Représentation des nombres en informatique décembre 2023

1 Représentation des entiers naturels

Les bases les plus utilisées sont la base 10 (vie courante), et en informatique, la base 2 (écriture binaire) et la base 16 (écriture hexadécimale).
Soit $n \in \mathbb{N}$ un entier naturel, sa représentation en base b avec $b > 1$, notée $n_{(b)}$, est la suite de symboles ou de chiffres représentant des nombres entre 0 et $b-1$ telle que

$$n_{(b)} = a_p a_{p-1} \cdots a_0 \Leftrightarrow n = \sum_{k=0}^{p} a_k b^k$$

Ainsi le nombre deux cent dix neuf s'écrit

- en base dix [2,1,9] avec des symboles choisis dans $\{0,1,2,3,4,5,6,7,8,9\}$
- en base deux [1,1,0,1,1,0,1,1] avec les symboles 0 et 1
- en base seize [D,B] avec les symboles choisis parmi $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$
  A représentant 10 et F représentant 15 .

L'écriture en base b avec n symboles permet de représenter tous les entiers de 0 (en base 2 tous les bits égaux à 0) à $b^n - 1$ (en base 2 tous les bits égaux à 1).

Savoir-faire : représenter en base b un entier naturel donné en base 10
Algorithme
Données : n, b
$i \leftarrow 0$
tant que $n \neq 0$ **faire**
    $a_i \leftarrow$ reste de la division euclidienne de n par b
    $n \leftarrow$ quotient de la division euclidienne de n par b
    $i \leftarrow i + 1$
Résultat : la représentation $a_i \cdots a_1 a_0$

2 Représentation des entiers relatifs

Un nombre entier relatif est toujours représenté par une chaîne de n bits. Deux choses varient : la longueur n de la chaîne et la convention de représentation du signe.

- **convention de la valeur signée**
  Une chaîne de n bits : $a_{n-1}a_{n-2}\cdots a_1 a_0$ est interprétée de la façon suivante : le premier bit (ou chiffre binaire) a_{n-1} est égal à 0 si l'entier est positif, 1 si l'entier est négatif. Les autres bits $a_{n-2}\cdots a_1 a_0$ codent alors la valeur absolue.
  Conséquences :
  — 0 a deux représentations 0000 et 1000 dans le cas d'une représentation avec 4 bits.
  — on représente les entiers compris entre $-(2^{n-1}-1)$ et $(2^{n-1}-1)$, intervalle symétrique
  — il faut deux algorithmes d'addition : un pour deux nombres de même signe, un autre pour l'addition de nombres de signes opposés.
- **convention du complément à 2**
  On représente un entier relatif par un entier naturel.
  Si on utilise des mots de n bits, on peut alors représenter les entiers relatifs x compris entre -2^{n-1} et $2^{n-1}-1$.
  Si x est positif ou nul, on le représente comme x, et sa représentation en base 2 sera un mot de n bits dont le premier bit est 0.
  Si x est strictement négatif, on représente x comme l'entier naturel $x + 2^n$, qui est compris entre 2^{n-1} et $2^n - 1$, sa représentation en base 2 sera un mot de n bits dont le premier bit vaut 1.

Pour obtenir la représentation en complément à 2 en n bits d'un nombre strictement négatif :
On cherche l'écriture en base 2 avec n bits de sa valeur absolue,
on inverse les bits (les 1 deviennent des 0 et les 0 deviennent des 1),
et on ajoute 1.

Exemple avec -10 écrit en 6 bits.
$10_{(2)} = 001010$ car $10 = 8 + 2 = 2^3 + 2^1 = 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$
on inverse les chiffres : on obtient 110101.
on ajoute 1 (attention aux retenues ! !) 110110. Ceci est la représentation en complément à 2 de -10.
C'est l'écriture en base 2 avec 6 bits du nombre $-10+2^6 = -10+64 = 54 = 32+16+4+2 = 2^5+2^4+2^2+2^1$

Exercice : trouver les représentations décimales des entiers relatifs dont les représentations binaires sur 8 bits sont 0001 0111 et 1000 1100.

En Python 3, il n'y a pas de limite pour un entier.

3 Représentation des nombres à virgule

La notation binaire permet de représenter des nombres à virgule, les nombres à droite de la virgule correspondant à des puissances négatives de 2.
Exemple : $10,1011$ en base 2 s'écrit $1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4}$

Un nombre est représenté sous la forme $sm2^n$, (norme IEEE 754) où

- s est le signe (0 pour +, 1 pour -),
- n est l'exposant (entier relatif)
- m est la mantisse.

3.1 Simple précision

En simple précision , un flottant normalisé s'écrit par une suite de 32 bits

$$\left(\underbrace{\varepsilon}_{\text{signe}}, \underbrace{p_7, \cdots, p_0}_{pseudo-exposant} , \underbrace{a_1, a_2, \cdots, a_{23}}_{pseudo-mantisse} \right)$$

qui représente le nombre $x = (-1)^{\varepsilon} \left(1 + \sum_{j=1}^{23} a_j 2^{-j} \right) 2^{p-127}$

avec :
$p \in [\![1, 254]\!]$, p est appelé pseudo-exposant, et il est codé en base 2 sur sur 8 bits par $p_7, \cdots, p_0$.

$a_1, a_2, \cdots, a_{23}$ est la pseudo-mantisse puisque la mantisse est $1.a_1, a_2, \cdots, a_{23}$

Exercice

1. Déterminer l'écriture décimale du nombre dont la représentation en simple précision avec la norme IEEE 754 est
   1 1000 0100 00010010000000000000000
2. Déterminer la représentation en simple précision avec la norme IEEE 754 du nombre décimal 14.75

3.2 Double précision

En double précision , un flottant normalisé s'écrit par une suite de 64 bits

$$\left(\underbrace{\varepsilon}_{\text{signe}}, \underbrace{p_{10}, \cdots, p_0}_{pseudo-exposant} , \underbrace{a_1, a_2, \cdots, a_{52}}_{pseudo-mantisse} \right)$$

qui représente le nombre $x = (-1)^{\varepsilon}\left(1+\sum_{j=1}^{52} a_j 2^{-j}\right)2^{p-1023}$

avec :
$p \in [\![1, 2046]\!]$, p est appelé pseudo-exposant, et il est codé en base 2 sur sur 11 bits par $p_{10}, \cdots, p_0$.

$a_1, a_2, \cdots, a_{52}$ est la pseudo-mantisse puisque la mantisse est $1.a_1, a_2, \cdots, a_{52}$

Exemples :
•le réel 20 s'écrit $+1.25 * 2^4$
Le signe est + donc le premier bit vaut 0.
L'exposant est 4 donc l'exposant décalé est 4+1023=1027 soit 100 0000 0011 en binaire.
La mantisse est 1,25 qui s'écrit en binaire 1,01. On garde la partie décimale soit 01 et on complète avec des zéros, .
Le codage de 20 est donc 0 100 0000 0011 0100 0000 ...0000

•le réel - 0.375 s'écrit $-1.5 * 2^{-2}$
Le signe est - donc le premier bit vaut 1.
L'exposant est -2 donc l'exposant décalé est -2+1023=1021 soit 011 1111 1101 en binaire.
La mantisse est 1,5 qui s'écrit en binaire 1,1. On garde la partie décimale soit 1 et on complète avec des zéros, .
Le codage de -0.375 est donc 1 011 1111 1101 1000 ...0000

•le réel 0.1 s'écrit $+1.6 * 2^{-4}$
Le signe est + donc le premier bit vaut 0.
L'exposant est -4 donc l'exposant décalé est -4+1023=1019 soit 011 1111 1011 en binaire.
La mantisse est 1,6 qui n'a pas d'écriture exacte en binaire.
$0.2 = 3 * 2^{-4} + 3 * 2^{-8} + 3 * 2^{-12} + 3 * 2^{-16} + ...$
Son écriture en binaire est donc 0.0011 0011 0011....
Comme $0.2 = 1.6 * 2^{-3}$, on obtient que la pseudo-mantisse de 1.6 est 1001 1001...1001 1001....
On arrondit, comme le 53ème chiffre est 1, on arrondit par valeur supérieure, 1001,1 est arrondi à 1010, donc 0.1 est codé par 0 011 1111 1011 1001 1001 ... 1001 1010.

3.3 nombres spéciaux en simple précision

- "+0" est le nombre $(-1)^0\left(1+\sum_{j=1}^{23} 0 * 2^{-j}\right)2^{-127}$, représenté par le nombre à 32 bits avec tous les bits valant 0, qui représente 0 (0^+) .
- "−0" est le nombre $(-1)^1\left(1+\sum_{j=1}^{23} 0 * 2^{-j}\right)2^{-127}$, représenté par le nombre à 32 bits dont le premier bit vaut 1 , et tous les autres bits valent 0, et qui représente 0 (0^-).
- "$+inf$ " est le nombre $(-1)^0\left(1+\sum_{j=1}^{23} 0 * 2^{-j}\right)2^{128}$, représenté par le nombre à 32 bits dont le premier bit vaut 0, les 8 suivants valent 1, et les 23 derniers bits valent 0 . Il représente $+\infty$.
- "$-inf$ " est le nombre $(-1)^1\left(1+\sum_{j=1}^{23} 0 * 2^{-j}\right)2^{128}$, représenté par le nombre à 32 bits dont le premier bit vaut 1, dont le premier bit vaut 0, les 8 suivants valent 1, et les 23 derniers bits valent 0 .Il représente $-\infty$.
- les nombres NaN (Not a Number) , de la forme $(-1)^{\varepsilon}\left(1+\sum_{j=1}^{23} a_j * 2^{-j}\right)2^{128}$, où l'un au moins des a_j est non nul. Ces nombres représentent des valeurs erronées (racine carrée d'un réel négatif par exemple).

4 Dépassement de capacité, problèmes de précision et arrondis

Si on dépasse l'exposant 1023 en double précision , on utilise les nombres spéciaux $+\infty$ et $-\infty$.
Si on est « trop »proche de 0 (exposant inférieur à -1022), soit le nombre est arrondi à 0, soit il se produit une erreur.
Arrondis :
avec Python
In[5] : $1 + 2 * * - 54 - 1$
Out[5] : 0.0
On calcule la somme $1 + 2^{-54}$ en premier, arrondie à 1.
En revanche :
In[6] : $1 - 1 + 2 * * - 54$
Out[6] : $5.551115123125783e - 17$
On calcule la somme 1-1 en premier, on obtient donc un arrondi de 2^{-54}
Principale conséquence : un test du type $a == b$ n'a pas de sens si a et b sont des nombres non entiers, on écrira plutôt une condition de la forme $abs(a - b) < epsilon$, avec $epsilon$ une valeur proche de zéro , choisie en fonction du problème à traiter.

5 Exercices

Exercice 1. Vrai ou Faux

1. Si l'écriture binaire d'un entier naturel se termine par n zéros, alors cet entier est divisible âr 2^n.
2. En base 8, on utilise les chiffres de 1 à 8.
3. Avec n bits, les entiers représentables à l'aide de leur écriture binaire sont inférieurs à 2^n
4. L'entier 170 s'écrit AA en hexadécimal.
5. Si les entiers non signés sont codés sur des mots de 4 bits, alors 1101+0101=0000
6. Avec des mots de 4 bits , on peut représenter tous les entiers relatifs de -8 à 8 bornes incluses.
   Les entiers non signés sont codés sur 8 bits. Il y a un dépassement de capacité si on calcule la somme de deux nombres commençant par 1.

Exercice 2. Trouver la représentation en base 16 des nombres : 3718, 54920 et 482919.

Exercice 3. C'est en $\underline{111\ 1011\ 0001}_2$ qu'a été transmis le premier message sur Internet. Exprimer ce nombre en base 10.

Exercice 4. Trouver le nombre flottant représenté en double précision par
1100 0100 0110 1001 0011 1100 0011 1000 0000 0000 0000 0000 0000 0000 0000 0000

Exercice 5. Vrai ou Faux

1. Tous les décimaux appartenant à $[1, 2]$ ont une représentation exacte en flottant.
2. Avec des mots de n bits, on peut représenter 2^n flottants au maximum.
3. Avec la représentation signe, mantisse, exposant, en binaire, pour effectuer une division par deux, on retire à la mantisse le bit de poids faible.
4. Avec la représentation signe, mantisse, exposant, en binaire, on obtient l'opposé d'un nombre en changeant le bit de poids faible.
5. Avec 1 bit pour le signe, 3 pour l'exposant et 6 pour la mantisse, on peut coder 512 flottants positifs.

6. L'expression 1+2.0**(-5)==1 a la valeur True si le codage des flottants utilise une mantisse codée sur 4 bits.
7. L'expression 1+10.0**(-3)==1 a la valeur True si le codage des flottants utilise une mantisse codée sur 10 bits.

Exercice 6. On représente des nombres réels avec une écriture signe, mantisse, exposant en binaire . On utilise six bits $b_0b_1b_2b_3b_4b_5$.
Le bit b_0 représente le signe : 0 pour +, 1 pour -.
Les bits $b_1b_2b_3$ représentent la mantisse tronquée.
Les bits b_4b_5 représentent l'exposant.
Par exemple l'écriture 010010 représente un nombre positif , sa mantisse vaut 1,100 soit 1,5 en décimal et l'exposant est 10 soit 2.
le nombre représenté est donc $1,5 * 2^2$ soit 6.

1. Quel est le plus petit nombre positif et le plus grand nombre que l'on peut écrire ainsi ?
2. Combien de nombres appartenant à l'intervalle $[1, 2[$ peut-on écrire ?
3. Quel est le successeur de 2 ?

Exercice 7. On considère les suites numériques (u_n) et (f_n) telles que
$u_0 = 2$, $u_1 = -4$ et $u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_{n-1}u_n}$
$f_n = \dfrac{4 * 5^{n+1} - 3 * 6^{n+1}}{4 * 5^n - 3 * 6^n}$.

1. Ecrire une fonction python U d'argument n qui renvoie u_n.
2. Ecrire une fonction python F d'argument n qui renvoie f_n.
3. Comparer les résultats pour n compris entre 0 et 30.
4. On peut démontrer (on ne le fera pas ici) que pour tout n, $u_n = f_n$. Laquelle des fonctions U ou F paraît donner les résultats les plus corrects ?