Représentation des entiers

Quentin Fortier

March 2, 2022

Question

Comment représenter un nombre?

Question

Comment représenter un nombre?

1ère méthode, avec des bâtons (système unaire) :

Question

Comment représenter un nombre?

1ère méthode, avec des bâtons (système unaire) :

Prend beaucoup de temps pour écrire un grand nombre : il faut n caractères pour écrire un nombre n.

<u>Idée</u>: regrouper par «paquets».

Ainsi, le nombre $1234 = 10^3 + 2 \times 10^2 + 3 \times 10 + 4$ signifie:

- \bullet 1 paquet de 10^3
- 3 paquets de 10
- 4 paquets de 1

<u>Idée</u>: regrouper par «paquets».

Ainsi, le nombre $1234 = 10^3 + 2 \times 10^2 + 3 \times 10 + 4$ signifie:

- \bullet 1 paquet de 10^3
- extstyle 2 paquets de 10^2
- 3 paquets de 10
- 4 paquets de 1

L'unique raison d'avoir regroupé par paquets de 10 est que nous avons 10 doigts sur nos mains.

Théorème (admis)

Soit $b\in\mathbb{N}^*.$ Tout entier $n\in\mathbb{N}^*$ peut s'écrire de façon unique sous la forme :

$$n = n_{p-1} \times b^{p-1} + \dots + n_1 \times b + n_0$$

où $0 \le n_i < b$, $\forall i$.

Théorème (admis)

Soit $b\in\mathbb{N}^*.$ Tout entier $n\in\mathbb{N}^*$ peut s'écrire de façon unique sous la forme :

$$n = n_{p-1} \times b^{p-1} + \dots + n_1 \times b + n_0$$

où $0 \le n_i < b$, $\forall i$.

Définition

La suite $n_{p-1},...,n_1,n_0$ est l'écriture de n en base b, et on écrit :

$$n = n_{p-1}...n_1 n_0$$

Il est facile de passer d'une base quelconque à la base décimale:

$$121_3 = 1 \times 3^2 + 2 \times 3 + 1 = 16 (= 16_{10})$$

Il est facile de passer d'une base quelconque à la base décimale:

$$121_3 = 1 \times 3^2 + 2 \times 3 + 1 = 16 \ (= 16_{10})$$
$$1011_2 =$$

Il est facile de passer d'une base quelconque à la base décimale:

$$121_3 = 1 \times 3^2 + 2 \times 3 + 1 = 16 (= 16_{10})$$

 $1011_2 = 11$

$$100...00_2 = 11...11_2 = 11...11_2$$

$$\underbrace{100...00}_{\ell}_2 = 2^{\ell}$$

$$\underbrace{11...11}_{\ell}_{2} =$$

$$\underbrace{100...00_{2}}_{\ell} = 2^{\ell}$$

$$\underbrace{11...11}_{\ell}_{2} = \sum_{k=0}^{\ell-1} \frac{2^{k} - 1}{2 - 1} = 2^{\ell} - 1$$

- Si la base n'est pas spécifiée, il s'agit de la base 10 (décimale).
- 2 Les ordinateurs utilisent la base 2 (binaire): 1 si il y a passage de courant, 0 sinon.
- On rencontre aussi la base 16 (hexadécimale) en informatique. Comme il n'y a que 10 chiffres, on est obligé d'utiliser des lettres:

$$A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$$

On peut additionner et multiplier deux nombres en base \emph{b} comme vous avez l'habitude.

Exercice

Que vaut $10111_2 + 101_2$?

Question

Écrire une fonction to_base10 telle que to_base10(b, L) renvoie le nombre dont la liste des chiffres en base b est L.

Question

Écrire une fonction to_base10 telle que to_base10(b, L) renvoie le nombre dont la liste des chiffres en base b est L.

```
def to_base10(b, L):
    n = 0
    for i in range(len(L)):
        n += L[i]*(b**i)
    return n
```

Question

Écrire une fonction to_base10 telle que to_base10(b, L) renvoie le nombre dont la liste des chiffres en base b est L.

```
def to_base10(b, L):
    n = 0
    for i in range(len(L)):
        n += L[i]*(b**i)
    return n
```

Question : Comment éviter le calcul de puissance ?

Les puissances à calculer sont $b^0, b^1, b^2...$ On peut stocker b^i dans une variable p et la multiplier par b pour calculer la puissance suivante :

```
def to_base10(b, L):
    n = 0
    p = 1 # va stocker les puissances successives de b
    for i in range(len(L)):
        n += L[i]*p
        p *= b
    return n
```

On veut maintenant passer de la base décimale à une autre base.

Pour passer un nombre n de la base 10 à la base 2, il faut trouver les n_i tels que:

$$n = n_{p-1} \times 2^{p-1} + \dots + n_1 \times 2 + n_0$$

On veut maintenant passer de la base décimale à une autre base.

Pour passer un nombre n de la base 10 à la base 2, il faut trouver les n_i tels que:

$$n = n_{p-1} \times 2^{p-1} + \dots + n_1 \times 2 + n_0$$

$$\Leftrightarrow n = \underbrace{2}_{b} \times \underbrace{(n_{p-1} \times 2^{p-2} + \dots + n_{1})}_{q} + \underbrace{n_{0}}_{r}$$

On veut maintenant passer de la base décimale à une autre base.

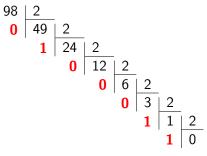
Pour passer un nombre n de la base 10 à la base 2, il faut trouver les n_i tels que:

$$n = n_{p-1} \times 2^{p-1} + \dots + n_1 \times 2 + n_0$$

$$\Leftrightarrow n = \underbrace{2}_{b} \times \underbrace{(n_{p-1} \times 2^{p-2} + \dots + n_{1})}_{q} + \underbrace{n_{0}}_{r}$$

Ainsi, n_0 est le reste de la division de n par 2, n_1 est le reste de la division de q par 2, et ainsi de suite...

Passons 98 de la base 10 à la base 2 :



Passons 98 de la base 10 à la base 2 :

Donc $98 = 1100010_2$ (on lit les restes à l'envers).

Exercice

Écrire une fonction from_base10 prenant une base b et un nombre n et renvoyant la liste des chiffres de n en base b.

Exercice

Écrire une fonction from_base10 prenant une base b et un nombre n et renvoyant la liste des chiffres de n en base b.

```
def from_base10(b, n):
    L = []
    while n != 0:
        L.append(n % b)
        n = n // b
    return L
```

Dans un ordinateur, on ne dispose que d'une mémoire finie. Les entiers positifs sont stockés en base 2, avec un nombre maximum de chiffres (souvent 32 ou 64).

Question

Quels sont les entiers que l'on peut représenter avec p bits?

Dans un ordinateur, on ne dispose que d'une mémoire finie. Les entiers positifs sont stockés en base 2, avec un nombre maximum de chiffres (souvent 32 ou 64).

Question

Quels sont les entiers que l'on peut représenter avec p bits?

Les entiers de
$$\underbrace{0...0}_p{}_2=0$$
 à $\underbrace{1...1}_p{}_2=2^p-1.$

Soit un total de 2^p entiers différents (on a deux choix pour chaque bit donc il y a bien 2^p possibilités).

Si les entiers sont stockés sur p bits, les éventuels chiffres qui « dépasseraient » sont supprimés.

Si les entiers sont stockés sur p bits, les éventuels chiffres qui « dépasseraient » sont supprimés.

Par exemple, si les entiers sont stockés sur 4 bits alors :

Si les entiers sont stockés sur p bits, les éventuels chiffres qui « dépasseraient » sont supprimés.

Par exemple, si les entiers sont stockés sur 4 bits alors :

Plus généralement, si les entiers sont stockés sur p bits alors en ajoutant 1 au plus grand entier représentable $(\underbrace{1...1}_2 = 2^p - 1)$, on obtient 0.

Le nombre de bits utilisés pour stocker un entier dépend:

- 1 du langage de programmation
- 2 du processeur (32 bits, 64 bits...)
- **3** ...

En Python il n'y a pas de problème de dépassement possible : le nombre de bits utilisés augmente automatiquement quand un entier devient trop grand.

Stockage des entiers signés

Pour l'instant, nous n'avons écrit que des nombres positifs.

Comment coder des nombres négatifs?

Stockage des entiers signés

Si on écrit nos nombres sur p bits, une première solution est de réserver le premier bit pour le signe:

- Un 0 signifie un nombre positif.
- Un 1 signifie un nombre négatif.

Le reste des bits est utilisé pour écrire le nombre en valeur absolue en base 2.

Stockage des entiers signés

La solution précédente a un gros désavantage: l'addition et la multiplication ne marchent plus.

```
+ 0 0 0 0 0 0 1 1
+ 0 0 0 0 0 1 1 1
```

=

Stockage des entiers signés

La solution précédente a un gros désavantage: l'addition et la multiplication ne marchent plus.

	1	0	0	0	0	0	1	1
+	0	0	0	0	0	0	1	1

Stockage des entiers signés

La solution précédente a un gros désavantage: l'addition et la multiplication ne marchent plus.

Le codage par complément à 2 (sur p bits) représente tous les entiers $n\in\{-2^{p-1},...,2^{p-1}-1\}$:

- Si $n \ge 0$, on représente n par son écriture en base 2.
- ② Si n < 0, on représente n par l'écriture en base 2 de $n + 2^p$

Le codage par complément à 2 (sur p bits) représente tous les entiers $n\in\{-2^{p-1},...,2^{p-1}-1\}$:

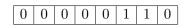
- Si $n \ge 0$, on représente n par son écriture en base 2.
- $oldsymbol{2}$ Si n<0, on représente n par l'écriture en base 2 de $n+2^p$

6 est représenté sur 8 bits par:

Le codage par complément à 2 (sur p bits) représente tous les entiers $n\in\{-2^{p-1},...,2^{p-1}-1\}$:

- Si $n \ge 0$, on représente n par son écriture en base 2.
- ② Si n < 0, on représente n par l'écriture en base 2 de $n + 2^p$

6 est représenté sur 8 bits par:

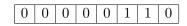


-6 est représenté sur 8 bits par l'écriture en base 2 de $-6+2^8=250$:

Le codage par complément à 2 (sur p bits) représente tous les entiers $n\in\{-2^{p-1},...,2^{p-1}-1\}$:

- Si $n \ge 0$, on représente n par son écriture en base 2.
- ② Si n < 0, on représente n par l'écriture en base 2 de $n + 2^p$

6 est représenté sur 8 bits par:



-6 est représenté sur 8 bits par l'écriture en base 2 de $-6+2^8\ = 250$:

La somme vaut bien 0.

Entiers signés sur 8 bits :

0	1	1	1	1	1	1	1	
0	1	1	1	1	1	1	0	$=2^7-2=126$
	:	:	:	:	:	:	:	:
0	0	0	0	0	0	0	1	=1
0	0	0	0	0	0	0	0	=0
1	1	1	1	1	1	1	1	= -1
1	1	1	1	1	1	1	0	=-2
:	:	:	:	:	:	:	:	:
1	0	0	0	0	0	0	1	· ·
1	0	0	0	0	0	0	0	$=-2^7=-128$

Entiers signés sur 8 bits :

0	1	1	1	1	1	1	1	
0	1	1	1	1	1	1	0	$=2^7-2=126$
:	:	:	:	:	:	:	:	:
0	0	0	0	0	0	0	1	=1
0	0	0	0	0	0	0	0	=0
1	1	1	1	1	1	1	1	= -1
1	1	1	1	1	1	1	0	=-2
:	:	:	:	:	:	:	:	:
1	0	0	0	0	0	0	1	· '
1	0	0	0	0	0	0	0	$=-2^7=-128$

On remarque que le 1er bit indique le signe.

Soit $n\geq 0$ un entier écrit en base 2 sur p bits et \tilde{n} son complémentaire (où les bits 0 et 1 sont inversés).

Alors : $n + \tilde{n} =$

Soit $n\geq 0$ un entier écrit en base 2 sur p bits et \tilde{n} son complémentaire (où les bits 0 et 1 sont inversés).

$$\mathsf{Alors}:\ n+\tilde{n}=\underbrace{1...1}_{p}{}_{2}$$

Soit $n\geq 0$ un entier écrit en base 2 sur p bits et \tilde{n} son complémentaire (où les bits 0 et 1 sont inversés).

Alors:
$$n + \tilde{n} = \underbrace{1...1}_{n}_{2} = 2^{p} - 1$$

Donc la représentation par complément à 2 de -n est $2^p-n=\tilde{n}+1.$

Si n<0, le codage par complément à 2 sur p bits de n peut donc s'obtenir de la façon suivante:

- Écrire |n| en base 2.
- Inverser les 0 et les 1.
- Ajouter 1.

Si n<0, le codage par complément à 2 sur p bits de n peut donc s'obtenir de la façon suivante:

- Écrire |n| en base 2.
- ② Inverser les 0 et les 1.
- 3 Ajouter 1.

Exemple avec n = -6, p = 8:

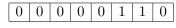
 $oldsymbol{0} - n$ est codé par

Si n<0, le codage par complément à 2 sur p bits de n peut donc s'obtenir de la façon suivante:

- Écrire |n| en base 2.
- ② Inverser les 0 et les 1.
- 3 Ajouter 1.

Exemple avec n = -6, p = 8:

$$\bullet$$
 $-n$ est codé par



Si n<0, le codage par complément à 2 sur p bits de n peut donc s'obtenir de la façon suivante:

- Écrire |n| en base 2.
- 2 Inverser les 0 et les 1.
- 3 Ajouter 1.

Exemple avec n = -6, p = 8:

- ② On inverse: 1 1 1 1 1 0 0 1

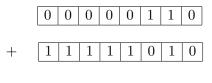
Si n<0, le codage par complément à 2 sur p bits de n peut donc s'obtenir de la façon suivante:

- Écrire |n| en base 2.
- ② Inverser les 0 et les 1.
- 3 Ajouter 1.

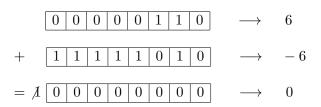
Exemple avec n = -6, p = 8:

- ② On inverse: 1 1 1 1 1 0 0 1
- **③** On ajoute 1: 1 1 1 1 0 1 0

Avec le codage par complément à 2, on peut additionner et multiplier des nombres, en « oubliant » les dépassements :



Avec le codage par complément à 2, on peut additionner et multiplier des nombres, en « oubliant » les dépassements :



Question

Écrire une fonction complement2(n, p) renvoyant le codage en complément à 2 sur p bits de n.