Codage de l'information

Laurent Noé-Léopold Weinberg
paternité : Francesco De Comité

Licence 1 Mathématiques Informatique Semestre 1
Université de Lille
Faculté des Sciences et Technologies

13 novembre 2025

Opérations sur les bits

Introduction

Opération arithmétiques

Chacun connaît les opérations arithmétiques usuelles sur les nombres entiers :

- addition (`+`),
- soustraction (`-`),
- multiplication (`*`),
- division (`//`) et
- exponentiation (`**`).

De nouvelles opérations

La représentation binaire des nombres entiers va permettre de définir de nouvelles opérations qui s'appuient sur les opérateurs logiques comme la conjonction, la disjonction, le ou-exclusif et la négation.

Opérations Booléennes

Rappel

- En cours de programmation, vous avez découvert la notion de type booléen.
- Ce type est constitué des deux valeurs de vérité : *Faux*, *Vrai*.

En python

Ces deux valeurs sont notées respectivement `False`, `True`.

Dans les tables de vérités

Ces deux valeurs seront notées respectivement 0 et 1

Opérations Booléennes

Le type booléen n'est pas venu seul : Des opérations sur les valeurs de ce types ont été définies.

Rappel

- La *négation*, la *conjonction*, la *disjonction* correspondent respectivement aux opérateurs *non*, *et*, le *ou*.
- En mathématique, en logique, en informatique, le *ou* est *inclusif*.
- Le *ou exclusif*, c'est soit l'un soit l'autre mais pas les deux.

en python

les opérateurs logiques sur les booléens sont `not`, `and`, `or`. Il n'y a pas vraiment d'opérateur pour le ou exclusif sur les booléens (ce n'est pas indispensable car le `!=` fait très bien l'affaire.)

Non

symbole mathématique

La notation mathématique du *non* est : $\neg$

Table de vérité du *non*

a	$\neg a$
0	1
1	0

Ou

symbole mathématique

La notation mathématique du *ou* est : $\vee$

Table de vérité du *ou*

$\vee$	0	1
0	0	1
1	1	1

Et

symbole mathématique

La notation mathématique du *et* est : $\wedge$

Table de vérité du *et*

$\wedge$	0	1
0	0	0
1	0	1

Ou exclusif

symbole mathématique

La notation mathématique du *ou exclusif* est : $\oplus$

Table de vérité du *ou exclusif*

$\oplus$	0	1
0	0	1
1	1	0

Généralisation du non

On généralise les opérations précédentes pour des listes de bits. Le *non bit à bit* est défini pour liste de bits. Le résultat est obtenu faisant la liste de la négation de chaque bit de la liste.

exemple

42 s'écrit sur huit bits de la manière suivante :

7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0

la négation de 42 bit à bit sur 8 bits s'écrit :

7	6	5	4	3	2	1	0
1	1	0	1	0	1	0	1

Généralisation du *ou*

Le *ou bit à bit* est défini pour deux listes de bits de même longueur. Le résultat est obtenu faisant la liste des résultats du *ou logique* sur chaque bit des deux opérandes en respectant les poids.

exemple

les nombres décimaux 42 et 71 s'écrivent en binaire sur huit bits de la manière suivante :

	7	6	5	4	3	2	1	0
x=	0	0	1	0	1	0	1	0

	7	6	5	4	3	2	1	0
y =	0	1	0	0	0	1	1	1

le *ou bit à bit* de 42 et 71 sur 8 bits s'écrit :

	7	6	5	4	3	2	1	0
$x \vee y =$	0	1	1	0	1	1	1	1

Généralisation du *et*

Le *et bit à bit* est défini pour deux listes de bits de même longueur. Le résultat est obtenu faisant la liste des résultats du *et logique* sur chaque bit des deux opérandes en respectant les poids.

exemple

les nombres décimaux 42 et 71 s'écrivent en binaire sur huit bits de la manière suivante :

	7	6	5	4	3	2	1	0
x=	0	0	1	0	1	0	1	0

	7	6	5	4	3	2	1	0
y =	0	1	0	0	0	1	1	1

le *et bit à bit* de 42 et 71 sur 8 bits s'écrit :

	7	6	5	4	3	2	1	0
$x \wedge y =$	0	0	0	0	0	0	1	0

Généralisation du *ou exclusif*

Le *ou exclusif bit à bit* est défini pour deux listes de bits de même longueur. Le résultat est obtenu faisant la liste des résultats du *ou exclusif logique* sur chaque bit des deux opérandes en respectant les poids.

exemple

les nombres décimaux 42 et 71 s'écrivent en binaire sur huit bits de la manière suivante :

	7	6	5	4	3	2	1	0
x=	0	0	1	0	1	0	1	0

	7	6	5	4	3	2	1	0
y =	0	1	0	0	0	1	1	1

le *ou exclusif bit à bit* de 42 et 71 sur 8 bits s'écrit :

	7	6	5	4	3	2	1	0
$x \oplus y =$	0	1	1	0	1	1	0	1

loi de composition interne

Definition

loi de composition interne : On appelle loi de composition interne sur un ensemble E toute fonction de $E \times E$ dans E.

exemple

- sur l'ensemble des entiers naturels, l'addition est une loi de composition interne.
- sur l'ensemble des vecteurs de $\mathbb{R}^3$, le produit vectoriel est un loi de composition interne.
- sur l'ensemble des entiers naturels, l'exponentiation est un une loi de composition interne.

contre-exemple

- sur l'ensemble des vecteurs du plan, le produit scalaire n'est pas une loi de composition interne.

différentes notations

Il existe plusieurs manières de représenter un appel de fonction.

notation *préfixée* :

La fonction est placée devant ses paramètres. On dit alors que la notation est préfixée.
C'est la notation généralement utilisée en mathématique.
exemple :

$$\underset{\text{symbole de fonction}}{\ln} \quad \underset{\text{paramètre}}{2}$$

notation *postfixée* :

La fonction est placée après ses paramètres. On dit alors que la notation est postfixée.
exemple : la factorielle. la notation est postfixée.

$$\underset{\text{paramètre}}{6} \quad \underset{\text{symbole de fonction}}{!}$$

notation (suite)

notation *infixée* :

la fonction est placée entre ses paramètres.
Si la fonction est une loi de composition interne,

- la fonction est désignée par un symbole qu'on appelle *opérateur*,
- le premier paramètre est appelé *opérande gauche*,
- le second est appelé *opérande droit*,
- et une séquence formée par un opérande gauche, un opérateur et un opérande droit, s'appelle une *opération*.

exemple : l'addition.

$$\underbrace{\underset{\substack{\text{paramètre}\\ \text{opérande gauche}}}{40} \quad \underset{\substack{\text{symbole de fonction}\\ \text{opérateur}}}{+} \quad \underset{\substack{\text{paramètre}\\ \text{opérande droit}}}{2}}_{\text{opération}}$$

Propriétés des opérations bit à bit

soit T un entier naturel strictement positif fixé.
Dans la suite, on travaille sur l'ensemble des entiers naturels qui s'écrivent avec au plus T bits.
Les opérations bits à bits qui ont été définies, sont des lois de composition interne de l'ensemble des entiers naturels de taille binaire au plus T.
Comme toutes les lois de composition internes, elles sont susceptibles de posséder des propriétés.

Associativité

définition

Une loi de composition interne $\top$ (ici pour truc) sur un ensemble E est dite *associative* lorsque :

$$\forall (x, y, z) \in E^3 \qquad (x \top y) \top z = x \top (y \top z)$$

exemple

La composition de fonction est associative.

contre-exemple

Dans l'ensemble des vecteurs de l'espace réél à 3 dimensions, le produit vectoriel n'est pas associatif.

$$(\overrightarrow{i} \wedge \overrightarrow{j}) \wedge \overrightarrow{j} = \overrightarrow{k} \wedge \overrightarrow{j} = -\overrightarrow{i}$$
$$\overrightarrow{i} \wedge (\overrightarrow{j} \wedge \overrightarrow{j}) = \overrightarrow{i} \wedge \overrightarrow{0} = \overrightarrow{0}$$

Commutativité

définition

Une loi de composition interne $\top$ sur un ensemble E est dite *commutative* lorsque :

$$\forall (x, y) \in E^2 \qquad x \top y = y \top x$$

exemples

- L'addition sur l'ensemble des entiers naturels est commutative,
- la multiplication sur l'ensemble des entiers naturels est commutative.

contre exemple

- la concaténation sur l'ensemble des mots sur un alphabet donné (dès que l'alphabet comporte plus d'un caractère) n'est pas commutative.

Élément neutre

Définition

on dit que e est un élément neutre pour la loi de composition interne $\top$ définie sur un ensemble E lorsque :

$$\forall x \in E \qquad e \top x = x \top e = x$$

Exemples

- On considère l'ensemble des applications de d'un ensemble X dans lui même est muni de la composition de fonction notée $\circ$. Cette opération est une loi de composition interne qui possède un élément neutre. L'élément neutre est la fonction d'identité noté *id*.
- On considère la multiplication sur l'ensemble des entiers naturel. Cette loi de composition interne possède l'élément 1 comme élément neutre.

Élément absorbant

Définition

on dit que a est un élément absorbant pour la loi de composition interne $\top$ définie sur un ensemble E lorsque :

$$\forall x \in E \qquad a \top x = x \top a = a$$

Exemples

- l'ensemble des entiers naturels $\mathbb{N}$ est muni de la multiplication notée $\times$, 0 est un élément absorbant.
- Dans le compactifié d'Alexandroff de $\mathbb{R}$,noté $\hat{\mathbb{R}}$, c'est-à-dire $\mathbb{R}$ union un singleton contenant un symbole supplémentaire noté ∞, on peut étendre la loi de composition interne qu'est l'addition en posant $\infty + \infty = \infty$ et $\forall x \in \mathbb{R} \qquad \infty + x = \infty = x + \infty$. Par construction l'élément ∞ est absorbant.

Symétriques

e est un élément neutre pour la loi de composition interne $\top$ définie sur un ensemble E

Définition

On dit que y est le symetrique de x si par définition $y \top x = x \top y = e$

exemple

dans l'ensemble des entiers relatifs $\mathbb{Z}$, -3 est un symétrique de 3 pour l'addition.

contre-exemple

On considère l'ensemble des polynômes sur une indeterminée X , f la dérivation, g la fonction qui a un polynôme associe la primitive de ce polynôme qui s'annulle en 0. $f \circ g = id$ mais $g \circ f \neq id$ donc g n'est pas le symétrique de f pour la composition.

Propriétés des opérations bit à bit (suite)

Comme le cours de codage n'est pas un cours de mathématiques, nous allons juste faire la liste de quelques propriétés, et surtout, on laisse les preuves aux mathématiciens ou en exercice pour les étudiants interessés.

catalogue non exhaustif de propriétés

- Le *et bit à bit* est commutatif et associatif, il posséde un élément neutre $2^T - 1$, un élément absorbant 0. En outre, le *et bit à bit* est distributif par rapport au *ou bit à bit*
- le *ou bit bit* est commutatif et associatif, il possède un élément neutre 0 et, un élément absorbant $2^T - 1$.
- le *ou exclusif bit à bit* est commutatif et associatif, possède un élément neutre, chaque élément possède un symétrique pour cette loi, lui même, cela forme une loi de groupe.

Applications des opérations bit à bits

- On peut utiliser le *et bit à bit* pour mettre à 0 les bits situés à certaines positions tout en gardant inchangés les bits qui se trouvent aux autres positions.
- On peut utiliser le *ou bit à bit* pour mettre à 1 les bits situés à certaines positions tout en gardant inchangés les bits qui se trouvent aux autres positions.
- On peut utiliser le *xor bit à bit* pout complémenter les bits situés à certaines positions tout en gardant inchangés les bits qui se trouvent aux autres positions.

Annuler des bits

exemple

le nombre x est un entier représenté sur 8 bits. Je ne veux conserver que les bits de poids 4 et de poids 5 et mettre à zéro tous les autres bits. J'utilise un masque formé de 0 aux positions que je veux annuler et de 1 aux positions que je veux conserver. puis j'effectue un *et bit à bit*.

	7	6	5	4	3	2	1	0
x=	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

	7	6	5	4	3	2	1	0
m=	0	0	1	1	0	0	0	0

	7	6	5	4	3	2	1	0
$x \wedge m =$	0	0	b_5	b_4	0	0	0	0

Mettre des bits à un

exemple

le nombre x est un entier représenté sur 8 bits. Je ne veux conserver que les bits de poids 4 et de poids 5 et mettre à un tous les autres bits. J'utilise un masque formé de 1 aux positions que je veux mettre à un et de 0 aux positions que je veux conserver. puis j'effectue un *ou bit à bit*.

	7	6	5	4	3	2	1	0
x=	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

	7	6	5	4	3	2	1	0
m=	1	1	0	0	1	1	1	1

	7	6	5	4	3	2	1	0
$x \vee m =$	1	1	b_5	b_4	1	1	1	1

Complémenter des bits

exemple

le nombre x est un entier représenté sur 8 bits. Je ne veux conserver que les bits de poids 4 et de poids 5 et complémenter tous les autres bits. J'utilise un masque formé de 1 aux positions que je veux complémenter et de 0 aux positions que je veux conserver. puis j'effectue un *ou exclusif bit à bit*.

	7	6	5	4	3	2	1	0
$x =$	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

	7	6	5	4	3	2	1	0
$m =$	1	1	0	0	1	1	1	1

	7	6	5	4	3	2	1	0
$x \oplus m =$	$\overline{b_7}$	$\overline{b_6}$	b_5	b_4	$\overline{b_3}$	$\overline{b_2}$	$\overline{b_1}$	$\overline{b_0}$

où $\overline{b_i}$ signifie la négation du bit b_i.

Définition

On définit deux fonctions supplémentaires : les décalages à gauche et à droite qu'on appelle *shr* et *shl*, les noms sont des des acronymes formés sur les mots anglais *shift* qui veut dire décaler, et *right* et *left*.

Remarque

Comme on travaille sur des nombres de taille binaire au plus T, lorsqu'on fait un décalage le bit qui sort est perdu, si c'est un 0 ce n'est pas grave, si c'est un 1 on dit qu'il y a un débordement. Le bit qui rentre reçoit la valeur 0.

décalage à gauche (vers les poids forts)

	7	6	5	4	3	2	1	0
x=	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

	7	6	5	4	3	2	1	0
shl x=	b_6	b_5	b_4	b_3	b_2	b_1	b_0	0

remarque

Lorsqu'il n'y a pas de débordement, le décalage à gauche correspond à une multiplication par 2

décalage vers la droite

Quand on fait un décalage vers la droite, le bit de poids faible est perdu, le bit de poids fort est mis à 0.

	7	6	5	4	3	2	1	0
$x =$	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

	7	6	5	4	3	2	1	0
$shr\, x =$	0	b_7	b_6	b_5	b_4	b_3	b_2	b_1

Remarque

le décalage vers la droite revient à calculer le quotient dans la division euclidienne par deux. Le bit qui sort est le reste...

fonctions itérées

X désigne un ensemble. f une application de X dans X. On définit une famille de fonctions notée $\left(f^{(n)}\right)_{n\in\mathbb{N}}$ par :

$$f^{(0)} = id$$

$$\forall n \in \mathbb{N} \qquad f^{(n+1)} = f \circ f^{(n)} = f^{(n)} \circ f = \underbrace{f \circ f \circ \ldots \circ f}_{n+1 \text{ exemplaires de } f}$$

definition

On appelle n^{e} itérée de f la fonction $f^{(n)}$ ainsi définie.

remarque

$$\forall x \in X \qquad f^{(n)}(x) = \underbrace{f\left(f\left(f\left(\ldots f\left(f\left(}_{n \text{ exemplaires de } f\left(} x \overbrace{\right)\right)\ldots\right)\right)}^{n \text{ parenthèses fermantes}}$$

opérateurs de décalage

Definition

- on note $x \ll n$ le resultat de n décalage à gauche itérés du nombre x c'est-à-dire $shl^{(n)}(x)$
- on note $x \gg n$ le resultat de n décalage à droite itérés du nombre x c'est-à-dire $shr^{(n)}(x)$

exemple

	7	6	5	4	3	2	1	0
$168 =$	1	0	1	0	1	0	0	0
$168 \gg 2 =$	0	0	1	0	1	0	1	0
$3 =$	0	0	0	0	0	0	1	1
$3 \ll 5 =$	0	1	1	0	0	0	0	0

Utilisation

On utilise les opérations binaires et les décalages pour effectuer des opérations qui nécessitent de récupérer et d'isoler certains bits.

récuperer E

On peut récurerer l'exposant biaisé d'un nombre flottant codé sur 16 bits en IEEE754.

point de code unicode

On peut fabriquer le codage utf-8 d'un point de code à l'aide d'opérations sur les bits.

Changer la casse d'une lettre codée en ASCII

En utilisant un ou exclusif bit à bit.

Acceleration de calculs

`https://en.wikipedia.org/wiki/Fast_inverse_square_root`

Généralisation

Aux entiers signés sans limitations

On considère que les nombres sont représentés par une suite infinie de bits *ultimement constante*.

definition

On dit qu'une suite est *ultimement constante* si elle est constante à partir d'un certain rang. Plus précisément, cela signifie qu'il existe un entier n_0 tel que pour tout entier n supérieur à n_0 on a l'égalité $b_n = b_{n_0}$.

- Les nombres positifs (ou nul) ont tous leurs bits nul à partir d'un certain rang.
- Les nombres négatifs ont tous leurs bits à un à partir d'un certain rang, et sont représentés avec le principe du complément à deux.

retour sur l'algorithme par divisions successives

exemple

appliquons l'algorithme à -7 :

$$-7 = 2 * (-4) + 1$$
$$-4 = 2 * (-2) + 0$$
$$-2 = 2 * (-1) + 0$$
$$-1 = 2 * (-1) + 1$$
$$-1 = 2 * (-1) + 1$$

...

On s'aperçoit que la suite est ultimement constante. si on s'autorisait un nombre infini de bits, -7 pourrait s'écrire $\overline{\ldots 1111001}_2$

lien avec le complément à deux

Ecrivons -7 en complément à deux sur 6 bits
7 s'écrit 000111_2.
On fait le complément $\overline{111000}_2$
puis on ajoute 1 : 111001_2
la représentation de -7 sur 6 bits en complément à deux est $\overline{111001}_2$

la représentation de -7 sur 16 bits en complément à deux est $\overline{1111111111111001}_2$

Un peu de Python

Les opérations bits à bits et de décalages sont disponibles en Python. Il s'agit d'une version généralisée aux entiers relatifs sans limitation.

Tableau de correspondance

nom	notation mathématique	en python
ou bit à bit	$a \vee b$	`a	b`
et bit à bit	$a \wedge b$	`a & b`
ou exclusif bit à bit	$a \oplus b$	`a ^ b`
la négation bit à bit	$\neg a$	`~ a`
décalage itéré gauche	$a \ll n$	`a << n`
décalage itéré droite	$a \gg n$	`a >> n`

Le type `bytes`

Il s'agit d'un type *iterable* pour représenter les suites d'octets.
les éléments de types `bytes` possèdent une longueur, peuvent être indicés.
Les éléments qui constituent un bytes sont des octets (entiers compris entre 0 et 255.)
Les valeurs litérales des bytes sont représentées comme une chaine de caractères précédées d'un b.
Comme les chaînes, les bytes ne sont pas mutables.
La fonction `bytes` permet de convertir une chaine en bytes à condition de préciser l'encodage en second paramètre. elle permet aussi de convertir un itérable dont les éléments sont des entiers compris entre 0 et 255 au sens large en bytes...

Le type bytes : exemples

exemples

```
>>> b=b"abcd"
>>> len(b)
4
>>> type(b)
<class 'bytes'>
>>> b[0]
97
>>> b[1]
98
>>> b[3]
100
>>> b[4]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: index out of range
```

fichier textuel/fichier binaire

Bien que tout fichier puisse être considéré comme un fichier binaire, traditionnellement et dans la plupart des langages de programmation, on fait la différence entre fichiers binaires et fichiers textuels. Le contenu d'un fichier textuel est constitué uniquement de texte dans un encodage donné (bien souvent utf-8 de nos jours).
Le cours de programmation va expliquer la lecture et l'écriture de fichiers textuels.
Le cours de codage va plutôt s'interesser aux fichiers binaires.

Les trois étapes

En python, il est possible de lire et d'écrire des fichiers.
Trois étapes sont nécessaires :

- ouverture : cette étape permet au langage de demander au système d'exploitation l'accès à une ressource
- lecture ou bien écriture dans le fichier
- fermeture du fichier : cette étape permet au langage de signaler au système qu'il peut libérer la ressource. Dans le cas de l'écriture, les écritures qui étaient en attente pour des raisons d'efficacité sont exécutées.

Ouverture

Ouverture du fichier

La fonction `open` permet d'ouvrir un fichier. Il faut préciser le nom et le mode d'ouverture. Lorsque l'ouverture s'est bien passée le résultat renvoyé est un *canal*.

modes autorisés

	binaire	textuel
lecture	`"rb"`	`"rt"`
ecriture	`"wb"`	`"wt"`
creation	`"xb"`	`"xt"`

lecture ou écriture

lecture ou écriture dans le fichier

- lecture la méthode `read` permet de lire un nombre donné d'octet dans le fichier (si on ne met rien, python lit tout le fichier !)
- la méthode `write` permet d'écrire les octets passés en paramètre

fermeture

fermeture du fichier

la methode `close` permet de refermer un fichier précédemment ouvert. Il ne faut rien passer en paramètre car cette méthode car elle s'applique sur l'objet lui même. L'appel à cette fonction possède un effet. Le résultat renvoyé est `None`

Un exemple d'écriture

exemple

```
>>> canal = open("toto.txt","wb")
>>> type(canal)
<class '_io.BufferedWriter'>
>>> canal.write(b"une lign\n")
9
>>> canal.close()
```

affichage du contenu

```
moi@maBaraque:~/codage$ cat toto.txt
une lign
moi@maBaraque:~/codage$ hexdump -C toto.txt
00000000  75 6e 65 20 6c 69 67 6e  0a                       |une lign.|
00000009
```

Un exemple de lecture

```
>>> canal = open("toto.txt","rb")
>>> lu = canal.read()
>>> canal.close()
>>> lu
b'une lign\n'
```