a) ;
    Fin Si
Fin Si
```

Remarque : L'algorithme, une fois écrit, n'est pas forcément optimal. Il est simplement sous une forme directement programmable sur machine. Mais il peut éventuellement être amélioré. Par exemple, dans le cas précédent, on pourrait remplacer le corps par :

```
a : réel ; b : réel ; c : réel ;
x1 : complexe ; x2 : complexe ;
/*x1 et x2 sont les racines du polynôme aX^2+bX+c*/

d ← b ∧ 2−4ac ;
r ← −b\(2a) ;
Si (d<0) Alors
    im ← i√(−d)\(2a) ;
Sinon
    im ← √d\(2a) ;
Fin Si
x1 ← r+im ;
x2 ← r−im ;
```