Représentation des entiers

I Écriture en base b d'un entier

2021 signifie dans notre écriture **décimale** $2\times10^3+0\times10^2+2\times10^1+1\times10^0$. C'est ce que l'on appelle une écriture en base 10. Mais d'autres bases peuvent être utilisées. En informatique, on utilise, outre la base 10, les bases 2 (**binaire**) et 16 (**hexadécimal**) [1].

Définition (Écriture dans une base)

Si b est un entier supérieur ou égal à 2, une **écriture en base** b d'un nombre $x \geq 1$ est de la forme

$$x = r_n b^n + r_{n-1}b^{n-1} + \cdots + r_1 b + r_0 \text{ que l'on notera } \overline{r_n r_{n-1}\ldots r_1 r_0}^{(b)}.$$

Les r_i sont les **chiffres** et sont compris entre 0 et $b-1$. r_n étant le premier chiffre, il est non nul par définition.

Exercice 1

1. Jusifier que 17 s'écrit $\overline{10001}^{(2)}$ en base 2.
2. Comment s'écrit 47 en base 5 ?
3. Que vaut $\overline{131}^{(6)}$?
4. Que vaut $\overline{B7}^{(16)}$?

Entraînement 1

1. Comment s'écrit 47 en base 4 ?
2. Comment s'écrit 255 en base 16 ?
3. Que vaut $\overline{88}^{(8)}$?

1. On utilise alors A,B,C,D,E et F pour représenter les valeurs 10, 11, 12, 13, 14 et 15.

Pour déterminer l'écriture d'un entier en base b, on peut utiliser deux algorithmes, suivant que l'on commence à calculer le premier chiffre en partant de la gauche ou le dernier.

Exercice 2 : De la gauche vers la droite

Écrivez 181 en binaire :

128	64	32	16	8	4	2	1

L'inconvénient de cette méthode est qu'il faut déjà calculer les puissances de 2 mais elle est assez commode à la main pour de petits nombres. Par contre, informatiquement, on utilisera plutôt l'algorithme suivant (très simple à coder et efficace).

Exercice 3 : De la droite vers la gauche

On remarque que si $x = r_n b^n + r_{n-1} b^{n-1} + \cdots + r_1 b + r_0$ alors $x = b\left(r_n b^{n-1} + r_{n-1} b^{n-2} + \cdots + r_1\right) + r_0$. Comme $0 \leq r_0 < b$, r_0 est le reste de la division euclidienne de x par b. Ainsi, pour calculer r_0, on fait cette division euclidienne sous la forme $x = bq_0 + r_0$ puis on itère sur q_0 ce qui donne $q_0 = bq_1 + r_1$ donc $x = b(bq_1 + r_1) + r_0 = q_1 b^2 + r_1 b + r_0$ et on continue sur q_1 ...

Déterminer l'écriture binaire de 77.

128	64	32	16	8	4	2	1

Entraînement 2

En utilisant successivement les deux méthodes vues dans les exercices précédent, écrire 205 en binaire.

Il est souvent important de savoir combien de chiffres seront nécessaires pour écrire un nombre en binaire. Pour cela, on a besoin de la fonction $\log_2$.

Propriété

Il existe une fonction mathématique $\log_2$ qui vérifient les propriétés suivantes :
- $\log_2$ est croissante sur $]0, +\infty[$
- $\log_2(2) = 1$
- $\log_2(a \times b) = \log_2(a) + \log_2(b)$
- $\log_2(a/b) = \log_2(a) - \log_2(b)$
- $\log_2(a^n) = n \log_2(a)$

On peux alors énoncer la propriété :

Propriété

Le nombre de chiffres de l'écriture de x en base 2 est $\lfloor \log_2(x) \rfloor + 1$.

Remarque : $\lfloor a \rfloor$ désigne la partie entière du nombre a. C'est le plus grand entier relatif inférieure ou égale à a. Par exemple : $\lfloor 3,26 \rfloor = 3$, $\lfloor -7 \rfloor = -7$ et $\lfloor -2,3 \rfloor = -3$.

Preuve : Admis

Exercice 4 On donne $10^3 < 1024 = 2^{10}$. En déduire une majoration du nombre de chiffres binaires nécessaires pour écrire 10^{100}.

Exercice 5

Les nombres a et b sont respectivement représentés avec α et β chiffres en binaire. Donner le nombre de chiffres nécessaires pour représenter $a + b$ et $a \times b$.

Entraînement 3

Le nombres a est représenté avec α chiffres en binaire. Donner le nombre de chiffres nécessaires pour représenter a^n.

Exercice 6

Écrire une fonction `base(n,b)` qui renvoie une liste donnant l'écriture en base $b \leqslant 10$ de n.

```
>>> base(77, 2)
[1, 0, 0, 1, 1, 0, 1]
```

Entraînement 4

Même consigne qu'à l'exercice précédent mais en fixant le nombre de chiffres de l'écriture :

```
>>> base_complet(77, 2, 8)
[0, 1, 0, 0, 1, 1, 0, 1]
```

Exercice 7

Même consigne qu'à l'exercice 6 mais renvoyer une chaine. On pourra avoir $b > 10$.

```
>>> base_chaine(255, 16)
'FF'
```

Entraînement 5 

Même consigne qu'à l'exercice précédent mais en fixant le nombre de chiffres de l'écriture :

Pour réaliser l'opération inverse, on utilise l'**algorithme de Hörner** :

Imaginons que l'on dispose séquentiellement des chiffres d'un grand nombre. Par exemple 4, puis 5, puis 0, puis 7, puis 3, enfin 5. Comment construire pas à pas le nombre correspondant ?

Exercice 8

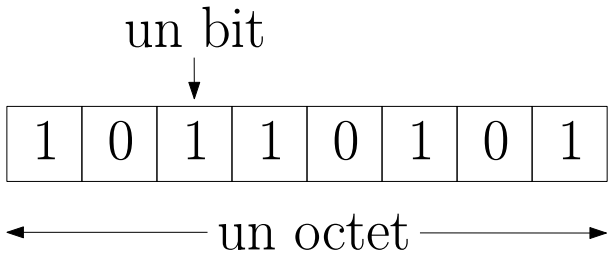
En déduire une fonction `valeur(L, b):` qui calcule le nombre représenté par la liste de chiffres `L` dans la base b.

```
>>>valeur([1, 0, 1, 1, 0], 2)
22
```

II Une première approche pour les négatifs

Les mémoires d'ordinateur sont constituées de "cases" prenant la valeur 0 ou la valeur 1 (il s'agit d'un circuit électronique ouvert ou fermé). Une telle case est appelée un **bit** (binary digit = chiffre binaire).

Les ordinateurs travaillent en regroupant les bits par paquets. Un paquet de 8 bits est appelé **octet** (byte).

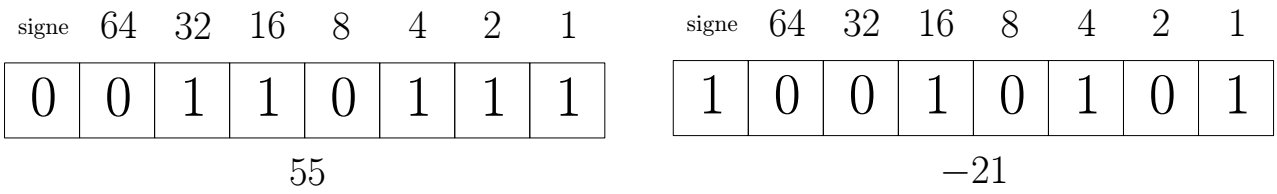


Ainsi, pour stocker un entier, il est naturel d'utiliser sa représentation binaire. L'exemple ci-dessus est donc le stockage en mémoire du nombre 181.

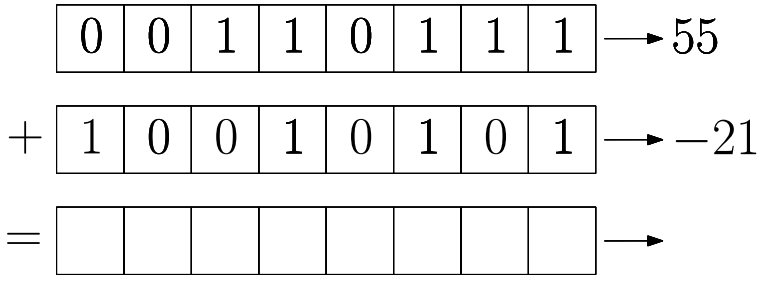
Si l'on travaille sur un octet, on peut donc représenter les entiers entre 0 et $1+2+\cdots+2^7 = 2^8-1 = 255$.

D'une manière générale, si les nombres sont codés sur n bits, on pourra représenter les entiers entre 0 et 2^n-1.

Mais il faut évidemment pouvoir travailler avec des entiers relatifs. Une première idée consiste à réserver le premier bit (celui le plus à gauche, dit "bit de poids fort") au signe de l'entier (par exemple 0 pour positif et 1 pour négatif) et les $n-1$ autres bits à la valeur absolue. Pour $n=8$, on peut alors représenter les entiers entre -127 et 127 :



Cette représentation des entiers pose plusieurs problèmes. Par exemple, le nombre 0 admet deux représentations différentes : 00000000 et 10000000 et il y a problème pour l'addition si l'on applique l'algorithme usuel (bit par bit de la droite vers la gauche avec propagation de retenue) :



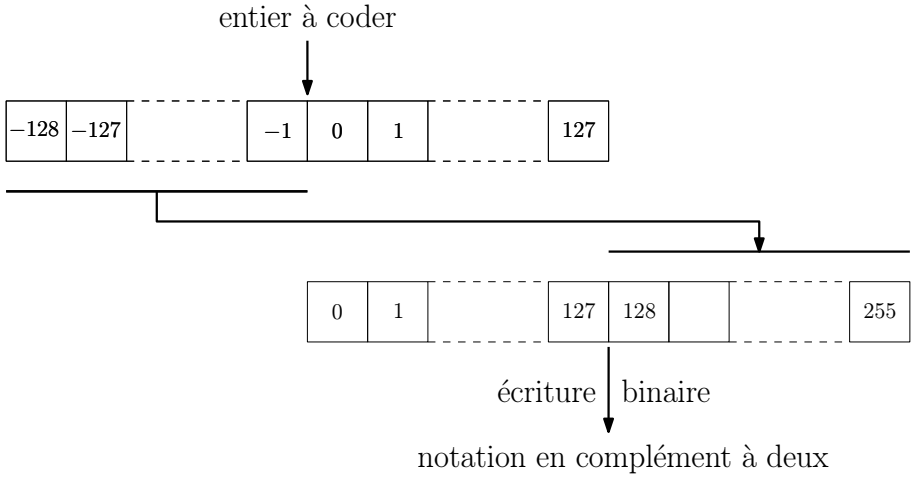
On préfère donc utiliser une autre représentation :

III La notation en complément à 2

Commençons par regarder le cas particulier des entiers codés sur un octet. On décide de représenter les entiers entre -128 et 127 (au lieu de 0 à 255) de la manière suivante :

— si $x \in [\![0, 127]\!]$ alors x est codé par son écriture binaire sur un octet.

— si $x \in [\![-128, -1]\!]$ alors x est codé par l'écriture binaire de $x + 256$ sur un octet.



Ainsi, on fait correspondre à chaque entier de $[\![-128, 127]\!]$ un entier de $[\![0, 255]\!]$ qui est ensuite codé sur 8 bits par sa représentation binaire. On remarquera que 0 admet alors une seule écriture : 00000000.

Voici par exemple la notation en complément à deux de -43 et 117 :

```
>>> base_complet(-43+256, 2, 8)
[1, 1, 0, 1, 0, 1, 0, 1]
>>> base_complet(117, 2, 8)
[0, 1, 1, 1, 0, 1, 0, 1]
```

On remarque que le signe d'un entier est immédiat à déterminer via sa notation en complément à deux : il est donné par le bit de poids fort. Le nombre est positif ou nul lorsque le bit de poids fort vaut 0 et négatif sinon.

Nous pouvons maintenant généraliser à la notation en complément à deux sur n bits. On représente les entiers entre -2^{n-1} et $2^{n-1} - 1$ de la manière suivante :

— si $x \in [\![0, 2^{n-1} - 1]\!]$ alors x est codé par son écriture binaire sur n bits.

— si $x \in [\![-2^{n-1}, -1]\!]$ alors x est codé par l'écriture binaire sur n bits de $x + 2^n$.

Voici deux codes permettant de passer d'un entier à sa notation en complément à deux et réciproquement :

— Notation en complément à 2 de n sur N bits :

```
def complement_a_deux(n, N):
    if n >= 0:
        return base_complet(n, 2, N)
    else:
        return base_complet(n+2**N, 2, N)
```

— Opération inverse :

```python
def valeur_cads(L):
    N = len(L)
    if L[0] == 0:
        return valeur(L[1:], 2)
    else:
        return valeurle(L, 2) - 2**N
```

La notation en complément à deux peut également être vue de la manière suivante :

La liste de bits $[r_{n-1}, \dots, r_0]$ code l'entier $-r_{n-1}2^{n-1} + r_{n-2}2^{n-2} + \cdots + r_0$

On voit bien alors que r_{n-1} donne le signe de l'entier.

Exercice 9 On travaille sur 8 bits. Quel entier est codé par :
— 01110011 :
— 11000001 :

Exercice 10 En utilisant cette remarque, écrire une fonction permettant de passer de la notation en complément à deux à l'entier représenté par cette notation :

Dépassement de capacité : supposons que l'on travaille avec une représentation sur 8 bits. On peut donc représenter les entiers de $[\![-128, 127]\!]$. Mais si l'on additionne deux tels entiers, il se peut que le résultat sorte de cet intervalle. Il se produit alors un dépassement de capacité (overflow ou underflow).

Si l'on veut additionner deux nombres, on commence par déterminer leur notation en complément à deux puis on applique l'algorithme habituel d'addition bit à bit avec retenue et on revient à la notation décimale. Le faire pour 55 et -21 puis 54 et 81 :

128	64	32	16	8	4	2	1

128	64	32	16	8	4	2	1

128	64	32	16	8	4	2	1

128	64	32	16	8	4	2	1

128	64	32	16	8	4	2	1

128	64	32	16	8	4	2	1

En Python 2, les entiers ont une taille limitée et dès qu'il y a dépassement de cette taille, ils changent de type et deviennent des entiers longs. En Python 3, il y a un seul type d'entiers et ils ont une taille illimitée.

La limite de la taille des entiers en Python 2 dépend de la machine (32 bits ou 64 bits).

Exercice 11 Expliquez le code suivant, exécuté en Python 2 :

```
>>> 2**30
1073741824
>>> 2**31-1
2147483647L
```

```
>>> x = sum(2**k for k in range(31))
>>> x
2147483647
>>> x+1
2147483648L
```

Entraînement 6  On travaille sur 16 bits avec la notation en complément à deux.

1. Quels sont les entiers que l'on peut représenter ?
2. Trouver la notation en complément à 2 de 21213 et -15155.
3. Quels sont les entiers représentés par 1000000000001111 et 0101010101000001 ?
4. Additionner ces deux nombres via leur représentation puis vérifier.

Exercices supplémentaires

Exercice 12

Écrire en Python une fonction `sommeBinaire(L, M)` qui renvoie la liste des chiffres de l'écriture binaire de la somme des nombres représentés en binaire par `L` et `M`.

Exercice 13

Écrire en Python une fonction `diffBinaire(L, M)` qui renvoie la liste des chiffres de l'écriture binaire de la différence des nombres représentés en binaire par `L` et `M`.

Exercice 14

Écrire en Python une fonction `prodBinaire(L, M)` qui renvoie la liste des chiffres de l'écriture binaire du produit des nombres représentés en binaire par `L` et `M`.

Exercice 15

Écrire une fonction `complement(b)` qui calcule le complément à 2 d'un nombre binaire donné sous forme d'un tableau de taille quelconque contenant uniquement de 0 et des 1. La fonction renverra un tableau de même taille.