Représentation de l'information

Bruno Grenet

Université de Montpellier

DIU Enseignement de l'informatique – Juin 2019

Représentations

- Mémoire d'un ordinateur
  - suite de bits
  - organisée en octets (8 bits)
  - organisés en mots (4 ou 8 mots)
  - ...
- Une donnée : une suite de bits ou d'octets ou de mots

Représentations

- Mémoire d'un ordinateur
  - suite de bits
  - organisée en octets (8 bits)
  - organisés en mots (4 ou 8 mots)
  - ...
- Une donnée : une suite de bits ou d'octets ou de mots

Comment donner un sens (sémantique) à ces bits ?

Formats :

- image : JPG, BMP, PNG, ...
- sons : WAV, MP3, ...
- texte : ASCII, Latin1, UTF-8, ...
- nombres : `int`, `float`, `unsigned long long int`

Représentations

- Mémoire d'un ordinateur
  - suite de bits
  - organisée en octets (8 bits)
  - organisés en mots (4 ou 8 mots)
  - ...
- Une donnée : une suite de bits ou d'octets ou de mots

Comment donner un sens (sémantique) à ces bits ?

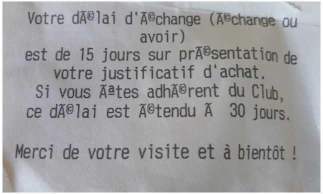
Formats :

- image : JPG, BMP, PNG, ...
- sons : WAV, MP3, ...
- texte : ASCII, Latin1, UTF-8, ...
- nombres : `int`, `float`, `unsigned long long int`

Tout est une question de convention !

Représentations

- Mémoire d'un ordinateur
  - suite de bits
  - organisée en octets (8 bits)
  - organisés en mots (4 ou 8 mots)
  - ...
- Une donnée : une suite de bits ou d'octets ou de mots



Comment donner un sens (sémantique) à ces bits ?

Formats :

- image : JPG, BMP, PNG, ...
- sons : WAV, MP3, ...
- texte : ASCII, Latin1, UTF-8, ...
- nombres : `int`, `float`, `unsigned long long int`

Tout est une question de convention !

Représentation des nombres

Représentation mathématique des entiers positifs

$$1492 = 1 \times 1000 + 4 \times 100 + 9 \times 10 + 2 \times 1$$

Représentation mathématique des entiers positifs

$$1492 = 1 \times 1000 + 4 \times 100 + 9 \times 10 + 2 \times 1$$

$$19 = \overline{10011}^{2} = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Représentation mathématique des entiers positifs

$$1492 = 1 \times 1000 + 4 \times 100 + 9 \times 10 + 2 \times 1$$

$$19 = \overline{10011}^{2} = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$764\,953 = \overline{bac19}^{16} = 11 \times 16^4 + 10 \times 16^3 + 12 \times 16^2 + 1 \times 16^1 + 9 \times 16^0$$

Représentation mathématique des entiers positifs

$$1492 = 1 \times 1000 + 4 \times 100 + 9 \times 10 + 2 \times 1$$

$$19 = \overline{10011}^{2} = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$764\,953 = \overline{bac19}^{16} = 11 \times 16^4 + 10 \times 16^3 + 12 \times 16^2 + 1 \times 16^1 + 9 \times 16^0$$

C'est la vision mathématique !

- Sous-tend la vision informatique
- Pas suffisant...

Représentation informatique des entiers positifs

Comment délimiter les nombres ?

Représentation informatique des entiers positifs

Comment délimiter les nombres ?

- Taille fixée (1492 sur 8 chiffres $\rightarrow$ 00001492)
  - une taille w = un « type » entier (`unsigned int`, `unsigned long`, ...)
  - entiers entre 0 et $2^w - 1$

Représentation informatique des entiers positifs

Comment délimiter les nombres ?

- Taille fixée (1492 sur 8 chiffres $\rightarrow$ 00001492)
  - une taille w = un « type » entier (`unsigned int`, `unsigned long`, ...)
  - entiers entre 0 et $2^w - 1$
- Taille quelconque : entiers *multiprécision*
  - tableaux de nombres
  - écriture en base 2^w

Représentation informatique des entiers positifs

Comment délimiter les nombres ?

- Taille fixée (1492 sur 8 chiffres $\to$ 00001492)
  - une taille w = un « type » entier (`unsigned int`, `unsigned long`, ...)
  - entiers entre 0 et $2^w - 1$
- Taille quelconque : entiers *multiprécision*
  - tableaux de nombres
  - écriture en base 2^w

Quel est le premier chiffre ?

- Unité en dernier : grand boutiste (*big endian*)
- Unité en premier : petit boutiste (*little endian*)

Représentation informatique des entiers positifs

Comment délimiter les nombres ?

- Taille fixée (1492 sur 8 chiffres $\rightarrow$ 00001492)
  - une taille w = un « type » entier (`unsigned int`, `unsigned long`, ...)
  - entiers entre 0 et $2^w - 1$
- Taille quelconque : entiers *multiprécision*
  - tableaux de nombres
  - écriture en base 2^w

Quel est le premier chiffre ?

- Unité en dernier : grand boutiste (*big endian*)
- Unité en premier : petit boutiste (*little endian*)

Python : entiers multiprécision

- Transparent pour l'utilisateur

Opérations arithmétiques

Taille fixée

- implantation matérielle de l'arithmétique binaire
  - Circuits électroniques pour addition, multiplication, ...
- calculs *modulo* 2^w :

```
unsigned int x = 3, y = 5, z = 4294967295;
printf("%u ; %u", x-y, z+1);
4294967294 ; 0
```

Opérations arithmétiques

Taille fixée

- implantation matérielle de l'arithmétique binaire
  - Circuits électroniques pour addition, multiplication, ...
- calculs *modulo* 2^w :

```
unsigned int x = 3, y = 5, z = 4294967295;
printf("%u ; %u", x-y, z+1);
4294967294 ; 0
```

Taille quelconque

- Logiciels (`cpython`, `GMP`, `BigInt`, ...)
- Addition, multiplication, ... en base 2^w à partir des opérations sur w bits
- Questions algorithmiques !
  - Le meilleur algo. de multiplication connu date de mi-mars

Opérations arithmétiques

Taille fixée

- implantation matérielle de l'arithmétique binaire
  - Circuits électroniques pour addition, multiplication, ...
- calculs *modulo* 2^w :

```
unsigned int x = 3, y = 5, z = 4294967295;
printf("%u ; %u", x-y, z+1);
4294967294 ; 0
```

Taille quelconque

- Logiciels (`cpython`, `GMP`, `BigInt`, ...)
- Addition, multiplication, ... en base 2^w à partir des opérations sur w bits
- Questions algorithmiques !
  - Le meilleur algo. de multiplication connu date de mi-mars

Analogie

- Circuits : tables de multiplication
- Logiciel : poser une multiplication

Représentation des nombres relatifs

Bonne idée : ajouter un bit de signe

- $\texttt{11101101} \rightarrow \texttt{1 1101101} \rightarrow -\overline{1101101}^2 = -109$
- Apparemment facile...
- ... mais que devient l'algorithme d'addition ?

Représentation des nombres relatifs

Bonne idée : ajouter un bit de signe

- $11101101 \to 1\ 1101101 \to -\overline{1101101}^2 = -109$
- Apparemment facile...
- ... mais que devient l'algorithme d'addition ?

Meilleure idée : complément à la base

- $11101101 \to \overline{11101101}^2 = 237 \to 237 - 2^8 = -19$
- Complément à la base sur w bits :
  - interprétation des bits comme entier positif n
  - si $n \geq 2^{w-1}$: remplacer par $n - 2^w$
  - Entiers représentables : $[-2^{w-1}, 2^{w-1} - 1]$

Représentation des nombres relatifs

Bonne idée : ajouter un bit de signe

- 11101101 $\rightarrow$ 1 1101101 $\rightarrow -\overline{1101101}^2 = -109$
- Apparemment facile. . .
- . . . mais que devient l'algorithme d'addition ?

Meilleure idée : complément à la base

- 11101101 $\rightarrow \overline{11101101}^2 = 237 \rightarrow 237 - 2^8 = -19$
- Complément à la base sur w bits :
  - interprétation des bits comme entier positif n
  - si $n \geq 2^{w-1}$: remplacer par $n - 2^w$
  - Entiers représentables : $[-2^{w-1}, 2^{w-1} - 1]$
- Opérations toujours valides !
  - *w bits représentent une classe d'équivalence de $\mathbb{Z}/2^w\mathbb{Z}$*
  - entiers positifs ou relatifs : choix de représentants de $\mathbb{Z}/2^w\mathbb{Z}$

Récapitulatif sur les entiers

Entiers de taille w fixée : calculs *modulo* 2^w, en matériel

- Positifs
  - représentation binaire classique
  - entiers compris entre 0 et $2^w - 1$
  - types C : `unsigned int`, `unsigned long`, ...
- Relatifs
  - complément à la base
  - entiers compris entre -2^{w-1} et $2^{w-1} - 1$
  - types C : `int`, `long`, ...

Récapitulatif sur les entiers

Entiers de taille w fixée : calculs *modulo* 2^w, en matériel

- Positifs
  - représentation binaire classique
  - entiers compris entre 0 et $2^w - 1$
  - types C : `unsigned int`, `unsigned long`, ...
- Relatifs
  - complément à la base
  - entiers compris entre -2^{w-1} et $2^{w-1} - 1$
  - types C : `int`, `long`, ...

Entiers de taille quelconque :

- Bibliothèques logicielles (`GMP`, `cpython`, `BigInt`, ...)
- Écriture en base 2^w ($w = 32$ ou 64)

Récapitulatif sur les entiers

Entiers de taille w fixée : calculs *modulo* 2^w, en matériel

- Positifs
  - représentation binaire classique
  - entiers compris entre 0 et $2^w - 1$
  - types C : `unsigned int`, `unsigned long`, ...
- Relatifs
  - complément à la base
  - entiers compris entre -2^{w-1} et $2^{w-1} - 1$
  - types C : `int`, `long`, ...

Entiers de taille quelconque :

- Bibliothèques logicielles (`GMP`, `cpython`, `BigInt`, ...)
- Écriture en base 2^w ($w = 32$ ou 64)

Python : entiers multiprécision

Représentation mathématique des nombres réels

$$3,14159\ldots = 3 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4} + 9 \times 10^{-5} + \cdots$$

Représentation mathématique des nombres réels

$$3,14159\ldots = 3 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4} + 9 \times 10^{-5} + \cdots$$

$$299\,792,458 = 2,99792458 \times 10^{6}$$

Représentation mathématique des nombres réels

$$3,14159\ldots = 3 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4} + 9 \times 10^{-5} + \cdots$$

$$299\,792,458 = 2,99792458 \times 10^{6}$$

$$\begin{aligned} 9,80665 &= \overline{1001,11001...}^{2} = 2^{3} + 2^{0} + 2^{-1} + 2^{-2} + 2^{-5} + \cdots \\ &= \overline{1,00111001...}^{2} \times 2^{3} \end{aligned}$$

Représentation mathématique des nombres réels

$$3,14159\ldots = 3 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4} + 9 \times 10^{-5} + \cdots$$

$$299\,792,458 = 2,99792458 \times 10^{6}$$

$$\begin{aligned} 9,80665 &= \overline{1001,11001...}^{2} = 2^{3} + 2^{0} + 2^{-1} + 2^{-2} + 2^{-5} + \cdots \\ &= \overline{1,00111001...}^{2} \times 2^{3} \end{aligned}$$

Remarque fondamentale

Sur un ordinateur, représentation finie $\Longrightarrow$ approximation des nombres réels

Représentation mathématique des nombres réels

$$3,14159\ldots = 3 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4} + 9 \times 10^{-5} + \cdots$$

$$299\,792,458 = 2,99792458 \times 10^{6}$$

$$\begin{aligned} 9,80665 &= \overline{1001,11001...}^{2} = 2^{3} + 2^{0} + 2^{-1} + 2^{-2} + 2^{-5} + \cdots \\ &= \overline{1,00111001...}^{2} \times 2^{3} \end{aligned}$$

Remarque fondamentale

Sur un ordinateur, représentation finie $\Longrightarrow$ approximation des nombres réels

Les flottants

- Représentation binaire, en notation scientifique, à précision fixée
- Norme IEEE-754 : fixe la représentation et les règles d'arrondi

Exemple sur 32 bits (`float`)

`seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm`

- s : 1 bit de signe
- e : 8 bits d'exposant
- m : 23 bits de *mantisse*

Exemple sur 32 bits (`float`)

seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm

- s : 1 bit de signe
- e : 8 bits d'exposant
- m : 23 bits de *mantisse*

$$(-1)^s \times \overline{\mathbf{1},\mathtt{mmm}\cdots\mathtt{m}}^2 \times 2^{e-127}$$

Exemple sur 32 bits (`float`)

`seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm`

- s : 1 bit de signe
- e : 8 bits d'exposant
- m : 23 bits de *mantisse*

$$(-1)^s \times \overline{\mathbf{1},\mathtt{mmm}\cdots\mathtt{m}}^2 \times 2^{e-127}$$

- Mantisse : le premier 1 n'est pas stocké !
- Exposant *décalé* de $2^{8-1} - 1 = 127$: exposants entre -127 à 128
- Cas particuliers d'exposants :
  - -127 : utilisé pour représenter 0 et les nombres *dénormalisés*
  - 128 : $\pm\infty$ et « `NaN` »

Cas particuliers

Exposant $e = 0000\ 0000$

- Si $m = 0 \cdots 0$: ± 0 en fonction du signe
- Sinon, nombre *dénormalisé* : $(-1)^s \times \mathbf{0}, \mathtt{mmm} \cdots \mathtt{m} \times 2^{\mathbf{-126}}$
  - Très petits nombres

Cas particuliers

Exposant $e = 0000\ 0000$

- Si $m = 0 \cdots 0$: ± 0 en fonction du signe
- Sinon, nombre *dénormalisé* : $(-1)^s \times \mathbf{0}, \mathtt{mmm} \cdots \mathtt{m} \times 2^{\mathbf{-126}}$
  - Très petits nombres

Exposant $e = 1111\ 1111$

- Si $m = 0 \cdots 0$: $\pm\infty$ en fonction du signe
  - Gestion des dépassements de capacité
- Sinon : `NaN`
  - Tout type d'erreurs ($1/0$, $\sqrt{-1}$, ...)

Cas particuliers

Exposant $e = 0000\ 0000$

- Si $m = 0 \cdots 0$: ± 0 en fonction du signe
- Sinon, nombre *dénormalisé* : $(-1)^s \times \mathbf{0}, \mathtt{mmm} \cdots \mathtt{m} \times 2^{\mathbf{-126}}$
    - Très petits nombres

Exposant $e = 1111\ 1111$

- Si $m = 0 \cdots 0$: $\pm\infty$ en fonction du signe
    - Gestion des dépassements de capacité
- Sinon : `NaN`
    - Tout type d'erreurs ($1/0$, $\sqrt{-1}$, ...)

Tout est histoire de convention

- Ne pas chercher à comprendre *pourquoi* ces choix ont été faits...
- Ce qui compte n'est pas de retenir les détails !

Cas particuliers

Exposant $e = 0000\ \ 0000$

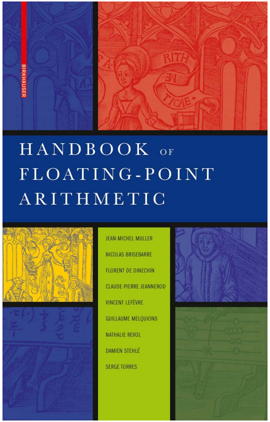
- Si $m = 0 \cdots 0$: ± 0 en fonction du signe
- Sinon, nombre *dénormalisé* : $(-1)^s \times \mathbf{0}, \mathtt{mmm} \cdots \mathtt{m} \times 2^{\mathbf{-126}}$
  - Très petits nombres

Exposant $e = 1111\ \ 1111$

- Si $m = 0 \cdots 0$: $\pm\infty$ en fonction du signe
  - Gestion des dépassements de capacité
- Sinon : `NaN`
  - Tout type d'erreurs ($1/0$, $\sqrt{-1}$, ...)

Tout est histoire de convention

- Ne pas chercher à comprendre *pourquoi* ces choix ont été faits...
- Ce qui compte n'est pas de retenir les détails !



Tailles et opérations arithmétiques

Tailles des flottants

- `float` : 32 bits
- `double` : 64 bits (11 d'exposant, 52 de mantisse)
- autre : 16 bits, 128 bits, etc.

Tailles et opérations arithmétiques

Tailles des flottants

- `float` : 32 bits
- `double` : 64 bits (11 d'exposant, 52 de mantisse)
- autre : 16 bits, 128 bits, etc.

Opérations arithmétiques

- Circuits électroniques pas simples !
- Opérations non exactes : renvoie le flottant *le plus proche* du résultat mathématique

Tailles et opérations arithmétiques

Tailles des flottants

- `float` : 32 bits
- `double` : 64 bits (11 d'exposant, 52 de mantisse)
- autre : 16 bits, 128 bits, etc.

Opérations arithmétiques

- Circuits électroniques pas simples !
- Opérations non exactes : renvoie le flottant *le plus proche* du résultat mathématique

Flottants multiprécision

- Logiciels (`MPFR`, `BigDecimal`, `Decimal`, ...)

En pratique en Python

- Type `float` Python = type `double` C
  - 11 bits d'exposant
  - 52 bits de mantisse
- Type `Decimal` : sorte de flottant multiprécision

En pratique en Python

- Type `float` Python = type `double` C
  - 11 bits d'exposant
  - 52 bits de mantisse
- Type `Decimal` : sorte de flottant multiprécision

Affichage ≠ Représentation interne

```
>>> n = 12 # entier
>>> x = 12.0 # flottant
>>> 2.4e-3
0.0024
>>> y = 0.1; y # Affichage simplifié
0.1
>>> 'f{y:.30f}' # Affichage de 30 décimales
0.100000000000000005551115123126
```

Récapitulatif sur les flottants

- Norme IEEE-754
  - Diverses tailles (16, 32, 64, … bits)
  - Représentation et arrondis normalisés
  - … mais complexes à comprendre en détail
- Opérations implantées en matériel
- En Python, flottants sur 64 bits :
  - 1 bit de signe
  - 11 bits d'exposants (2^{e-1023})
  - 52 bits de mantisse
- Affichage *agréable* mais trompeur !

Récapitulatif sur les flottants

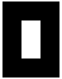
- Norme IEEE-754
  - Diverses tailles (16, 32, 64, … bits)
  - Représentation et arrondis normalisés
  - … mais complexes à comprendre en détail
- Opérations implantées en matériel
- En Python, flottants sur 64 bits :
  - 1 bit de signe
  - 11 bits d'exposants (2^{e-1023})
  - 52 bits de mantisse
- Affichage *agréable* mais trompeur !

Les flottants sont entièrement déterministes et prévisibles… si on en est capable !

Représentation des images

Les formats *Portable Pixmap*

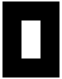
```
P1      # Format : PBM
3 4     # Dimensions : largeur hauteur
1 1 1   # Première ligne
1 0 1   # Deuxième ligne
1 0 1   # Troisième ligne
1 1 1   # Quatrième ligne
```



```
P3     # Format : PPM
3 3    # Dimensions
7      # Niveaux de couleurs
0 7 0   7 7 0   7 0 0
4 0 4   0 0 3   2 2 2
0 7 0   7 7 0   7 0 0
```

Les formats *Portable Pixmap*

```
P1      # Format : PBM
3 4     # Dimensions : largeur hauteur
1 1 1   # Première ligne
1 0 1   # Deuxième ligne
1 0 1   # Troisième ligne
1 1 1   # Quatrième ligne
```



```
P3     # Format : PPM
3 3    # Dimensions
7      # Niveaux de couleurs
0 7 0   7 7 0   7 0 0
4 0 4   0 0 3   2 2 2
0 7 0   7 7 0   7 0 0
```

Description

1. `P1` (format PBM), `P2` (format PGM) ou `P3` (format PPM)
2. Dimensions (hauteur $\times$ largeur)
3. Si PGM ou PPM : valeur de *couleur* maximale
4. Lignes suivantes : description pixel par pixel, ligne par ligne
   - Si PBM ou PGM : un entier par pixel
   - Si PPM : trois entiers par pixel

En Python

Représentation d'une image

- Image : matrice (liste de listes) de pixels
- Pixel : un entier pour PBM et PGM ; un triplet `(R,G,B)` pour PPM

```
im1 = [[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]]
im2 = [[(0,7,0), (7,7,0), (7,0,0)], [(4,0,4), (0,0,3), (2,2,2)],
       [(0,7,0), (7,7,0), (7,0,0)]]
```

En Python

Représentation d'une image

- Image : matrice (liste de listes) de pixels
- Pixel : un entier pour PBM et PGM ; un triplet `(R,G,B)` pour PPM

```
im1 = [[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]]
im2 = [[(0,7,0), (7,7,0), (7,0,0)], [(4,0,4), (0,0,3), (2,2,2)],
       [(0,7,0), (7,7,0), (7,0,0)]]
```

Fichier `es.py` : entrées-sorties

- `sauver(image, fmt, nom, maxi = 0)` : sauve l'`image`
  - avec le format `fmt` (`'PBM'`, `'PGM'`, `'PPM'`)
  - dans le fichier `nom.ext` où `ext in [pbm, pgm, ppm]`
  - avec comme couleur maximale `maxi` (si 0, le max trouvé pour PGM/PPM)

En Python

Représentation d'une image

- Image : matrice (liste de listes) de pixels
- Pixel : un entier pour PBM et PGM ; un triplet `(R,G,B)` pour PPM

```
im1 = [[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]]
im2 = [[(0,7,0), (7,7,0), (7,0,0)], [(4,0,4), (0,0,3), (2,2,2)],
        [(0,7,0), (7,7,0), (7,0,0)]]
```

Fichier `es.py` : entrées-sorties

- `sauver(image, fmt, nom, maxi = 0)` : sauve l'`image`
  - avec le format `fmt` (`'PBM'`, `'PGM'`, `'PPM'`)
  - dans le fichier `nom.ext` où `ext in [pbm, pgm, ppm]`
  - avec comme couleur maximale `maxi` (si 0, le max trouvé pour PGM/PPM)
- `charger_image(nom)` : charge le fichier `nom` et renvoie `(image, fmt, maxi)`
  - `image` : la matrice
  - `fmt` : `'PBM'`, `'PGM'` ou `'PPM'`
  - `maxi` : couleur maximale

Visualisation des images

Configurer votre ordinateur

Pour chacun des trois formats :

1. Clic-droit sur l'image
2. `Ouvrir avec une autre application...`
3. « Utiliser une commande personnalisée »
4. Entrer `display`
5. Cocher « Utiliser cette action par défaut pour ce type de fichier »
6. Ouvrir