Bits à bits - TD

qkzk

Opérations bits à bits

Rappelons les opérations bits à bits élémentaires :

Opérations	Notation	Exemple	Remarque
AND bit à bit	`&`	`0b1100 & 0b1010 = 0b1000`	
OR bit à bit	`|`	`0b1100 | 0b1010 = 0b1110`	
XOR bit à bit	`^`	`0b1100 ^ 0b1010 = 0b0110`	
Décalage à droite	`>>`	`0b1101 >> 2 = 0b11`	`x >> n` revient à diviser x par 2^n
Décalage à gauche	`<<`	`0b1101 << 2 = 0b110100`	`x << n` revient à multiplier x par 2^n

Les objectifs de cette série d'exercices sont :

- *de vous familiariser avec les opérations bit à bit,*
- *de comprendre qu'on peut, grâce à ces opérations, extraire et manipuler n'importe quel bit d'une représentation binaire.*

Une représentation binaire n'est pas toujours qu'un entier, c'est parfois de l'information dans un message.

Lorsque des composants communiquent, ils échangent des "entiers" qu'il faut interpréter bit par bit : "si le bit 8 est à 1, cela signifie que la carte graphique est prête à recevoir des images".

6. Masquer

Le principe d'un *masque* est similaire au masque de carnaval : on cache une partie pour ne voir que ce qui nous intéresse.

1. Donner les résultats de :
   a. `0b1011 & 0b0011`
   b. `0b1011 & 0b0100`
   c. `22 & 7`
2. De manière générale, comment *récupérer les n derniers bits d'un nombre ?*

   Par exemple, les 5 derniers bits de 0*b*1100 0101 sont 00101.

   Décrire une opération réalisant ce résultat pour n quelconque.

7. Mettre un bit à 1

1. a. Comparer les représentations de `12` et `12 | 1`
   b. Comparer les représentations de `13` et `13 | 1`
   c. On considère un entier `x`. Quel est le résultat de l'opération `x | 1` ?

On dispose d'un entier `x` dont on ne connait pas la représentation.

On voudrait mettre à 1 le bit de position `n` (en comptant depuis 0 de la droite vers la gauche)

Par exemple avec `n = 4` :

```
                bit 4 à 1
xxxx xxxx xxxx ------------>  xxxx xxx1 xxxx
      654 3210                        ^
```

2. Proposer une opération bit à bit qui réalise cet exploit.

8. Alterner un bit

1. a. Comparer les représentations de `12` et `12 ^ 1`

   b. Comparer les représentations de `13` et `13 ^ 1`

   c. On considère un entier `x`. Quel est le résultat de l'opération `x ^ 1` ?

On dispose d'un entier `x` dont on ne connait pas la représentation.

On veut *inverser* le bit de position `n` en partant de la fin (et en comptant à partir de 0)

Par exemple :

```
                retourne bit 4
xxxx xxx0 xxxx ---------------->  xxxx xxx1 xxxx
xxxx xxx1 xxxx ---------------->  xxxx xxx0 xxxx
```

Les autres bits sont inchangés.

2. Proposer une opération bit à bit qui permette d'alterner le bit d'indice n (*en partant de la fin de la représentation*).

9. Convertir les 0 finaux en 1

1. Comparaison des représentations binaires de `x` et `x - 1`.

   a. Comparer les représentations binaires de 16 et 15.

   b. Recommencer avec 13 et 12.

   c. De manière générale. Compléter la phrase suivante :

"*Pour passer de la représentation binaire de n à celle de $n-1$, on change le dernier ... et tous les ... après ce ...*

On veut échanger les derniers bits d'un entier, à partir de son dernier bit à 1 :

```
             0 finaux en 1
1101 1000    ------------>  1101 1111
```

Les bits précédents le dernier 1 sont inchangés, ceux après (les 0 finaux) deviennent des 1.

Proposer une opération bit à bit.

10. extraire la fin. *Retour sur le masque*

On considère un entier sur 8 bits comme :

```
nombre  = 0b_1011_0111
indices =    0123 4567
```

On souhaite récupérer les bits numéro 2, 3 et 4 du nombre (en comptant à partir de 0).

Dans l'exemple ci-dessus cela donnerait `110`.

La démarche se fait en deux temps :

1. D'abord on décale (vers la droite ? vers la gauche ?) jusqu'à ce que ces bits occupent la position finale.
2. Ensuite on fait un *masque*, c'est à dire un Et bit à bit avec une valeur limite.

11. Dénombrer les bits à 1 d'un entier

Nous allons étudier deux algorithmes qui répondent à la même question : **compter les bits valant 1 dans un entier**

Exemple : `13 = 0b 1101` donc `nombre_bits_a_un(13) = 3`

Méthode 1

- On converti l'entier en binaire (exemple : `bin(13) = "0b1101"`)
- On itère sur la représentation et compte les `"1"`.

1. Écrire une fonction Python qui implémente cet algorithme
2. La faire tourner sur 5, 9 et 14.

Méthode 2, de Brian Kernighan – Accrochez vous.

Remarques initiales :

- Retirer 1 à un entier inverse tous les bits après le dernier bit à 1. Par exemple :

```
décimal         binaire
10                 1010
9 = 10 - 1         1001
8 =  9 - 1         1000
7 =  8 - 1         0111
6 =  7 - 1         0110
5 =  6 - 1         0101
```

- Donc, si on soutrait 1 et qu'on fait un ET bit à bit avec lui même : `n & (n - 1)`, on passe à 0 tous les derniers bits :

```
décimal         binaire
10                 1010
9 = 10 - 1         1001
10 & 9             1000
```

- Si on répète `n = (n & (n - 1))` jusqu'à avoir `n == 0` et qu'on compte les tours, on a le nombre de bits à 1 dans un entier.

```
 décimal                binaire

n   = 10                  1010 : 2 bits à 1
           Tour 1
n-1 =  9                  1001
n = (n & (n-1))
10 & 9 = 8                1000

n   =  8                  1000
          Tour 2
n-1 =  7                  0111
n = (n & (n-1))
8 & 7 = 0                 0000
```

L'algorithme s'arrête parce que `n` vaut 0. Donc 10 comporte **deux bits à 1**.

1. Écrire une fonction Python qui implante cet algorithme
2. La faire tourner sur 5, 9 et 14.

En Python

def masquer(n: int, masque: int) -> int: “” “n & masque” “” return n & masque

print(“Masquer”, bin(117), bin(31), bin(masquer(117, 31)))

def passer_bit_a_1(n: int, indice: int) -> int: “” “Renvoie l'entier avec son bit d'indice n à 1. (à partir de la fin)” “” return n | (1 « indice)

print(“Passer bit 3 à 1”, bin(117), bin(passer_bit_a_1(117, 3)))

def alterner_bit(n: int, indice: int) -> int: “” “Renvoie l'entier en alternant son bit d'indice n. (à partir de la fin)” “” return n ^ (1 « indice)

print(“Alterner bit 3”, bin(117), bin(alterner_bit(117, 3))) print(“Alterner bit 2”, bin(117), bin(alterner_bit(117, 2)))

def convertir_0_finaux(n: int) -> int: “” “Converti les 0 de la fin du nombre en 1” “” return n | (n - 1)

print(“Convertir 0 finaux”, bin(0b10101000), bin(convertir_0_finaux(0b10101000)))

def derniers_bits(n: int, nb_bits: int) -> int: “‘ “Renvoie l’entier représenté par les derniers bits de n” “” return masquer(n, (1 « nb_bits) - 1)

print(“5 derniers bits”, bin(117), bin(derniers_bits(117, 5)))

def compter_bits_a_1_version1(n: int) -> int: “‘ “renvoie le nombre de bits à 1 de l’entier” “” nb = 0 binaire = bin(n) for i in range(2, len(binaire)): if binaire[i] == “1”: nb = nb + 1 return nb

def compter_bits_a_1_version2(n: int) -> int: “‘ “renvoie le nombre de bits à 1 de l’entier” “” nb = 0 while n != 0: n = n & (n - 1) nb = nb + 1 return nb

print(“compter_bits_a_1_version1”, bin(0b11101010), compter_bits_a_1_version1(0b11101010)) print(“compter_bits_a_1_versio bin(0b11101010), compter_bits_a_1_version2(0b11101010))