Codage binaire

Antoine Louatron

Table des matières

I Représentation des nombres entiers

I.1 Nombres entiers naturels

I.1.1 Représentation décimale

La manière habituelle pour écrire les nombres est d'utiliser "la base 10". C'est à dire qu'un nombre comme 8736 est en fait écrit comme somme de puissances croissantes de 10 (avec un coefficient < 10) :

$$8736 = 6 \times 10^0 + 3 \times 10^1 + 7 \times 10^2 + 8 \times 10^3$$

(Rappel : pour tout nombre $x \neq 0, x^0 = 1$.

I.1.2 Proposition
Tout nombre entier naturel non nul admet une unique représentation *binaire*, c'est à dire qu'il s'écrit d'une manière unique comme une somme de certaines puissances de 2. Plus précisément, pour tout entier naturel $n \neq 0$ il existe un unique $r \in \mathbb{N}$ et une unique famille $(b_0, \ldots, b_r) \in \{0,1\}^{r+1}$ telle que

$$n = \sum_{k=0}^{r} b_i 2^i$$

I.1.3 Exemple
Prenons $37 = 7 + 3 \times 10$.

On a également $37 = 32 + 5 = 32 + 4 + 1 = 1 + 2^2 + 2^5$. On note $100101_{(2)}$ cette représentation binaire. Les 1 marquent la position des puissances de 2 présentes. Le chiffre le plus a droite est la coefficient de 2^0.

Exercice 1
Donner la représentation décimale de $1101011_{(2)}$.

I.1.4 Recherche de l'écriture binaire

I.1.5 Proposition
Soit $n \in \mathbb{N}^*$. Le bit de poids faible (le facteur de 2^0) dans l'écriture de n vaut 0 ssi n est pair (et donc vaut 1 ssi n est impair).

Preuve.
Notons $n = (b_r \ldots b_1 b_0)_{(2)}$ la représentation binaire de n. On cherche à déterminer b_0 suivant la parité de n.
Or $n = 2 \times (2^{r-1} b_r + \cdots + b_1) + b_0$.

— SI n est pair alors $b_0 = n - 2 \times (2^{r-1} b_r + \cdots + b_1)$ est lui aussi un nombre pair. Comme $b_0 \in \{0,1\}$, $b_0 = 0$.

— Réciproquement, si b_0 est nul alors n s'écrit comme 2 fois un entier donc est pair par définition. ∎

I.1.6 Remarque

1. Avec les notations de la preuve précédente, l'écriture binaire de $\frac{n-b_0}{2}$ est la même que celle de n mais tronquée du dernier bit.
2. Si on effectue la division euclidienne de n par 2, alors le reste sera b_0 et le quotient $\frac{n-b_0}{2}$

I.1.7 Algorithme

— **Problème :** étant donné un nombre $n \in \mathbb{N}$, calculer sa représentation en base 2 (binaire).

— **Méthode :** effectuer la division euclidienne (avec reste qui vaut donc 0 ou 1) de n par 2 et noter le reste. Recommencer la procédure avec le quotient obtenu jusqu'à obtenir 0.

— **Solution :** noter les reste successifs de droite à gauche pour lire la représentation binaire.

I.1.8 Exercice
Appliquer à 357, 1278, 4096.

I.1.9 Écriture en pseudo-code

Entrée : n un entier positif.

```
TantQue n > 0
    q, r <- quotient et reste dans le division de n par 2
    Noter r
    n <- q
FinTantQue
```

Les restes successifs notés constituent la représentation binaire de n.

Intéressons nous à une propriété intéressante de cet algorithme : sa terminaison. Il s'agit de démontrer par un raisonnement logique que notre algorithme se termine en un nombre fini d'étapes pour toute entrée n.

Ici on va remarquer deux propriétés qui font de n un **variant de boucle** :

1. n est un entier naturel (positif)
2. Après un passage dans la boucle, n est **strictement** plus petit : on a ici $q \leqslant \frac{n}{2} < n$ car $n \geqslant 1$ pour qu'on soit dans la boucle.

Ainsi, on va effectuer un nombre fini de passage dans la boucle (au maximum n, car n décroit strictement donc baisse d'au moins 1 et doit rester > 0 pour ré-exécuter la boucle).

I.1.10 Questions

1. Quel est le plus grand nombre codable dans un octet (8 bits) ?
2. Dans 4 octets ? (c'est le cas classique des entiers dans les langages de programmation)

Exercice 2
Combien de nombre entiers distincts peut-on stocker si on dispose de $1Ko = 1000$ octets ?

I.2 Les entiers relatifs

I.2.1 Codage en pratique des négatifs

On sacrifie un bit pour représenter le signe. En pratique, on utilise pas exactement le premier bit pour le signe mais un représentation plus efficace pour calculer.

I.2.2 Complément à 2^n

On décide de stocker les entiers sur n bits exactement (en pratique, n vaut souvent 32 voire 64).
On peut représenter les entiers allant de -2^{n-1} à 2^{n-1} (combien d'entiers distincts ?) de la manière suivante.

1. Les entiers positifs sont représentés comme précédemment, avec des 0 à gauche pour compléter les n bits. Un entier positif a donc une représentation mémoire qui commence toujours par un 0.
2. Les entiers $i < 0$ sont représentés par $i + 2^n$. En pratique, cela revient à trouver le codage de $i - 1$ et remplacer tous les 0 par des 1.

I.2.3 Exemple
Les entiers relatifs codables sur 4 bits (avec $2^4 = 16$, on représente $[\![-8, 7]\!]$) sont les suivants :

Entier	Code
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

II Nombres flottants

II.1 Les nombres décimaux

II.1.1 Représentation finie

On ne peut bien évidemment pas retenir en mémoire toutes les décimales d'un nombre réel quelconque (cf $\pi, \sqrt{2}$). On travaille donc avec des *approximations*. Le principe est celui de la notation scientifique des calculatrices : on retient une partie décimale avec un nombre fixé de chiffre et un exposant (qui doit être dans un intervalle d'entier fixé au préalable).

ATTENTION : le séparateur n'est pas la virgule mais le point.

II.1.2 Nombres binaires à virgule

Quel sens donner à $1.0011_{(2)}$? On adapte la définition de la base 10 :

$$3.1416 = 3 + \frac{1}{10} + \frac{4}{100} + \frac{1}{1000} + \frac{6}{10000}$$

Ainsi $1.0011_{(2)} = 1 + \frac{0}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16}$.

Il est ainsi impossible de donner une représentation binaire finie de certains nombres décimaux. Par exemple $0.4 = 0.011001100110011\ldots$.

Même certains nombres qui ont une représentation finie en base 10 seront en fait des valeurs approchées en mémoire.

II.2 Représentation mémoire et limitations

II.2.1 Principe

On fixe a priori un nombre de bit bien défini pour chaque partie d'un nombre flottant

La représentation classique utilise 64 bits pour stocker les nombres flottants et la base utilisée est 2 :

— 1 bit pour le signe,

— 52 bits pour la partie significative,

— 11 bits pour l'exposant (les valeurs $00000000000_{(2)}$ et $11111111111_{(2)}$ sont réservées pour traiter des cas particuliers).

Quel est l'intervalle théorique pour les exposants ? Pour pouvoir utiliser des exposants négatifs, on convient de retrancher 1023 à tous les exposants stockés.

II.2.2 Conséquences des approximations

Les dernières décimales calculées peuvent être erronées.

On ne peut pas, en général, se fier aux comparaisons (égalité, supérieur, inférieur) entre nombres flottants, car des erreurs d'arrondis successives peuvent fausser le résultat.

On ne peut pas comparer deux flottants par un simple test d'égalité

II.2.3 Remarque

Il existe dans cette représentation un plus petit nombre réel strictement positif ! on voit donc bien que deux réels distincts peuvent être considéré égaux, il suffit que leur différence soit plus petite que ce "plus petit réel".

II.2.4 Exemples en python

```
1 - (0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1)
1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1
```

II.2.5 Explication de 10^{-16}

Calculer 2^{-53}. Il s'agit de la précision relative des écritures en flottant en python : le premier bit qui n'apparaît pas en mémoire.

III Représenter d'autres données

III.1 Du texte

III.1.1 Solution naïve

On décide de représenter chacune des 26 lettres de l'alphabet par un entier naturel de $[\![0, 25]\!]$. Combien de bits doit-on utiliser au minimum pour pouvoir coder toutes ces lettres ?

III.2 Etendre la table des caractères

En pratique on utilise plutôt de 1 à 2 octets par lettres suivant la convention retenue, pour pouvoir encoder tous les caractères spéciaux. Il existe de très nombreuses conventions encore utilisées.

III.3 Images

Classiquement, une image est représentée comme une suite de pixels, chaque pixel étant donné par 3 entiers de $[\![0, 255]\!]$ (sur un octet donc) représentant l'intensité des couleurs rouges, vert et bleu pour ce pixel.

Combien d'octet faut-il utiliser pour stocker une image de 12M de pixels ?