Fondements de l'informatique

Arnaud Labourel
Courriel : arnaud.labourel@lif.univ-mrs.fr

Université de Provence

Quelques annonces

- Pas de cours la semaine prochaine
- Les TD et les TP commence la semaine prochaine
- Partiel le 3 ou 4 novembre
- Examen du 4 au 10 janvier 2012

But du cours

Les mathématiques au service de l'informatique

Apprendre à utiliser des notions mathématiques afin de résoudre des problèmes de l'informatique

Programme

- Systèmes de numération : premiers pas vers la notion de représentation finie
- Ensembles et dénombrement
- Relations et fonctions
- Algèbre de Boole
- Simplification de formules booléennes
- Codes correcteurs d'erreurs
- Probabilités combinatoires
- Introduction à la logique propositionnelle
- (Théorie de l'information de Shannon)

Contenu

Volume horaire

- 10 cours (20h)
- 15 TD (30h)
- 5 TP (10h, programmation en Python)

Évaluation

- Un examen (E)
- Un partiel (P)
- Une note de contrôle continu (CC)

$$\text{Note finale} = \max\left(E, \frac{2E+\max\left(P, \frac{CC+2P}{3}\right)}{3}\right)$$

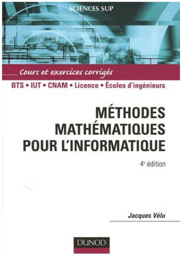
Sources

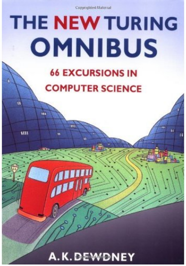
Transparents et feuilles de TD/TP

`http://www.lif.univ-mrs.fr/~labourel/FI/index.html`

Livres

- Méthodes mathématiques pour l'informatique, Jacques Vélu
- The new Turing Omnibus, A.K. Dewdey





Définition de l'informatique

Informatique ?

- Représenter
- Modéliser
- Approximer
- Calculer sur des modèles, des représentations
- Résoudre de manière efficace et précise (et le prouver)

Un exemple : prévision météorologique

Un problème : prévoir le temps

- Étape 1 : Acquisition des données
  - nombreuses données
  - sources très différentes : images satellite, relevés au sol et par ballon sonde, radar météorologique
- Étape 2 : Modèle numérique
  - équations qui régissent l'atmosphère (mécanique des fluides)
  - impossible de tout prendre en compte → approximation
- Étape 3 : Simulation
  - Calculer pas à pas l'évolution du système suivant le modèle numérique
  - De nos jours, plusieurs simulations sont lancées en parallèle avec des modèles différents

Définition de l'informatique

Une définition

Informatique = Domaine des concepts et autres techniques employées pour le traitement automatique de l'information.

Une citation

« La science informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes »

Edsger Dijkstra.

Information et codage

- Processeur = système automatique de traitement d'information
- Information = éléments tels que texte, parole, image, mesure d'une grandeur physique, nombre, etc...
- Information représentée sous une forme physique appropriée au traitement qu'elle doit subir
- Première étape essentielle : codage de l'information. Signaux (images, paroles, textes) codés in fine sous forme de 0 et de 1 (système binaire).

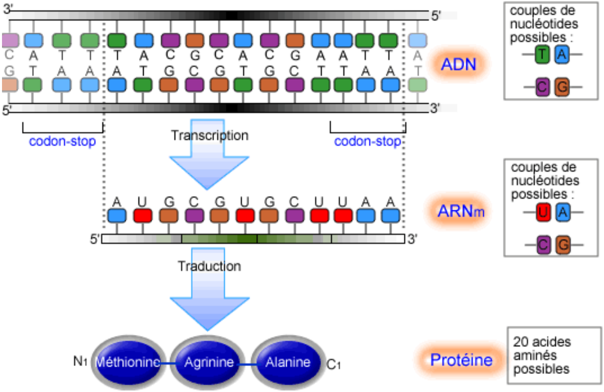
De l'information au bit

Nombreuses étapes entre le phénomène réel et son codage dans la machine.

Exemple de codage en plusieurs phases :

- En biologie : de l'ADN à la protéine.
- En informatique : du son au fichier mp3

Exemple en biologie



Exemple en multimédia

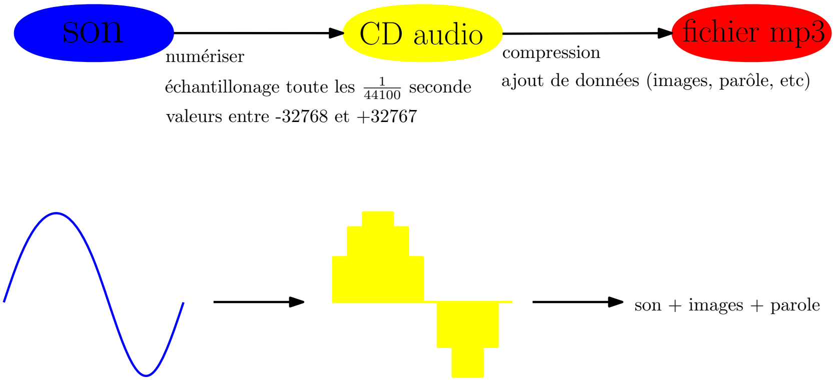


Diagramme commutatif

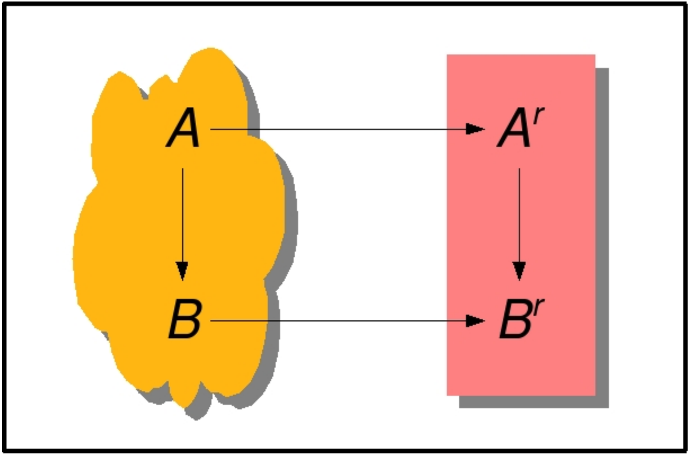
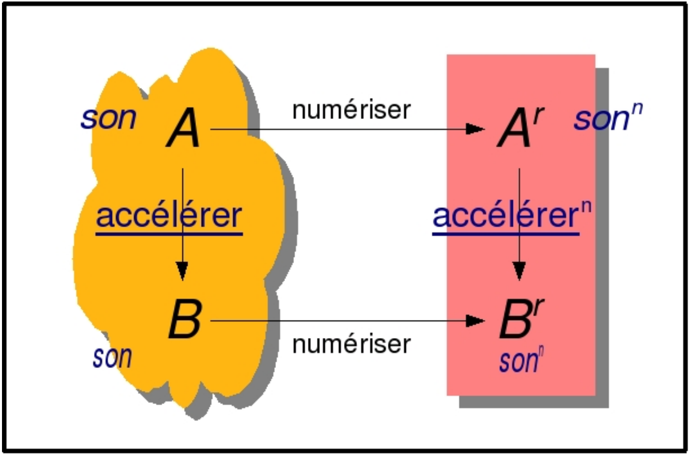


Diagramme commutatif



Modéliser : représenter en plus simple

De l'infini au fini

DD = 1To, RAM = 2Go, très grande capacité de stockage.

Modéliser et approximer

- Se ramener au fini, mais rester fidèle : représenter la modélisation dans un monde fini
- Ne pas faire d'erreur de calcul (ou les connaître)

Les systèmes de numération

Systèmes de numération

Les systèmes de numération

- Système conventionnel de comptage en base 10 incompatible avec la machine

$\Longrightarrow$ Etude d'autres systèmes de numération

- Systèmes de numération : utilisation de symboles appelés *digits*
- Le nombre de digits utilisés correspond à la base du système

Système binaire : base 2 (symboles – ou digits – 0 et 1)

Système Hexadécimal : Base 16 (symboles – ou digits – 0 à 9, et A B C D E F)

Principe d'une base

- Base : le nombre qui définit le système de numération
- Base du système décimal $= 10$, base du système octal $= 8$, etc.

Formule magique en base β

$$\sum_{i=n}^{i=0}(b_i\beta^i) = b_n\beta^n + \cdots + b_2\beta^2 + b_1\beta^1 + b_0\beta^0$$

où :

b_i est le chiffre de la base de rang i

β^i est la puissance de la base β d'exposant de rang i

Principe d'une base

$$\sum_{i=n}^{i=0}(b_i\beta^i) = b_n\beta^n + \cdots + b_2\beta^2 + b_1\beta^1 + b_0\beta^0$$

Exemple en base $\beta = 10$

$$2011 = (2 \times 10^3) + (0 \times 10^2) + (1 \times 10^1) + (1 \times 10^0)$$

rang i	3	2	1	0
chiffre b_i	2	0	1	1
élément $b_i\beta^i$	2×10^3	0×10^2	1×10^1	1×10^0

Le système décimal

- Origine : le nombre de doigts ?
- 10 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Poids du digit = la puissance de 10 qu'il multiplie
- Système de numération de position

  Les digits s'écrivent de gauche à droite, par ordre *décroissant* des puissances de 10.

- La formule magique se précise, pour un nombre x composé de n chiffres

$$(x)_{10} = \sum_{i=n}^{i=0} (b_i 10^i) = b_n 10^n + \cdots + b_2 10^2 + b_1 10 + b_0$$

où b_i est un des 10 digits.

Le système octal

- Origine : pratique car c'est une puissance de 2
- 8 digits : 0, 1, 2, 3, 4, 5, 6, 7
- On adapte la formule magique :

$$(x)_8 = \sum_{i=n}^{i=0}(b_i 8^i) = b_n 8^n + \cdots + b_2 8^2 + b_1 8 + b_0$$

- Pour lever les ambiguités : $(x)_\beta$ pour préciser le système de numération

$$(2011)_{10} = 3 \times 8^3 + 7 \times 8^2 + 3 \times 8^1 + 3 \times 8^0 = (3733)_8$$

Pour se détendre

Pourquoi les informaticiens mélangent toujours Noël et Halloween ?
$\Longrightarrow$Parce que DEC 25 $= 25_{10} =$ OCT 31 $= 31_8$

Le système binaire

Au centre de l'informatique

- 2 digits : 0 et 1 (vrai et Faux, ON et OFF, Oui et Non etc.)
- On adapte la formule magique :

$$(x)_2 = \sum_{i=n}^{i=0}(b_i 2^i) = b_n 2^n + \cdots + b_2 2^2 + b_1 2 + b_0$$

$$\begin{aligned}(2009)_{10} &= 1024 + 512 + 256 + 128 + 64 \\ &\quad +0 \times 32 + 16 + 8 + 0 \times 4 + 0 \times 2 + 1 \\ &= (11111011001)_2\end{aligned}$$

Le système binaire (cont'd)

Octets, bits et bytes

- Binary digits = Bits la plus petite unité
- Un octet = 8 bits (et en anglais : un byte)
- Un octet = la taille nécessaire pour coder en binaire un caractère parmi 256 ($256 = 2^8$)

Préfixes binaires (kilo, mega...)

- Souvent utilisés lorsqu'on a affaire à de grandes quantités d'octets : puissances de 2
- Ne pas confondre 15 Mbit et 15 Mo = 15 Mbytes...
- Dérivés (mais différents) des préfixes SI (Système International d'Unités : puissance de 10)

Préfixes binaires conventionnels

Nom	Symbole	Puissance	$\backsim$Déc	Nombre
unité		$2^0 = 1$	10^0	un
kilo	k/K	$2^{10} = 1024$	10^3	mille
mega	M	$2^{20} = 1048576$	10^6	million
giga	G	$2^{30} = 1073741824$	10^9	milliard
tera	T	$2^{40} = 1099511627776$	10^{12}	billion
peta	P	$2^{50} = 1125899906842624$	10^{15}	billiard
exa	E	$2^{60} = 1152921504606846976$	10^{18}	trillion

Exemple de confusion dans les systèmes de mesure

La délinquance des fabricants de disques dur

Un disque dur de 1 To.

- Informatique : $1024 \times 1024 \times 1024 \times 1024 = 1\,To$
- Fabricant de disque : $1000 \times 1000 \times 1000 \times 1000 \approx 0.9095\,To$

Le système hexadécimal

Pratique pour l'adressage mémoire

- Unité de RAM, puissance de 2
- 16 digits : 0,1,2,...,9, A, B, C, D, E, F
- On adapte la formule magique :

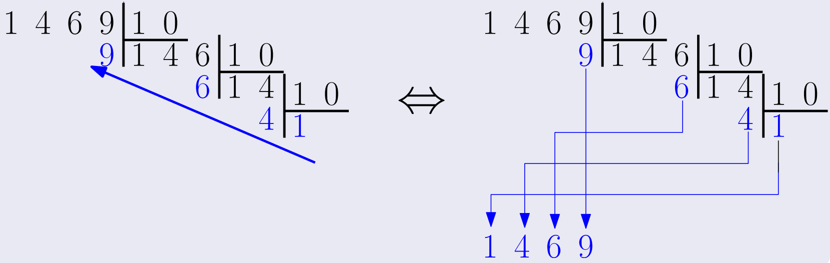
$$(x)_{16} = \sum_{i=n}^{i=0} (b_i 16^i) = b_n 16^n + \cdots + b_2 16^2 + b_1 16 + b_0$$

Exemple

$$\begin{aligned}(2009)_{10} &= 7 \times 16^2 + 13 \times 16 + 9 \\ &= 7 \times 16^2 + D \times 16 + 9 \\ &= (7D9)_{16}\end{aligned}$$

Principe général

Décomposition en puissances de 10

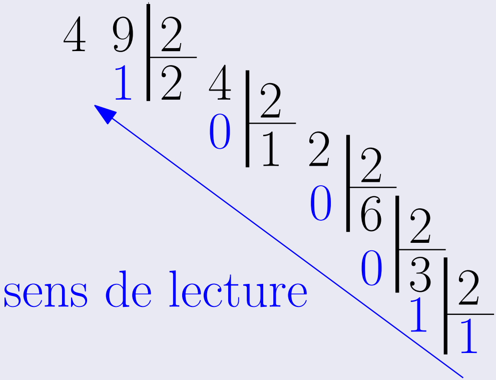


On réalise autant de divisions par 10 que nécessaires, en gardant les restes, et on lit de droite à gauche

$\Longrightarrow$ idem pour les autres bases

Conversion décimal vers binaire

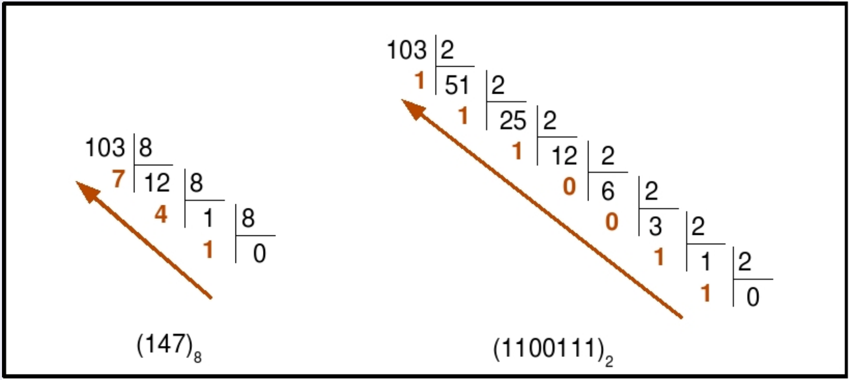
Même principe : divisons par 2, gardons les restes



$$(49)_{10} = (110001)_2$$

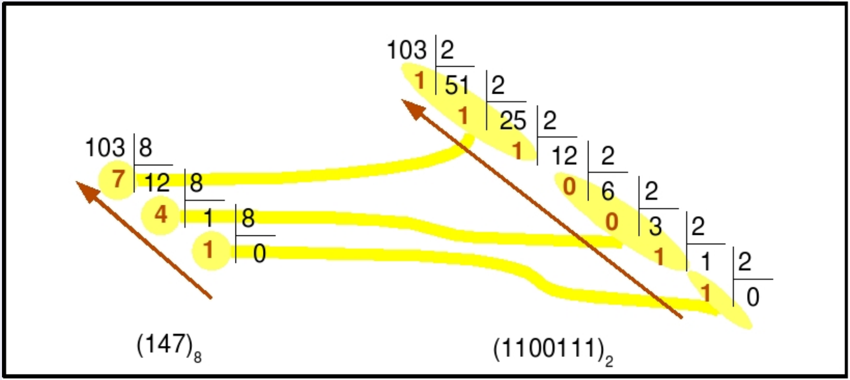
Relation binaire / octal

Examinons $(103)_{10}$ en bases 8 et 2



Relation binaire / octal

Examinons $(103)_{10}$ en bases 8 et 2



Relation binaire / puissances de 2

- Equivalence : trois bits $\Longleftrightarrow$ chiffre octal
- Conversion simplifiée $\begin{array}{l} (\quad 1 \quad 4 \quad 7 \quad)_8 \\ (\overbrace{001}\ \overbrace{100}\ \overbrace{111})_2 \end{array}$
- Autre exemple : $\begin{array}{cccccc} (& 2 & 0 & 5 & 4 &)_8 \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ (& \overbrace{010} & \overbrace{000} & \overbrace{101} & \overbrace{100} &)_2 \end{array}$
- Même type de relation entre binaire / hexa (groupe de 4 bits)

$$\begin{array}{cccccc} (& A & 5 & 0 & F &)_{16} \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ (& \overbrace{1010} & \overbrace{0101} & \overbrace{0000} & \overbrace{1111} &)_2 \end{array}$$

Relation octal / hexadecimal

Astuce

Il suffit de passer par le binaire :

- De l'octal au binaire : $$\begin{array}{cccccc} (& 3 & 7 & 1 & 2 &)_8 \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ (& \overbrace{011} & \overbrace{111} & \overbrace{001} & \overbrace{010} &)_2 \end{array}$$
- Du binaire à l'hexadécimal : $$\begin{array}{ccccc} (& \underbrace{0111} & \underbrace{1100} & \underbrace{1010} &)_2 \\ & \updownarrow & \updownarrow & \updownarrow & \\ (& 7 & C & A &)_{16} \end{array}$$

Nombre de digits pour représenter un nombre

Nombre de chiffres de x dans une base β

- Soit $x = b_n b_{n-1} b_{n-2} \cdots b_2 b_1 b_0$ écrit dans la base β.
- Si $b_n \neq 0$, on note $n+1 = N_\beta(x)$ le nombre de chiffres nécessaires pour exprimer x dans la base β.
- Estimons $N_\beta(x)$
- Exemple : $\beta = 2$, combien de bits pour exprimer $(45)_{10}$?

Théorème

$N_\beta(x)$ est le plus petit entier strictement supérieur à $\mathrm{Log}_\beta(x)$

Nombre de digits pour représenter un nombre (cont'd)

Théorème

$N_\beta(x)$ est le plus petit entier strictement supérieur à $\mathrm{Log}_\beta(x)$

Exemple pour $\beta = 2$ et $x = (1503)_{10}$

$\ln(1503) = 7,32$ et $\ln(2) = 0,693$
Donc $\mathrm{Log}_2(1503) = \frac{7,32}{0,693} = 10,6$ et $N_2(1503) = 11$
Donc il faut 11 bits pour représenter $(1503)_{10}$ en binaire. En effet, ce nombre s'écrit, en base 2 :

$$10111011111$$

Systèmes de numération meilleurs que les bâtons

Avec les bâtons, $56_{10} = ||||||||||||||||||||||||||||||||||||||||||||||||||||||||$

Indifférence de la base β pour x grand

Théorème

Le rapport $\frac{N_\beta(x)}{N_{\beta'}(x)}$ tend vers $\frac{\mathrm{Log}(\beta')}{\mathrm{Log}(\beta)}$ quand x tend vers l'infini

Exemple

Pour écrire un grand nombre en base 2, il faut environ $3,32$ fois plus de digits qu'en base 10 car

$$\frac{\ln(10)}{\ln(2)} = 3,32...$$

Bienfaits de la représentation binaire

Calculer y^x, si x entier positif

$y^2 = y \times y$ puis $y^3 = y^2 \times y$ puis $y^4 = y^3 \times y$

Méthode plus astucieuse et plus rapide

1. Rep. x en binaire : $x = b_n 2^n + b_{n-1}2^{n-1} + \cdots + b_1 \times 2 + b_0$
2. Du coup, $$\begin{aligned} y^x &= y^{b_n 2^n + b_{n-1}2^{n-1} + \cdots + b_1 \times 2 + b_0} \\ &= (y^{2^n})^{b_n}(y^{2^{n-1}})^{b_{n-1}} \cdots (y^2)^{b_1}(y)^{b_0} \end{aligned}$$
3. Or on simplifie ce produit car $$(y^{2^k})^{b_k} = \begin{cases} y^{2^k} & \text{si } b_k = 1 \\ 1 & \text{si } b_k = 0 \end{cases}$$
4. On multiplie tous les y^{2^i} pour lesquels b_i n'est pas nul.

Exemple de calcul de puissance

Calculons $y^{1041} = y^1 + y^{16} + y^{1024}$

- Méhode classique
  On calcule $y^2 = y \times y$ puis $y^3 = y^2 \times y$ puis $y^4 = y^3 \times y$ puis $y^5 \times y$ et y^6 et y^7 .... y^{103} ......y^{677}..... y^{1040} et enfin y^{1041}.
  (1040 multiplications – $(x - 1)$)
- Méthode via système binaire
  On calcule $y^4 = y^2 \times y^2$ puis $y^8 = y^4 \times y^4$ puis $y^{16} = y^8 \times y^8$ puis y^{32} puis y^{64} puis y^{128} puis y^{256} et $y^{512} = y^{256} \times y^{256}$ et finalement $y^{1024} = y^{512} \times y^{512}$
  On calcule pour finir $y \times y^{16} \times y^{1024}$
  ($10 + 2$ multiplications – $< 2\mathrm{Log}_2(x)$)

Représentation des nombres

De l'infini au fini

Les nombres en mathématiques

Une infinité d'entiers naturels ($\mathbb{N}$), une infinité de réels ($\mathbb{R}$), et la précision des irrationnels

Les nombres en informatique

- Représentation dans le système binaire
- Limites matérielles
  - Quelle que soit la taille d'une disquette (sa capacité), il y aura toujours un entier qui ne pourra pas y être stocké.

    Soit une disquette de taille 3 kBits $= 3 \times 1024 = 3072$ chiffres binaires : pas d'entier $> 2^{3072}$.
  - Représentation approchée des réels

    Pas de représentation numérique exacte de $\sqrt{2}$: nécessité de représentations symboliques

Représentation binaire des données entières

Les entiers non-signés

- Les entiers naturels (zéro et les positifs)
- Pas de gestion du signe : codage binaire pur
- Sur un octet, on représente les entiers de 0 à $2^8 - 1 = 255$
- Sur deux octets, on représente les entiers de 0 à $2^{16} - 1 = 65535$

Représentation binaire des données entières

Les entiers signés

- Tous les entiers (dans la limite de capacité)
- Il faut un bit pour représenter le signe (positif ou négatif) et codage binaire de la valeur absolue
- Sur un octet, on représente les entiers de $-2^7 + 1 = -127$ à $2^7 - 1 = 127$
- Sur deux octets, on représente les entiers de $-2^{15} + 1 = -32767$ à $2^{15} - 1 = 255 = 32767$

Représentation binaire des données entières

Alternative : le complément à deux

- Toujours éventuellement un bit pour le signe, mais autre façon de coder la valeur absolue
  1. On exprime la valeur en base 2
  2. Tous les bits sont inversés
  3. On ajoute une unité au résultat
- Exemple (sur 2 octets, un négatif de plus) :

$$\begin{array}{ccc} 1 & = & (0000000000000001) \\ -1 & = & (1111111111111111) \\ 3 & = & (0000000000000011) \\ -3 & = & (1111111111111101) \\ 32767 & = & (0111111111111111) \\ -32767 & = & (1000000000000001) \\ -32768 & = & (1000000000000000) \end{array}$$

Représentation binaire des données entières

Répartition des entiers dans le complément à deux

positifs						négatif			
0	1	2	...	32767	-32768	...	-3	-2	-1
0000000000000000	0000000000000001	0000000000000010		0111111111111111	1000000000000000		1111111111111101	1111111111111110	1111111111111111

Pourquoi utiliser cette représentation ?

Pour simplifier l'addition !

$$
\begin{array}{r}
0000000000000011 \\
+1111111111111111 \\
\hline
\color{blue}{1}0000000000000010
\end{array}
\qquad
\begin{array}{r}
3 \\
+(-1) \\
\hline
2
\end{array}
$$

Représentation des données réelles

Une approximation

- Problématiques :
  1. assurer la précision derrière la virgule
  2. pouvoir représenter de grands nombres

  $$152140021536955471089444875,0000000000000000000000001$$

- Avant et après la virgule

Solutions classiques

- Virgule fixe
- Virgule flottante

Virgule fixe

Nombre fixe de chiffres après la virgule

- Opérations plus simple $\rightarrow$ processeurs moins chers
- Plus facile à coder dans la machine

Deux parties

- La partie entière codée en binaire (complément à deux)
- La partie décimale : chaque bit correspond à l'inverse d'une puissance de 2

Exemple

$$-3,625_{10} = \underbrace{1111111111111101}_{-3} \quad \underbrace{1010000000000000}_{0,625 = 0,5 + 0,125 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8}}$$

Capacité de représentation en virgule fixe

Représentation rigide

- Petits nombres : gaspillage des digits à gauche de la virgule
- Peu de décimales : gaspillage à droite
- Plus simple à mettre en œuvre, pour des ordres de grandeur comparables

Bornes

Si n est le nombre de bits de la partie entière, et d est le nombre de bits de la partie fractionnaire

- Borne maximale : $2^{n-1} - \frac{1}{2^d}$
- Borne minimale : -2^{n-1}

Partie décimale de $0,347$ en binaire

$0,347 \times 2 = 0,694 < 1 \Longrightarrow$ on pose 0: $0,347 = 0,0$

$0,694 \times 2 = 1,388 > 1 \Longrightarrow$ on pose 1: $0,347 = 0,01$

$0,388 \times 2 = 0,766 < 1 \Longrightarrow$ on pose 0: $0,347 = 0,010$

$0,766 \times 2 = 1,552 > 1 \Longrightarrow$ on pose 1: $0,347 = 0,0101$

$0,552 \times 2 = 1,104 > 1 \Longrightarrow$ on pose 1: $0,347 = 0,01011$

$0,104 \times 2 = 0,208 < 1 \Longrightarrow$ on pose 0: $0,347 = 0,010110$

$0,208 \times 2 = 0,416 < 1 \Longrightarrow$ on pose 0: $0,347 = 0,0101100$

$0,416 \times 2 = 0,832 < 1 \Longrightarrow$ on pose 0: $0,347 = 0,01011000$

$0,832 \times 2 = 1,664 > 1 \Longrightarrow$ on pose 1: $0,347 = 0,010110001$

$0,664 \times 2 = 1,328 > 1 \Longrightarrow$ on pose 1: $0,347 = 0,0101100011$

Virgule flottante

Solution la plus répandue

- Norme IEEE 75 : deux formats (32bits et 64bits) selon précision (simple et double)
- Ordinateurs actuels : implémentation matérielle de ce mode de représentation (dans le micro-processeur)

Un triplet

Le signe s – La mantisse m – L'exposant e : $x = sm\beta^e$

$$-1540,412654 = -1,540412654.10^3 = -0,1540412654.10^4$$

$$101110,001101 = 1,01110001101.2^5$$

Virgule flottante (cont'd)

Remarques

- Mantisse de taille fixée
- On fait flotter la virgule en faisant varier e
- Base souvent 2 (Héxa chez anciennes machines, 10 chez certaines calculatrices)

Précisions $1 \leq m < 2$

	taille	signe	e	m	valeur
Simple précision	32b	1	8	23	$-1^s m 2^{e-127}$
Double précision	64b	1	11	52	$-1^s m 2^{e-1023}$

Exemple du nombre $-6,625_{10}$

En simple précision

- Binaire (valeur absolue) : $110,101$
- Normalisation de la mantisse : $1,10101.2^2$
- Occupation des 24 bits de mantisse
  $1,\underbrace{10101000000000000000000}$
- Décalage de l'exposant (S.P. $= 127$) donc exposant $= (2+127)_{10} = 10000001_2$
- Signe $= 1$ car négatif

1	10000001	10101000000000000000000

Approximations et erreurs

Exemples introductifs

Chez les entiers

- Supposons les entiers positifs codés sur 2 octets (de 0 à 65535)
- Calculons $(39001 + 27446)_{10} = 66447_{10}$

$$\begin{array}{rr} & 1001100001011001 \\ + & 0110101100110110 \\ \hline = & 1\,\underbrace{0000001110001111}_{=(911)_{10}} \end{array}$$

- $\Longrightarrow$ Dépassement de capacité

Exemples introductifs (cont'd)

Chez les réels (flottants)

- Considérons la représentation en virgule flottante D.P.
- Le nombre $2^{60} + 1$ n'est pas représentable (pas assez de capacité), et approximé par 2^{60}

$$1,000000000000000000....000001$$

- Conséquence : $(2^{60} + 1) - 2^{60} = 0$

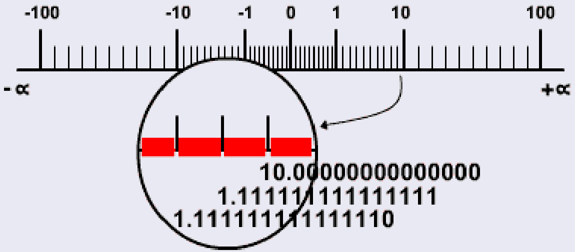
Dispositions particulières (exposant)

- bit d'information : NaN : Not a Number (avec propagation)
- bit d'information : +INF et -INF (avec propagation)

Intervalle et précision

Infiniment petit et infiniment grand

Densité des réels (ici codés en 16bits)



Ecart entre x et sa représentation

Estimation de l'approximation

$$\frac{\Delta x}{x} = \frac{\Delta m}{m} \leq \frac{\beta^{-N}}{\beta^{-1}} = \beta^{1-N}$$

Exemple en base 2

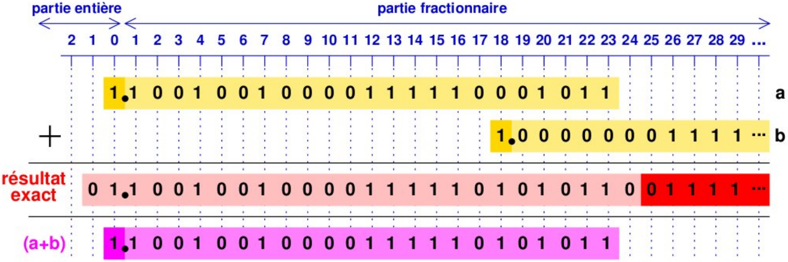
- En simple précision, mantisse sur 23 bits

$$\frac{\Delta x}{x} \leq 2^{-22} = 4194304_{10} \backsim 4,2.10^{6}$$

- En double précision, mantisse sur 52 bits

$$\frac{\Delta x}{x} \leq 2^{-51} = 2251799813685248_{10} \backsim 2,3.10^{15}$$

Illustration du phénomène d'absorption



Conséquences de ces approximations

Erreurs d'arrondis

Lorsque N est dépassé, approximation :

- arrondi vers la décimale la plus proche
- troncature

$\Longrightarrow$ Calcul flottant non-associatif

$$\sum_{i=1}^{n} \frac{1}{i} \neq \sum_{i=n}^{1} \frac{1}{i}$$

Somme des inverses : leçon

Pour $n = 10^9$

- Simple précision (32 bits)
  - $\sum_{i=1}^{n} \frac{1}{i} \to 15.403$
  - $\sum_{i=n}^{1} \frac{1}{i} \to 18.807$
- Double précision (64 bits)
  - $\sum_{i=1}^{n} \frac{1}{i} \to 21.3004815023\underline{85}$
  - $\sum_{i=n}^{1} \frac{1}{i} \to 21.3004815023\underline{61}$

Règle générale

Sommer en premier les termes ayant la plus petite valeur absolue

Phénomène de compensation (élimination)

Erreurs d'approximation liées à la soustraction

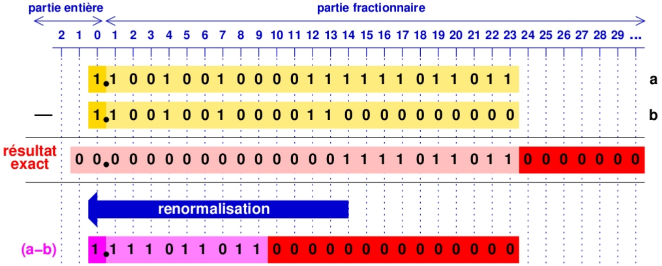
Les termes sommés s'annulent si trop proches

$$e^{-10} \approx \sum_{k=0}^{n} (-1)^k \frac{10^k}{k!} \rightsquigarrow e^{-10} \approx -1,295050418187$$

Eviter les sommations dans lesquelles les termes de signes opposés se compensent

$$e^{10} \approx \sum_{k=0}^{n} \frac{10^k}{k!} \rightsquigarrow e^{-10} = \frac{1}{e^{10}} \approx 0,000045401$$

Visualisation de l'élimination



Coefficient d'amplification

- Deux façons de calculer

$$I_n = \int_0^1 x^n e^{-x} dx = -e^{-1} + nI_{n-1} \text{ et } I_0 = 1 - e^{-1}$$

  qui converge vers 10^3
  1. En montant (de I_0 à I_n)
  2. En descendant, de $I_{4n} = a$ quelconque à I_n
- Bizarrement, c'est la seconde solution qui converge
- Coefficient d'amplification de l'erreur d'arrondi

$$0,000001 \text{ au lieu de } 0,000001\boxed{41269}$$

  Après multiplication par 1000

$$0,010000 \text{ au lieu de } 0,0141269$$

  $\Longrightarrow$ mieux vaut diviser...

Missiles Patriot : 0,34s de retard et 28 morts

En février 1991, guerre du Golfe : un missile Patriot (tueur de missiles Scud) tue 28 soldats

L'erreur : accumulation d'arrondis

- Nombres en virgule fixe sur 24 bits
- Temps compté par horloge interne en 1/10 de seconde

$$1/10 = 0,1_{10} = 0,000110011001100110011001100110011..._2$$

- Arrondi à 24 chiffres : erreur tous les $10^{\text{èmes}}$ de sec.
- Au moment de l'attaque, batterie du Patriote démarrée depuis 100h donc accumulation de 0,34s d'erreur
- Vitesse Scud $\approx$ 1 km/s : cible ratée

Explosion d'Ariane 5

Juin 1996, explosion d'Ariane 5 (1 milliard de dollars)

Conversion et approximation sévère

- Reprise à l'identique du logiciel d'Ariane 4 pour la gestion des centrales de guidage : nombres de 16 bits en entiers signés
- Tout le reste d'Ariane 5 : nombres de 64 bits en virgule flottante
- Passage d'un système à l'autre de la vitesse horizontale de la fusée par rapport à la plate-forme de tir : plus grande que 32767
- Effet modulo et déviation de la trajectoire impossible à rectifier

Autres bugs célèbres

Pentium d'Intel, 1994

Erreur dans la table de référence des divisions

Conversion Euro

Erreurs d'arrondis, amplifiées par opérations de conversion et de reconversion avant totalisation !

Augmentation du Vancouver Stock Exchange (1983)

(alors que les prix n'ont pas varié) Troncature de 4 vers 3 décimales et amplification quotidienne