Informatique en CPGE (2018-2019)
Représentation des nombres

S. B.

Lycée des EK

13 octobre 2018

Les ordinateurs, comme d'autres appareils, permettent de mémoriser, de transmettre et de transformer des nombres, des textes, des images, des sons, etc. Ils fonctionnent avec des petits circuits électroniques qui ne peuvent être, chacun, que dans deux états représentés par 0 ou 1. Ce qui correspond à fermé ou ouvert, faux ou vrai, ...

Une telle valeur, 0 ou 1, s'appelle un booléen, un chiffre binaire (nombre en base 2) ou encore un bit (binary digit), de symbole b. La mémoire des ordinateurs contient une multitude de ces circuits à deux états appelé circuit mémoire un bit. Un circuit composé de plusieurs de ces circuits mémoire un bit, se décrit par une suite finie de 0 et de 1, que l'on appelle un mot. Un mot de 8 bits est un octet. Le byte, de symbole B, ou multiplet, est la plus petite unité adressable d'un ordinateur et correspond en général à un octet, (soit 1 B = 8 b).

Un nombre réel peut-être approximé avec la précision souhaitée par un nombre décimal qui peut s'exprimer à l'aide de nombres entiers.

Pour coder un caractère, nous attribuons un nombre à chaque caractère selon des normes précises. Ensuite, un texte est une suite de caractères, donc est codé par une suite de nombres.

Une image sur un écran d'ordinateur est composée de pixels (picture elements). Ce sont des petits carrés qui composent un quadrillage. Si l'image est en couleurs, trois paramètres codent le niveau de rouge, de vert et de bleu. C'est le système RGB, (red, green, blue) ou système RVB en français. Chacune de ces trois composantes est codée par un nombre de 0 à 255.

De même un système permet de coder un son par une suite de nombres.

Donc de manière générale, l'information peut être codée par des nombres, c'est-à-dire "numérisée".

Il ne reste plus qu'à savoir représenter un nombre entier par une suite composées de 0 et de 1. Et un fichier stocké sur un ordinateur ne sera qu'une suite plus ou moins longue composées de 0 et de 1.

Représentation en base dix

On utilise pour les nombres entiers naturels la notation décimale à position. Pour cela, dix chiffres sont nécessaires et suffisants : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
Par exemple, 3207 représente 3 milliers + 2 centaines + 0 dizaines + 7 unités.

$$3207 = 3 \times 10^3 + 2 \times 10^2 + 0 \times 10^1 + 7$$

Pratiquement, si on dispose de 3207 objets identiques, et que l'on constitue des paquets de 10 objets, on en obtient 320 et il reste 7 objets. Ensuite, avec les 320 paquets on forme des ensembles de 10 paquets, on en obtient 32 ; puis on regroupe les 32 ensembles par 10, on obtient 3 groupes et il reste 2 ensembles.

On utilise la base dix tous les jours, mais d'autres bases sont envisageables.

La base deux

En base deux il n'y a que deux chiffres, 0 et 1, et le principe reste le même qu'en base dix.

Par exemple, pour 47 : on obtient 23 paquets de deux et 1 unités, puis les 23 paquets se regroupent en 11 paquets de paquets et il reste 1 paquets, ensuite ...
$47 = 23 \times 2 + 1 = (11 \times 2 + 1) \times 2 + 1 =$
$((5 \times 2 + 1) \times 2 + 1) \times 2 + 1 = \ldots$

$$47 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1$$

Donc le nombre écrit 47 en base dix s'écrit 101111 en base deux. On peut utiliser la notation 101111_2.

Dans la notation binaire :

- le bit (ou chiffre) le plus à gauche est appelé bit de poids fort

- le bit (ou chiffre) le plus à droite est appelé bit de poids faible

Pour mesurer des quantité d'information, l'unité de base est le bit qui prend soit la valeur 0, soit la valeur 1.
Un ensemble de 4 bits consécutifs est appelé un quartet,
un ensemble de 8 bits consécutifs est appelé un octet (byte),
deux octets consécutifs (16 bits) forment un mot (word).

C'est l'octet qui est utilisé comme unité de référence pour mesurer la capacité des mémoires.

Remarque : pour multiplier par deux un nombre écrit en base deux, il suffit d'ajouter un zéro à droite du nombre :
$1011_2 \times 10_2 = 10110_2$

Une base quelconque

On peut généraliser à une base quelconque les méthodes précédentes.
Pour écrire les entiers naturels en base k, on a besoin de k chiffres. Quand on a n objets, on les groupe par paquets de k, qu'on regroupe à leur tour en paquets de k paquets, etc.
Autrement dit, on fait une succession de divisions par k, jusqu'à obtenir un quotient égal à 0. Ici encore, la multiplication d'un nombre par k consiste à ajouter un zéro à droite du nombre.

La base seize ou hexadécimale

La base seize est particulièrement intéressante. On a besoin de 16 chiffres : 0, 1, 2, 3, 4, 5 , 6, 7, 8, 9, puis A (dix), B (onze), C (douze), D (treize), E (quatorze) et F (quinze).

Si on travaille en base deux, l'écriture peut être très longue. Or un octet (huit bits) peut s'écrire simplement en base seize. On partage l'octet en deux et chaque partie de quatre bits s'écrit avec un chiffre de la base seize.

Démonstration

Le nombre seize s'écrit 10000 en base deux et 10 en base seize : $10000_2 = 10_{16}$;
donc
$(a_0a_1a_2a_3a_4a_5a_6a_7)_2 = (a_0a_1a_2a_3)_2 \times 10000_2 + (a_4a_5a_6a_7)_2$
$= (b_0)_{16} \times 10_{16} + (b_1)_{16} = (b_0b_1)_{16}$

Par exemple le nombre écrit 11010101 en base deux s'écrit D5 en base seize. En effet 11010101 se partage en 1101 et 0101 qui donnent respectivement D et 5 en base seize :
$11010101_2 = 1101_2 \times 10000_2 + 0101_2 = D_{16} \times 10_{16} + 5_{16} = D5_{16}$

Dans l'ordinateur

Dans un ordinateur c'est le codage binaire qui est utilisé.
Par exemple avec le nombre dont la représentation décimale est 59 :

$$59 = 1 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$
$$59 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$59 = 111011_2.$$

Avec des octets, soit huit bits, on peut représenter les entiers naturels de 0 (00000000 en base deux) à 255 (1111 1111 en base deux). Donc 59 sera représenté par 00111011.
Si on utilise seize bits, soit deux octets, on peut représenter les entiers naturels jusqu'à 65535 (1111 1111 1111 1111 en base deux) et dans ce cas 59 sera représenté par 0000000000111011.
Avec n bits, ce système permet de représenter les nombres entre 0 et $2^n - 1$, donc tout nombre k qui s'écrit :

$$k = \sum_{i=0}^{n-1} b_i 2^i \qquad b_i \in \{0, 1\}$$

Actuellement un entier est en général codé sur 4 octets (soit 32 bits) ou 8 octets (soit 64 bits).

Addition de nombres binaires

Pour additionner deux nombres binaires on procède comme en base 10.
A partir de 0+0=0, 1+0=0+1=1 et 1+1=10 on pose l'addition comme en base dix et chaque fois qu'il y a 1+1, on écrit 0 et on retient 1.
Par exemple :

Addition de nombres binaires

Pour additionner deux nombres binaires on procède comme en base 10.
A partir de 0+0=0, 1+0=0+1=1 et 1+1=10 on pose l'addition comme en base dix et chaque fois qu'il y a 1+1, on écrit 0 et on retient 1.
Par exemple :

1 1 1 0 1 vingt-neuf

Addition de nombres binaires

Pour additionner deux nombres binaires on procède comme en base 10.

A partir de 0+0=0, 1+0=0+1=1 et 1+1=10 on pose l'addition comme en base dix et chaque fois qu'il y a 1+1, on écrit 0 et on retient 1.

Par exemple :

	1	1	1	0	1	vingt-neuf
+	1	1	0	0	1	vingt-cinq

Addition de nombres binaires

Pour additionner deux nombres binaires on procède comme en base 10.

A partir de 0+0=0, 1+0=0+1=1 et 1+1=10 on pose l'addition comme en base dix et chaque fois qu'il y a 1+1, on écrit 0 et on retient 1.

Par exemple :

	1	1	1	0	1	vingt-neuf
+	1	1	0	0	1	vingt-cinq
	1			1		retenues
1	1	0	1	1	0	cinquante-quatre

Entiers relatifs

On pourrait utiliser un bit pour le signe et les autres bits pour la valeur absolue. Mais cette méthode a des inconvénients. On a préféré la méthode de représentation par complément à deux qui permet de réaliser des opérations arithmétiques sans problème. Celle-ci consiste à réaliser un complément à un de la valeur, puis d'ajouter 1 au résultat.

Par exemple sur 6 bits, pour obtenir le codage de -5 :

Par exemple sur 6 bits, pour obtenir le codage de -5 :

000101 (codage de 5 en binaire sur 6 bits)

Par exemple sur 6 bits, pour obtenir le codage de -5 :

000101 (codage de 5 en binaire sur 6 bits)
111010 (complément à un, on inverse chaque bit)

Par exemple sur 6 bits, pour obtenir le codage de -5 :

000101 (codage de 5 en binaire sur 6 bits)
111010 (complément à un, on inverse chaque bit)
111011 (on ajoute 1) : représentation de -5 en complément à deux sur 6 bits.

Avec n bits, ce système permet de représenter les nombres entre -2^{n-1} et $2^{n-1}-1$, et tous les nombres négatifs ont le bit de poids fort (b_{n-1}) égal à 1.
Les entiers naturels de 0 à $2^{n-1}-1$ représentent les entiers relatifs positifs ou nul correspondants et les entiers naturels de 2^{n-1} à 2^n-1 représentent les entiers relatifs négatifs de -2^{n-1} à -1.
Les machines actuelles utilisent 32 ou 64 bits (4 ou 8 octets).

Pour simplifier, avec un octet, soit huit bits, on peut représenter les entiers naturels n de 0 jusqu'à 255 et donc les entiers relatifs de $-2^7 = -128$ à $2^7 - 1 = 127$; les nombres n de 0 à 127, (de 0000 0000 à 0111 1111 en base deux), servent à représenter les entiers relatifs positif ou nul r avec $r = n$ et les nombres n de 128 à 255, (de 1000 0000 à 1111 1111 en base deux), représentent les entiers relatifs négatifs r avec $r = n - 256$.

Le nombre $n = 127$ représente l'entier relatif $r = 127$;

Le nombre $n = 127$ représente l'entier relatif $r = 127$;

le nombre $n = 128$ représente l'entier relatif
$r = 128 - 256 = -128$;

Le nombre $n = 127$ représente l'entier relatif $r = 127$;

le nombre $n = 128$ représente l'entier relatif
$r = 128 - 256 = -128$;

le nombre $n = 255$ représente l'entier relatif
$r = 255 - 256 = -1$.

Remarque 1 : un principe équivalent sur le cercle trigonométrique parcouru dans le sens direct de degré en degré est de n'utiliser que des nombres positifs, soit, dans l'ordre, 0, 1, 2, . . . , 178, 179, 180, 181, . . . , 358, 359 (ceci correspond aux entiers naturels), ou d'utiliser aussi des nombres négatifs, soit, dans l'ordre, 0, 1, 2, . . . , 178, 179, -180, -179, . . . , -3, -2, -1.

Remarque 1 : un principe équivalent sur le cercle trigonométrique parcouru dans le sens direct de degré en degré est de n'utiliser que des nombres positifs, soit, dans l'ordre, 0, 1, 2, . . . , 178, 179, 180, 181, . . . , 358, 359 (ceci correspond aux entiers naturels), ou d'utiliser aussi des nombres négatifs, soit, dans l'ordre, 0, 1, 2, . . . , 178, 179, -180, -179, . . . , -3, -2, -1.

Remarque 2 : tous les nombres négatifs ont leur bit de poids fort égal à 1, alors que les nombres positifs ont leur bit de poids fort égal à 0.

Remarque 3 : avec ce codage, si on peut faire effectuer par une machine l'addition de deux nombres et le passage à l'opposé d'un nombre, on peut lui faire effectuer toutes les opérations classiques.

Remarque 3 : avec ce codage, si on peut faire effectuer par une machine l'addition de deux nombres et le passage à l'opposé d'un nombre, on peut lui faire effectuer toutes les opérations classiques.
Exemple sur un octet : 5 est codé par 00000101, -3 est codé par 11111101, donc l'addition 5+(-3) se fait en ajoutant les nombres bit à bit, soit (1)00000010 qui est bien 2. (On ne garde que 8 bits.)

Remarque 3 : avec ce codage, si on peut faire effectuer par une machine l'addition de deux nombres et le passage à l'opposé d'un nombre, on peut lui faire effectuer toutes les opérations classiques.
Exemple sur un octet : 5 est codé par 00000101, -3 est codé par 11111101, donc l'addition 5+(-3) se fait en ajoutant les nombres bit à bit, soit (1)00000010 qui est bien 2. (On ne garde que 8 bits.)

Remarque 4 : si on avait choisi de coder les négatifs avec simplement un bit de signe (0 pour + et 1 pour -), alors -3 serait codé par 10000011 et l'addition 5+(-3) bit à bit donnerait 10001000 qui coderait -8 !

En Python, les entiers sont représentés par le type `int` (integer).

Les plupart des processeurs calculent avec des nombres binaires de taille limitée à 32 bits ou 64 bits pour les processeurs les plus récents. Les langages de programmation en général font de même et les calculs sur ces entiers, gérés directement par le processeur, sont très rapides.

Python permet de calculer avec des entiers de taille illimitée ; il gère lui-même l'encodage des nombres pour des tailles plus grandes afin de ne transmettre au processeur que des nombres sur 32 bits et le programmeur n'a pas à s'en occuper. La taille des entiers utilisés par Python n'est donc limitée que par la quantité de mémoire disponible dans l'ordinateur. Mais plus les nombres sont grands et plus les calculs vont être décomposés par l'interpréteur en calculs multiples sur des nombres plus simples et donc rallonger le temps de traitement.

Définition

La norme IEEE 754 (Standard for Binary Floating-Point Arithmetic) définit la représentation de nombres réels appelés **flottants** ou **nombres à virgule flottante**.
Un nombre réel est représenté par un nombre décimal, la meilleure approximation possible, qui s'écrit sous la forme s $m \times 2^n$ ou s est le signe du nombre, n son exposant et m sa mantisse. Cette représentation est semblable à la notation scientifique des calculatrices, sauf qu'elle est en base deux et non en base dix. En faisant varier l'exposant n, on fait "flotter" la virgule.

Le signe s est $+$ ou $-$, l'exposant n est un entier relatif et la mantisse m est un nombre à virgule, compris entre 1 inclus et 2 exclu. (En notation scientifique, c'est un nombre à virgule compris entre 1 inclus et 10 exclu). Par exemple le réel 10 s'écrit $+1,25 \times 2^3$; le réel $-0,1875$ s'écrit $-1,5 \times 2^{-3}$.

Les flottants IEEE peuvent être codés sur 32 bits ("simple précision" avec 1 bit pour le signe, 8 bits pour l'exposant et 23 bits pour la mantisse) ou 64 bits ("double précision" avec 1 bit pour le signe, 11 bits pour l'exposant et 52 bits pour la mantisse).

Si on code les nombres sur 64 bits :

- Le signe $+$ est représenté par 0 et le signe $-$ par 1.

- L'exposant n est un entier relatif compris entre -1022 et 1023 ; on le représente par l'entier naturel $n + 1023$, (exposant décalé), qui est compris entre 1 et 2 046. Les deux entiers naturels 0 et 2 047 sont réservés pour des situations exceptionnelles ($+\infty$, $-\infty$, NaN, etc).

- La mantisse *m* est un nombre binaire à virgule compris entre 1 inclus et 2 exclu, comprenant 52 chiffres apres la virgule. Comme cette mantisse est comprise entre 1 et 2, elle a toujours un seul chiffre avant la virgule et ce chiffre est toujours un 1 ; il est donc inutile de le représenter et on utilise les 52 bits pour représenter les 52 chiffres apres la virgule.

Dans l'ordinateur les nombres seront alors codés par :

" signe exposant (décalé) mantisse (tronquée) "

ces trois parties étant codées en binaire.

Exemple 1 : $10 = +1,25 \times 2^3$

Le signe est $+$ donc le premier bit vaut 0 ;

l'exposant est 3 donc l'exposant décalé est $3 + 1023 = 1026$ et en binaire cela donne 100 0000 0010 ;

la mantisse est $1,25$ qui s'écrit en binaire 1,01. On garde la partie décimale 01 et on complète avec des zéros.

Le codage de 10 est donc : 0 100 0000 0010 0100 0000 . . . 0000.

Exemple 2 : $-0,1875 = -1,5 \times 2^{-3}$

Le signe est $-$ donc le premier bit vaut 1 ;

l'exposant est -3 donc l'exposant décalé est $-3 + 1023 = 1020$ et en binaire cela donne 011 1111 1100 ;

la mantisse est $1,5$ qui s'écrit en binaire 1,1. On garde la partie décimale 1 et on complète avec des zéros.

Le codage de $-0,1875$ est donc : 1 011 1111 1100 1000 0000 ... 0000.

En Python, les nombres réels ou nombres en virgule flottante sont du type `float`. Pour qu'une donnée numérique soit considérée par Python comme étant du type `float`, il suffit que sa formulation contienne un point décimal ou une puissance de 10 par exemple, 3.14 ou 1. ou .0001 ou 1e12 .

Précautions d'emploi :

Les calculs en virgule flottante sont pratiques, mais attention :

Précautions d'emploi :

Les calculs en virgule flottante sont pratiques, mais attention :

- leur précision est limitée ; cela se traduit par des arrondis qui peuvent s'accumuler de façon gênante. (Pour cette raison, en comptabilité les calculs ne sont pas effectués en virgule flottante).

- il n'y a aucun nombre dans un voisinage suffisament petit de zéro : une représentation flottante possède une plus petite valeur négative, une plus petite valeur positive, et entre les deux le zéro, mais aucune autre valeur !
  Que vaut dans l'ordinateur la plus petite valeur positive divisée par 2 ?

  Pour comparer deux nombres flottants on vérifiera donc que la valeur absolue de leur différence est inférieure ou égale à une précision donnée.

- il y a un nombre maximal (max=1.7976931348623157e+308), donc une puissance de deux maximale (1023) et une puissance de dix maximale (308).

```
>>> 1.0e308
```

```
>>> 1.0e308
1e+308
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
inf
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
inf
>>> 2.0**1023
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
inf
>>> 2.0**1023
8.98846567431158e+307
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
inf
>>> 2.0**1023
8.98846567431158e+307
>>> 2.0**1024
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
inf
>>> 2.0**1023
8.98846567431158e+307
>>> 2.0**1024
Traceback (most recent call last) :
  File "<pyshell#22>", line 1, in <module>
    2.0**1024
OverflowError : (34, 'Result too large')
```

```
>>> 1.0e308
1e+308
>>> 1.0e309
inf
>>> 2.0**1023
8.98846567431158e+307
>>> 2.0**1024
Traceback (most recent call last) :
  File "<pyshell#22>", line 1, in <module>
    2.0**1024
OverflowError : (34, 'Result too large')
```

Remarque : un problème d'overflow est responsable de l'autodestruction de la fusée Ariane 5, le 4 juin 1996, 39 secondes après le décollage ; la forte accélération de la fusée a provoqué un dépassement de capacité lors du calcul des position et vitesse.

Il peut être tentant de réorganiser des expressions en virgule flottante comme on le ferait d'expressions mathématiques. Cela n'est cependant pas anodin, car les calculs en virgule flottante, contrairement aux calculs sur les réels, ne sont pas associatifs.

Il peut être tentant de réorganiser des expressions en virgule flottante comme on le ferait d'expressions mathématiques. Cela n'est cependant pas anodin, car les calculs en virgule flottante, contrairement aux calculs sur les réels, ne sont pas associatifs. Par exemple, dans un calcul en flottants IEEE double précision, $(10^{50} + 1) - 10^{50}$ ne donne pas 1, mais 0. La raison est que $10^{50} + 1$ est approximé par 10^{50}.