Chap. III Travailler en base 2, 10 ou 16

BRUNO DARID

13 août 2019

PLAN

1 Repères historiques

L'invention du système binaire est souvent attribué à tort au savant allemand Wilhelm Gottfried Leibniz (1646-1716). En effet, le mathématicien et astronome anglais Thomas Harriot (1560-1621) avait déjà travaillé sur les systèmes non décimaux.
Leibniz s'inspire des figures de l'empereur chinois Fuxi (-3000) que l'on peut considérer comme les premières expressions d'un codage binaire.

Fig. 1 – Trigramme de Fuxi

2 Comment représenter des informations sur une machine numérique ?

2.1 Unité d'information

Les machines numériques sont composées d'éléments manipulant l'information sous forme de **deux états distincts**. C'est la raison pour laquelle on les appelle machine **binaire**. Par convention, ces états sont notés 0 ou 1. En anglais, ces informations élémentaires sont appelées *binary digit* ou **bit**.
Un bit permet de représenter $2^1 = 2$ informations.

— Citer un système qui nécessite **un** bit pour coder son information.

— Combien de bits sont nécessaires pour distinguer les 4 familles d'un jeu de cartes ?

En généralisant on a la propriété importante : n bits permettent de représenter 2^n informations. Autres exemples :

— Combien de bits sont nécessaires pour distinguer les 26 lettres de l'alphabet ?

— Justifier le fait qu'à l'origine 7 bits étaient suffisants pour coder du texte (*en anglais !*)

```
In [1]: ##################Réponses aux questions##################
        # 1) Un détecteur de fumée: présence de fumée activation alarme -> état 1
        # pas de fumée, pas d'alarme -> état 0
        # 2) Deux bits sont suffisants (voir schéma)

In [ ]: # 3) 5 bits nécessaire, car 2^4 < 26 < 2^5
        # 4) 26*2 (minuscules + majuscules) + 10 (chiffres) + 30(ponctuation) = 92
        # Or 2^6 < 92 < 2^7
```

Les machines numériques manipulent habituellement des groupes de bits. Ainsi, un groupe de 8 bits forment un **octet** (!!!! ATTENTION !!!!! en anglais un **octet** est traduit par **BYTE**). Au

delà de 8 bits on utilise le terme **mot**. On parle par exemple de mot de 16 bits, de mot de 32 bits (on trouve aussi le terme *double mot*) ou de mot de 64 bits (ou *quadruple mot*).

Les quantités d'informations stockées ou manipulées s'expriment avec les préfixes habituels reliés aux puissances de 10 (1 kilooctet (ko) $= 1 \times 10^3$ octets, 1 mégaoctet (Mo) $= 1 \times 10^6$ octets, 1 gigaoctet (Go) $= 1 \times 10^9$ octets etc).
Remarque : certains informaticiens/électroniciens utilisent encore des définitions de ces quantités exprimées en puissances de 2 (2^{10}, 2^{20}, 2^{30}, etc). Pour éviter les confusions, l'International Electrotechnical Commission (IEC) a normalisé ces appellations en décembre 1998. Voir lien.

2.2 Représentation binaire des entiers naturels

2.2.1 Représentation positionnelle

On présente les nombres entiers de manière classique en *représentation positionnelle*. Par exemple en base 10, la suite 123 signifie $1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$. En base 2, la suite 10011 signifie $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$.
Dans un motif binaire, le bit le plus à droite est le **bit de poids faible** ou **LSB** (*Least Significant Bit*). A l'opposé, le bit le plus à gauche est le **bit de poids fort** ou **MSB** (*Most Significant Bit*). Le bit de poids faible peut être utilisé pour trouver la parité du nombre : 0 indique un nombre pair et 1 un nombre impair.

2.2.2 Conversion binaire - décimale

La conversion binaire - décimale d'un entier naturel découle simplement du mode de représentation positionnelle : on additionne les puissances de 2 présentes dans le motif binaire. Ainsi, pour l'exemple précédent on a :
$10011_2 = (1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10} = 19_{10}$
Remarque
La base a été indiquée ici en indice, on peut l'omettre lorsqu'il n'y a pas d'ambiguité.

2.2.3 Conversion décimale - binaire

On décompose le nombre entier en une somme de puissance de 2 par divisions entières successives tant que le quotient est supérieur ou égal à 2.
Exemple : soit $N = 23$ à convertir en binaire

```
In [23]: %%latex
         \begin{align*}
         23 &=  2\times 11+1 \\
          &=  2\times(2\times5 +1)+1\\
          &= 2\times(2\times(2\times2 +1)+1)+1\\
          &= 2\times(2\times(2\times(2\times1+0)+1)+1)+1
         \end{align*}
```

$$\begin{aligned} 23 &= 2 \times 11 + 1 \\ &= 2 \times (2 \times 5 + 1) + 1 \\ &= 2 \times (2 \times (2 \times 2 + 1) + 1) + 1 \\ &= 2 \times (2 \times (2 \times (2 \times 1 + 0) + 1) + 1) + 1 \end{aligned}$$

Le résultat est donc $N = 10111_2$. On peut le retrouver en juxtaposant les restes des divisions « potences »(voir exemple au tableau).
Applications

1. Trouver la représentation binaire de 77_{10} et 123_{10}.
2. Trouver la représentation décimale de $1101\ 1011_2$.

3 Représentation hexadécimale

3.1 Chiffres hexadécimaux

La représentation binaire devient rapidement encombrante. On lui préfère souvent la représentation *hexadécimale* (base 16). L'utilisation de cette base nécessite 16 caractères. Le tableau ci-dessous présente ces caractères ainsi que leur équivalence en base 10 et 2.

Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadécimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Fig. 2 – Valeurs binaire et décimale des chiffres hexadécimaux

3.2 Conversions décimale - hexadécimale

Les principes de conversion décimal-hexdécimal et vice-versa sont identiques aux principes de conversion vus pour le binaire.
Applications

1. Trouver la représentation hexdécimale de 77_{10} et 123_{10}.
2. Trouver la représentation décimale de $6A0_{16}$.

3.3 Conversions binaire - hexadécimal

Pour passer du binaire à l'hexadécimal il suffit de grouper les bits par 4 et de les remplacer par leur équivalent hexdécimal (*voir tableau ci-dessus*).
Exemple
$1101\ 1011_2 = DB_{16}$
Pour convertir de l'hexadécimal en binaire, on remplace chaque chiffre hexadécimal par son équivalent binaire sur 4 bits.
Exemple $7A5_{16} = 0111\ 1010\ 0101_2$

4 À retenir

L'information numérique est manipulée sous forme de *bits*. Les machines numériques travaillent avec des groupes de bits : des octets (8 bits) ou des mots (16, 32 ou 64 bits).
Les nombres entiers sont représentés en machine par des motifs binaires, c'est-à-dire une décomposition suivant les puissances de 2, obtenue par divisions successives par 2 par exemple.

Afin de limiter l'encombrement des motifs binaires on peut utiliser la représentation hexadécimale où chaque groupe de 4 bits du nombre binaire est remplacé par un chiffre hexadécimal compris entre 0 et F. Le passage du décimal à l'hexadécimal se fait par divisions successives par 16.

5 E1C3 Manipulations binaires en python

5.1 Préambule : types natifs

Python possède trois types numériques natifs, parmi lesquels on peux citer les entiers **relatifs** (*integers*) et les nombres décimaux à virgule flottante (*float*). Pour manipuler du texte, on dispose du type chaine de caractères (*string*). Ces dernières sont déclarées en étant entourées de double quotes " " ou de simples quotes ' '. Par exemple

```
mavariable = "programmation"
```

Pour accéder au type d'un objet `obj` on utilise la fonction native `type()` sur cet objet :

```
type(obj)
```

1. Donner le type des objets suivants : 11, 11.0 et "11".
2. Représentent-ils le même objet ?

5.2 Convertisseur decimal binaire version 1

Python possède une fonction native qui permet la conversion d'un entier en binaire : la fonction `bin()`.

1. Consulter la documentation associée à cette fonction.
2. Convertir l'entier 77 en binaire et l'affecter à une variable nommée `valeurbinaire`.
3. Quel est le type de `valeurbinaire` ?
4. On souhaiterait afficher `valeurbinaire` sans le préfixe `0b`. La fonction (on dit ici *méthode* associée aux chaines de caractères) qui pourrait être utilisée est `strip()`. Exemple d'utilisation :

   ```
   com = "# Ceci n'est pas un commentaire valide en python"
   print(com.strip("# "))
   Ceci n'est pas un commentaire valide en python
   ```

   — Consulter la documentation associée à `str.strip`.
   — Afficher `valeurbinaire` sans le préfixe `0b`.

5.3 Convertisseur decimal binaire version 2

Dans cette partie on va implémenter en python l'algorithme (*voir Fig. 3*) évoqué au paragraphe 2.2.3. L'algorithme est écrit en pseudo-code, dans lequel l'affectation est notée ← et la chaine de caractères vide "".

1. Ecrire une fonction en python `dec2bin(n)` qui retourne la conversion en binaire d'un entier n.
   Indications :

Algorithme 1 : Conversion binaire

Données : n entier naturel
Résultat : chaine correspondant à la conversion binaire de n
resultat : *chaine* $\leftarrow$ ""
quotient : *entier* $\leftarrow n$
si $n = 0$ **alors**
  resultat $\leftarrow$ "0"
sinon
  tant que *quotient* $\neq 0$ **faire**
    resultat $\leftarrow$ "reste division quotient par 2" + *resultat*
    quotient $\leftarrow$ *quotient*//2
retourner *resultat*

Fig. 3 – Algorithme de conversion décimal-binaire

— en python le reste de la division de a par b est obtenu par `a % b`;
— la division entière de a par b s'obtient par `a // b`;
— pour convertir un nombre x en caractère "x" on utilise `str(x)`.

2. Justifier simplement que la boucle **Tant que** termine.
3. *Pour les plus rapides* : retourner le résultat sur 16 bits en complétant le cas échéant avec des zéros. Indication : se documenter sur la fonction `str.zfill()`.

6 E2C3 Manipulations hexdécimales en python

6.1 Préambule

Python possède une fonction native permettant de convertir un entier en hexadécimal : `hex()`.

1. Consulter la documentation de `hex`.
2. Convertir $N_1 = 352_{10}$ en hexadécimal.
3. Afficher la valeur sans le préfixe '0x'. *Conseil* : voir l'exercice du paragraphe 5.2.

6.2 Traitement d'une chaine hexadécimale

Une fonction dont le code est donnée ci-dessous est proposée. Malheureusement, une partie de sa spécification n'a pas été complétée.

1. Que réalise cette fonction ? Faire des tests avec les nombres "10 *FA* 10 00" et "84 *BD* 10 01".
2. Afficher la conversion en entier de ces nombres préalablement transformés par la fonction `big2little()`.

```
In [3]: def big2little(nh):
            """
            Retourne .....
            On suppose que nh est une chaine de longueur multiple de 2 pouvant
            être interprété comme un nombre hexadécimal.
```

```
        """
        resultat = ""
        l = len(nh)
        for i in range(0,l-1,2):
            resultat = nh[i:i+2] + resultat
        return resultat
```

```
In [4]: ########REPONSES##########
```

6.3 Application : entête d'un fichier image

Les fichiers images au format BMP sont bien documentés. On peut aisemment consulter les informations avec un éditeur hexadecimal. Les 2 premiers octets servent à identifier le fichier et les 4 suivants nous donnent la taille du fichier (*en hexadécimal !*).
En python l'ouverture d'un fichier se fait avec la fonction native `open()` et obeit à la construction suivante :

```
f = open('nom_fichier','rb')#lecture en mode binaire
#traitement(s)
#
f.close()#important !!
```

La séquence `f = open()` retourne un objet "fichier" qui dispose de nombreuses *méthodes*, parmi lesquelles on peut citer `read(n)` qui permet de lire n octets.

1. Ouvrir le fichier "alien.bmp" en lecture binaire.
2. Lire et afficher le type de fichier.
3. Lire les 4 octets suivants et les stocker dans une variable `taille_big`.
4. Stocker dans une variable `taille` la taille du fichier BMP. Il suffit de passer `taille_big` à la fonction `big2little()`.
5. Convertir cette taille en entier (fonction `int()`). Cette valeur est-elle cohérente avec celle fournie par le système d'exploitation ?

```
In [ ]: ###############CORRECTION##################
        f = open('alien.bmp', 'rb')# ouverture du fichier en lecture binaire
        type_fichier = f.read(2)# on lit 2 octets
        print(type_fichier)
        taille_big = f.read(4)# on lit les 4 octets suivants
        f.close()# on n'a plus besoin du fichier, on le referme !
        taille = big2little(taille_big)
        print(int(taille, 16))# conversion en entier à partir de la base 16
```

7 Résumé des fonctions python rencontrées

Le type d'un objet python `obj` s'obtient par `type(obj)`. Les changements de base se font avec les fonctions `int(), bin(), hex()`. La conversion en chaine de caractères est réalisée par la fonction `str()`. Les chaines de caractères possèdent de nombreuses méthodes. Dans ce chapitre, la

méthode `strip()` a été utilisée ; elle permet d'éliminer des caractères en début et fin de chaine.
La division euclidienne d'un entier a par un entier b a également été abordée : $a = b \times q + r$. En python, le quotient s'obtient par `q = a // b` et le reste par `r = a % b`.