Initiation à l'algorithmique

Représentation et codage des données

Antoine Meyer

6 décembre 2010

Problématique

Données informatiques : tout ce qui n'est pas un programme

- Dans la vie : documents, images, sons, vidéos...
- Dans un algorithme : nombre entier, flottant, caractère... tableau, liste, pile, arbre, graphe !
- Dans un ordinateur : données brutes (*tas* de valeurs numériques)

Problème : comment passe-t-on d'un niveau à l'autre ?
(question de la *structuration* des données)

Problématique

Notion de structure de données

- Représentation appropriée des données pour un but fixé
- Arbitrage : complexité des opérations fréquentes

Exemples de traitements particuliers des données :

- Compression (ZIP, MP3, JPG, MPEG, ...)
- Cryptographie (chiffrement, paiement sécurisé, signature numérique, ...)
- Correction d'erreurs (disques redondants, lecture de CD-ROM, transmissions spatiales, ...)

Plan

Plan

Unités et vocabulaire

Unité la plus petite : le *bit* (0 ou 1)

- Lié à la technologie matérielle (composants, stockage)
- Favorise une omniprésence de la numération en base 2

Unité usuelle : l'octet (= 8 bits)

- Permet de représenter $2^8 = 256$ valeurs différentes
- Ordre de grandeur : un caractère (lettre, symbole)

Autres unités courantes : multiples du bit ou de l'octet (puissances de 2 et non de 10 !)

Unités et vocabulaire

Unité	Valeur	Ordre de grandeur
kilo-octet	1 Ko $= 2^{10}$ o $= 1024$ o	un mél
mega-octet	1 Mo $= 2^{20}$ o $= 1024$ Ko	une disquette
giga-octet	1 Go $= 2^{30}$ o $= 1024$ Mo	un DVD
tera-octet	1 To $= 2^{40}$ o $= 1024$ Go	un (gros) disque dur

Unités et vocabulaire

Quel ordre de grandeur ?

- Cet exposé
- Une photo numérique
- Le débit d'une connexion ADSL
- Un système d'exploitation moderne
- Ce que Google connait de l'Internet

Unités et vocabulaire

Quel ordre de grandeur ?

- Cet exposé — moins de 100 Ko
- Une photo numérique — quelques Mo
- Le débit d'une connexion ADSL — environ 10Mbps
- Un système d'exploitation moderne — quelques Go
- Ce que Google connait de l'Internet — 200 To (soit moins de 0,01% du total estimé)

La mémoire d'un ordinateur

Données stockées de multiples façons :

- Semi-conducteurs (RAM, clé USB)
- Support magnétique (bande, disquette, disque dur)
- Support optique (CD-ROM, DVD, BluRay)

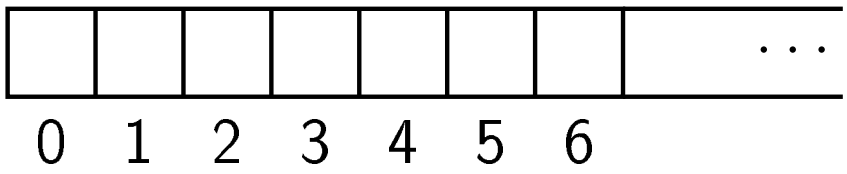
Caractéristiques variables :

- Persistance des données ou non
- Temps d'accès, taux de transfert...

La mémoire d'un ordinateur

Concept abstrait :

- Succession de "cellules" numérotées



- Numéro d'une cellule : adresse
- Taille des cellules : une "unité d'information"
- Nombre de cellules : virtuellement infini
- Accès en lecture ou écriture en temps constant
- Modèle théorique : machine de Turing

La mémoire d'un ordinateur

En réalité :

- Taille d'une cellule fixée (par ex. 32 bits / 4 o)
- Nombre de cellules borné
- Structure (beaucoup) plus complexe
- Temps d'accès très variables

Codage des nombres entiers

Entiers positifs représentés directement en binaire :

$$(b_n \dots b_0)_2 = \sum_{i=0}^{n} b_i 2^i \quad \text{avec} \quad b_i \in \{0,1\}$$

- Exemple : $(10101)_2$ représente 42

Entiers négatifs en *complément à 2* :

- Exemple : $(10101)_2$ représente aussi -11

Pour faire la différence, savoir de quoi on parle

- En math : (x,y) n'est pas le même point du plan selon le système de coordonnées choisi
- Notion de *type* d'une valeur ou variable

Spécificités techniques

En général, taille fixée à l'avance

- Entiers sur 1 octet de 0 à 255 ou -128 à 127
- Entiers sur n bits de 0 à $2^n - 1$ ou -2^{n-1} à $2^{n-1} - 1$
- Sur les doigts ?

Avantages :

- Simplicité
- Opérations calculées par matériel

Inconvénients :

- Problèmes de débordement ($255 + 1 = \ldots 0$!)
- Calculs avec grands entiers difficiles !

Liens avec les math (de primaire au lycée)

- Vue sous un autre angle du système de numération positionnel
- Changements de base (autres bases courantes : octal, hexadécimal)
- Retour sur les opérations arithmétiques
- Arithmétique modulaire (p. ex. dépassements de valeur)
- Logique et opérateurs booléens
- Etc.

Codage des caractères

Idée : caractère $\leftrightarrow$ représentation en binaire (i.e. un nombre)

- Norme ASCII : caractères codés sur 7 bits (128 valeurs)
- Norme UTF-8 : caractères codés sur 1 à 4 octets (plusieurs milliers de symboles)

Attention à utiliser le même codage pour écrire et pour lire :

> *Ãa m'Ã©nerve ces chaÃ®nes de caractÃ"res qui dÃ©raillent !*

Chaîne de caractères : juxtaposition des codes (p. ex. octets) des caractères la composant, terminée par un symbole spécial

Codage des caractères

Table des caractères ASCII :

b_7 →					0	0	0	0	1	1	1	1
b_6 →					0	0	1	1	0	0	1	1
b_5 →					0	1	0	1	0	1	0	1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Codage des nombres « réels »

Représentation de nombres décimaux en base 2 :

$$(b_n \dots b_0, b_{-1} \dots b_{-m})_2 = \sum_{i=-m}^{n} b_i 2^i$$

Problèmes :

- On dispose d'une taille fixée pour chaque nombre $\rightarrow$ nécessité d'approximation
- Certains nombres décimaux en base 10 ne sont pas représentables en base 2 (par exemple : 0.1)
  Exercice : trouver les premiers chiffres après la virgule de 0.1 exprimé en base 2
- Idéalement, on souhaiterait faire des calculs fiables et justes sur les réels !

Codage des nombres « réels »

Représentation des décimaux binaires sur n bits :

- Solution 1 : virgule fixe
  - k bits pour la partie entière
  - $n - k$ bits pour la partie fractionnaire
  - Peu d'étendue : environ 2^k au maximum
  - Précision : environ 2^{n-k}, régulière
- Exemple : 0011,0010

Codage des nombres « réels »

Représentation des décimaux binaires sur n bits :

- Solution 2 : virgule flottante
  - Equivalent de la notation scientifique ($\pm m.2^e$)
    (m : mantisse, e : exposant)
  - 1 bit de signe, k pour l'exposant, $n - k - 1$ pour la mantisse
  - Bonne étendue : environ 2^{k-1} au maximum
  - Précision : variable (médiocre autour de zéro, pire ailleurs)
  - Format standardisé, le plus utilisé (IEEE 754)
- Exemple : $signe = 0; e = 1; m = 1{,}1001001$

Imprécision en calcul flottant

Quelques exemples de calculs imprécis en virgule flottante

- $3 * 0.1 \neq 0.3$ (!)
- $0.1^2 \neq 0.01$
- $\sin(\pi) \neq 0$
- $\tan(\pi) \neq +\infty$
- $(a + b) + c \neq a + (b + c)$
- $(a + b)c \neq ac + bc$
- Etc.

Imprécision en calcul flottant

Calcul approché de π par la méthode d'Archimède. Soit

$$t_0 = \frac{1}{\sqrt{3}} \qquad t_{i+1} = \frac{\sqrt{t_i^2 + 1} - 1}{t_i} \tag{1}$$

$$t_{i+1} = \frac{t_i}{\sqrt{t_i^2 + 1} + 1} \tag{2}$$

On a

$$\lim_{i \to \infty} 6 \times 2^i \times t_i = \pi.$$

Selon la formule (1 ou 2) utilisée, on obtient des résultats très différents !

Imprécision en calcul flottant

Itération	Formule (1)	Formule (2)
0	3.4641016151377543863	3.4641016151377543863
1	3.2153903091734710173	3.2153903091734723496
2	3.1596599420974940120	3.1596599420975006733
3	3.1460862151314012979	3.1460862151314352708
4	3.1427145996453136334	3.1427145996453689225
5	3.1418730499801259536	3.1418730499798241950
6	3.1416627470548084133	3.1416627470568494473
7	3.1416101765997805905	3.1416101766046906629
8	3.1415970343230776862	3.1415970343215275928
9	3.1415937488171150615	3.1415937487713536668
10	3.1415929278733740748	3.1415929273850979885
11	3.1415927256228504127	3.1415927220386148377
12	3.1415926717412858693	3.1415926707019992125
13	3.1415926189011456060	3.1415926578678454728
14	3.1415926717412858693	3.1415926546593073709
15	3.1415919358822321783	3.1415926538571730119
16	3.1415926717412858693	3.1415926536566394222
17	3.1415810075796233302	3.1415926536065061913
18	3.1415926717412858693	3.1415926535939728836
19	3.1414061547378810956	3.1415926535908393901
20	3.1405434924008406305	3.1415926535900560168
21	3.1400068646912273617	3.1415926535898608396
22	3.1349453756585929919	3.1415926535898122118
23	3.1400068646912273617	3.1415926535897995552
24	3.2245152435345525443	3.1415926535897968907
25		3.1415926535897962246
26		3.1415926535897962246

Plan

Compression : exemple introductif

Représentation naïve d'une photographie

- "Point" sur une image : pixel (*picture element*)
- Photographie numérique moderne : $\approx 10 Mpixels$
- Codage de chaque couleur : $3 \times 8 = 24$ bits (rouge, vert, bleu)
- Sans compression du tout, plus de 30 Mo !
- Avec les compressions actuelles, environ 10 fois moins
- Avec la vidéo, écarts encore plus spectaculaires

Compression de données

Autres usages fréquents :

- Compression de musique
- Compression de données textuelles ou diverses
  - Archivage
  - Diffusion

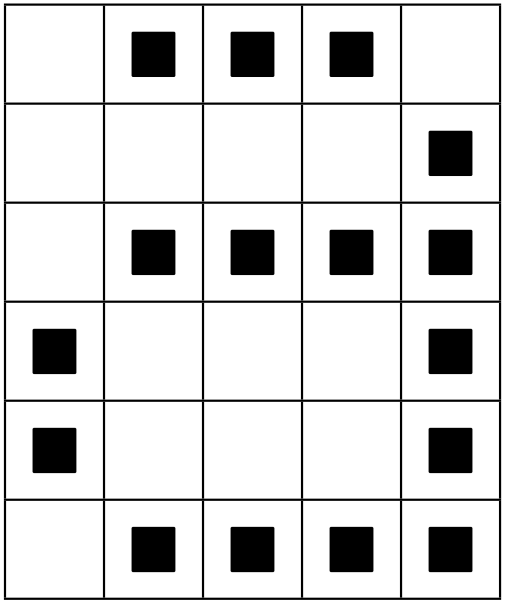
Types d'algorithmes de compression :

- Avec perte d'information (destructive)
  souvent spécialisés, p.ex. MP3, JPEG, MPEG
- Sans perte (bijective)
  décompression exacte, p.ex. ZIP, FLAC

Quelques algorithmes sans perte

Codage par plages

- Représente une succession de valeurs identiques par la valeur suivie du nombre d'occurrences
- Exemple simple : codage d'un fax en noir et blanc (exemple tiré de csunplugged.org)



1,3,1
4,1
1,4
0,1,3,1
0,1,3,1
1,4

Quelques algorithmes sans perte

Codage de Huffman

- Idée : analyse statistique des données à coder
- Chaque symbole peut être codé avec une taille différente
- Plus un symbole est fréquent, plus il reçoit un code court
- Algorithme à la base de nombreux outils modernes

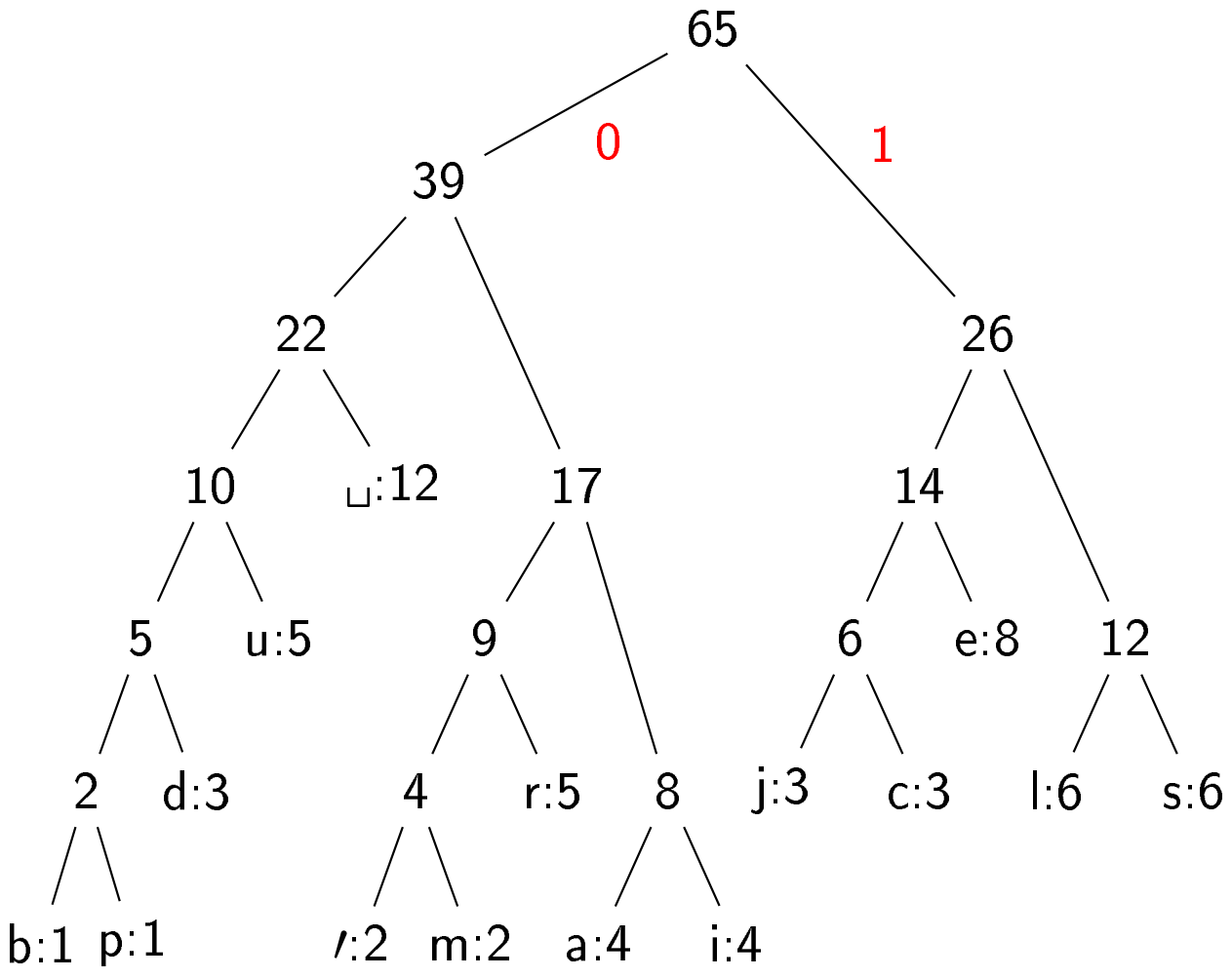
Exemple : *« j'aime aller sur le bord de l'eau les jeudis ou les jours impairs »*

Codage de Huffman

b	1
p	1
’	2
m	2
d	3
j	3
o	3
a	4
i	4
r	5
u	5
l	6
s	6
e	8
␣	12

Codage de Huffman

b	1
p	1
/	2
m	2
d	3
j	3
o	3
a	4
i	4
r	5
u	5
l	6
s	6
e	8
␣	12



Quelques algorithmes sans perte

Codage de type Lempel-Ziv

- Idée : repérer les fragments répétés dans le texte
- Chaque nouvelle occurrence d'un fragment de 2 lettres ou plus est remplacée par un lien vers la première occurrence
- Très nombreuses variantes, également très utilisé

Exemple : « *un chasseur sachant chasser doit savoir chasser sans son chien* »

Détection et correction d'erreurs

Enjeu

- Transmissions critiques (sécurité par exemple)
- Réseau ou support non fiable (disques optiques, transmissions spatiales, réseau type Internet)
- Résistance aux pannes

Idée principale

- Ajout d'information redondante au message
- Spécification d'un algorithme de vérification et/ou correction

Applications : CD-ROM, disques dur redondants, ASCII + parité, numéros de sécurité sociale, de compte en banque, ISBN, codes barres...

Quelques exemples

- Bit de parité (détection d'au plus une erreur) : ASCII sur 8 bits
- Bits de parité bi-dimensionnels (détection et correction d'au moins une erreur) : exemple du tableau de cartes
- Somme de contrôle décimale (exemple de hachage) : numéro ISBN