Les Flottants

Introduction

Les nombres à virgule peuvent aussi être représentés en base 2.

Maintenant, on ne peut stocker dans la mémoire d'une machine qu'un nombre fini de décimales, donc certains nombres, comme $\frac{1}{3} \approx 0,33333....$, seront représentés par des valeurs approchées.

Il existe deux codages possibles en machine : le **codage en virgule fixe**, et le **codage en virgule flottante**.

• L'idée du **codage en virgule fixe** est de retenir un nombre fixe de chiffres après la virgule.

• Dans le cas du **codage en virgule flottante**, on retient un nombre fixe de **chiffres significatifs** : beaucoup de chiffres après la virgule si le nombre est petit et beaucoup de chiffres avant la virgule si un nombre est grand.

On utilise principalement le second aujourd'hui.

Codage en virgule flottante

Ce codage s'inspire de l'**écriture scientifique des nombres décimaux** qui se compose d'un *signe*, d'un nombre décimal **m**, appelé *mantisse*, compris dans l'intervalle $[1\ ;\ 10[$, et d'un entier relatif **n** appelé *exposant*.

Ce qui donne un nombre de la forme : $\pm m \times 10^n$.

Par exemple :

$345 = 3,45 \times 10^2$;

$-3\ 723,451 = -3,723\ 451 \times 10^3$;

$0,03\ 21 = 3,21 \times 10^{-2}$

Norme IEEE 7554

La représentation des nombres flottants a été définies dans la **norme internationale IEEE 754**.

Elle se décompose en trois parties : un signe s, une mantisse m et un exposant n, mais en base 2.
Ce nombre aura la forme : $(-1)^s m \times 2^n$.

On utilise principalement deux formats : sur 32 bits appelé *simple précision* ou sur 64 bits appelé *double précision*.

- Le signe s est codé sur un bit : 0 pour + et 1 pour - ;
- La mantisse appartient à l'intervalle $[1\ ;\ 2[$;
- L'exposant n est décalé d'une valeur d qui dépend du format choisi (32 ou 64 bits), afin de coder des exposants négatifs et positifs.

Simple précision (32 bits)

Si on code le nombre à virgule sur 32 bits, on utilise le bit de poids fort pour le signe s, les 8 bits suivants sont réservés pour coder l'exposant décalé n+127 et les 23 derniers pour la mantisse.

Avec 8 bits pour l'exposant décalé, on peut coder des entiers de 0 à 255, ce qui permet de représenter des exposants de -126 à 127. (On n'utilise pas les valeurs 0 et 255 qui sont réservées pour des nombres particuliers.)

La mantisse étant comprise dans l'intervalle $[1\ ;2[$, elle représente un nombre de la forme 1,.... c'est à dire un nombre qui commence nécessairement par 1. Par conséquent, on ne va coder que les chiffres après la virgule.

Exemple :

Écrivons le nombre $175,125$ en virgule flottante, en simple précision (32 bits) :

- 1ère étape :

On écrit la partie entière en binaire :

$$175_{10} = 10101111_2$$

On écrit la partie décimale en binaire :

Remarque :

Les chiffres à droite de la virgule vont être écrit avec des puissances de 2 négatives : 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}....

$$0,125_{10} = \frac{1}{2^3} = 0 \times \frac{1}{2^1} + 0 \times \frac{1}{2^2} + 1 \times \frac{1}{2^3} = 0,001_2$$

Méthode pratique : on multiplie par 2 pour décaler la virgule vers la gauche, et on prend la partie entière. On recommence avec le reste, jusqu'à obtenir 1.

$$0,125 \times 2 = \underline{0},25$$

$$0,25 \times 2 = \underline{0},5$$

$$0,5 \times 2 = \underline{1}$$

Ainsi on obtient, $175,125_{10} = 10101111,001_2$

Vérifions : $1 \times 2^7 + 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-3} = 175,125$

- 2ème étape :

On normalise l'écriture en base 2 :

$$10101111,001_2 = 1,0101111001 \times 2^7$$

On a le début de la mantisse 0101111001 que l'on complète si nécessaire avec des 0 pour avoir 23 bits.

Ainsi $m = 01011110010000000000000$

Il reste a coder l'exposant en binaire, en n'oubliant pas le décalage de 127.

Il faut donc coder le nombre $n + 127 = 7 + 127 = 134$

or $134_{10} = 10000110_2$

Le nombre étant positif, on commence par 0.

Finalement, en simple précision, $175,125$ se code :

$$\underbrace{0}_{s}\underbrace{10000110}_{n+127}\underbrace{01011110010000000000000}_{m}$$

On place le bit de signe en premier, puis le codage de l'exposant décalé, et enfin le codage de la mantisse.

Remarques :

Cette écriture est difficile à lire, alors on l'écrit en hexadécimal pour la raccourcir :

Méthode : on coupe l'écriture binaire en paquets de 4 bits ($2^4 = 16$) que l'on traduit en hexadécimal.

La représentation hexadécimale de $175,125_{10}$ est $432F2000_{16}$.

SYNTHESE :

Écriture décimale	binaire flottant	hexadécimal
$175,125$	01000011001011110010000000000000	$432F2000$

Remarques :

• En simple précision on peut représenter des nombres décimaux compris approximativement dans l'intervalle $[10^{-38}\ ;\ 10^{38}]$.

• On peut vérifier l'écriture en virgule flottante sur le site internet :

www.h-schmidt.net/FloatConverter/IEEE754.html

Double précision (64 bits)

Dans ce format, le décalage de l'exposant est de $2^{10} = 1024$; l'exposant décalé est codé sur 11 bits et la partie à droite de la virgule de la mantisse est codée sur 52 bits.

Remarque :

• En double précision on peut représenter des nombres décimaux compris approximativement dans l'intervalle $[10^{-308}\ ;\ 10^{308}]$.

Valeurs particulières

On utilise les valeurs 0 et 255 des exposants décalés pour coder des **valeurs particulières** :
Le **nombre zéro** que l'écriture des nombres flottants ne permet pas de représenter ; l'**infini** qui est utilisé pour représenter des dépassements de capacité et **Nan** (Not a Number) qui permet de représenter les résultats d'opérations invalides comme 5/0...

Signe	Exposant	la partie à droite de la virgule de la mantisse	valeur spéciale
0	0	0	$+0$
1	0	0	-0
0	255	0	$+\inf$
1	255	0	$-\inf$
0	255	$\neq 0$	NaN

Exercice 1 :

1. Donner la représentation en virgule flottante (sous forme hexadécimal) des nombres suivants, en simple précision.

   $-6,53125$; 129 ; $-0,15625$; $210,5$

2. Convertir en décimale les nombres suivants, représentés en simple précision, et codés en hexadécimal.

   $42E48000$; $3F880000$; $C7F00000$

Exercice 2 :

1. Donner la représentation en virgule flottante du nombre $13,62$ en simple précision. Que remarque t-on ?
2. Faire de même avec $\frac{1}{3}$.

Remarque :

Certain nombre ont une partie décimale qui ne peut s'écrire sous la forme d'une somme **finie** de puissances de 2 ; il faudra alors tronquer la représentation en virgule flottante. Ainsi, ce n'est pas toujours la valeur exacte du nombre qui est codée, mais une valeur approchée.

Comparaison de flottants

Exercice 3 :

1. Donner la représentation en virgule flottante de $0,1$.
2. En déduire la représentation en virgule flottante de $0,2$.
3. Que donne la somme $0,1 + 0,2$?
4. Exécuter ce même calcul dans l'interpréteur Python.
   Que remarque t-on ? Pourquoi un tel résultat ?

Remarque :

Plutôt que de tester l'égalité entre deux flottants, il est préférable d'écrire un test d'inégalité entre les deux valeurs.

par exemple, tapez :

```
x = 0.1 +0.2
y= 0.3
abs(x - y) < 1e-12
```

(on vérifie que l'écart entre les deux valeurs est inférieur à 10^{-12}).

Exercice 4 :

Écrire un programme en Python qui permet d'obtenir la représentation en virgule flottante d'un nombre décimal non nul appartenant à l'intervalle $]-1\ ;\ 1[$.

Conclusion

L'utilisation d'un nombre limité de chiffres binaires, que ce soit pour les nombres à virgule ou les nombres entiers est la source de bugs conséquents.

Exemples :

• Lors du premier conflit Etats-Unis/Irak en 1991, les américains disposaient d'antimissiles pour intercepter les missiles Irakiens. Ceux-ci disposaient d'une horloge interne émettant un signal toutes les 0,1 secondes.

Or la représentation de 0,1 en flottants n'est pas exacte et cette petite erreur, au bout de 100 heures à conduit à un décalage de l'horloge interne d'un missile de 0,34 secondes. Mais au vu de la grande vitesse du missile, cela a engendré un décalage de 500 m, et l'antimissile a raté le missile Irakien, qui a provoqué la mort de 28 personnes...

On parle alors de **propagation de l'erreur**.

• Si on travaille sur une machine qui code les entiers sur 8 bits, un simple calcul du type $53 + 100$ donne un résultat (153) qui ne peut être codé car il n'est pas compris entre -127 et 126.

On parle alors de **dépassement de capacité** ou overflow.