Informatica per Statistica
Riassunto della lezione del 04/10/2013

Igor Melatti

Il sistema binario e la rappresentazione di informazioni

- Tra le varie attività che gli competono, un computer ha anche necessità di memorizzare *informazioni*
- È chiaro che, per farlo, è necessario *rappresentare* tali informazioni in qualche modo
- All'interno di un computer, per motivi ingegneristici, ogni informazione viene rappresentata come sequenza (talvolta chiamata *stringa*) di 0 ed 1 (chiamati *bit*)
  - numeri di vari tipi
    - interi senza segno (notazione binaria)
    - interi con segno (notazione binaria in complemento a due)
    - numeri reali a precisione finita (standard IEEE 754, che combina notazione binaria con la notazione mantissa–esponente)
  - caratteri alfanumerici, comprensivi di punteggiatura (ASCII, Unicode)
  - immagini (JPG, PNG, GIF, ...)
  - video (AVI, MPEG, ...)
  - testi formattati, ovvero con interruzioni di pagina, tabelle, grassetti, font, ... (DOC, ODT, ...)
  - banche dati (MDB, ODB, ...)
  - sequenze di istruzioni in linguaggio macchina (EXE, ELF, ...)
  - ...
- Tutte le possibilità elencate sopra sono, in pratica, delle descrizioni di come organizzare stringhe di bit per rappresentare numeri, caratteri, immagini etc etc

- Esistono opportuni programmi capaci di generare tali stringhe, e di interpretare opportunamente stringhe già esistenti
  - ad esempio, il Photoshop di Windows, o il GIMP di Linux (o motissimi altri programmi analoghi), sono in grado di generare la sequenza di bit necessaria a rappresentare un'immagine disegnata con i loro strumenti secondo un formato ben preciso (JPG, PNG, GIF, ...)
  - lo stesso sa fare anche un software che gestisca uno scanner, o un software presente su uno smartphone per fare foto
  - Photoshop e GIMP sono anche in grado di, data una stringa di bit rappresentante un'immagine (stringa che si trova memorizza in un file su disco), mostrare l'immagine corrispondente
  - questo perché tutti questi programmi grafici sono stati scritti in modo da "conoscere" gli "standard" delle immagini (appunto indicate da acronimi come BMP, JPG, PNG, GIF, a seconda delle caratteristiche)
  - in questo caso, lo standard specifica come organizzare una stringa di bit in modo da rappresentare un'immagine
- Quando le informazioni sono rappresentate da lunghe stringhe di bit, spesso queste stringhe sono a loro volta rappresentate usando la base ottale e l'esadecimale, come vedremo
  - giusto per avere un'idea: se una fotocamera fa una foto a 1MB, vuol dire che genera una sequenza di circa 8 milioni di bit
- Per quanto riguarda i numeri naturali (interi positivi)
  - si usa quella che viene chiamata *notazione posizionale*
  - da notare che la *notazione decimale*, usata ogni giorno per rappresentare i numeri in tutto il mondo, non è altro che un caso particolare della notazione posizionale
  - le notazioni binaria, ottale ed esadecimale che verranno esposte qui di seguito sono altri casi particolari della notazione posizionale
  - l'idea della notazione posizionale è quella di partire da una *base* $b > 1$ (10 per la notazione decimale, 2 per quella binaria, 8 per quella ottale, 16 per quella esadecimale) e da dei simboli elementari chiamati *cifre*
    - le cifre dovranno essere esattamente b
    - usando le cifre arabo-indiane, si va da 0 a $b - 1$
    - se $b > 10$, allora si usano altri simboli, ad es. lettere maiuscole
    - infatti, nella notazione esadecimale si usano, come cifre, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, D, C, E, F
  - con questa premessa, è possibile rappresentare qualsiasi numero come sequenza di cifre

```
1 DaDecimaleABinario(n ∈ ℕ)
2   H = n.
3   Finché  H ≠ 0 ripeti i seguenti passi:
4     K = H mod 2, H = H div 2.
5     Dai in output il bit K,
6       posizionandolo a sinistra del bit precedente.
```

Figure 1: Conversione da Decimale a Binario

```
1 DaDecimaleABinarioMigliore(n ∈ ℕ)
2 H = n.
3 Ripeti i seguenti passi:
4   K = H mod 2, H = H div 2.
5   Dai in output il bit K,
6     posizionandolo a sinistra del bit precedente.
7 Finché  H ≠ 0
```

Figure 2: Conversione da Decimale a Binario (versione più completa)

- se si vuole conoscere il valore di una sequenza di cifre $c_n \dots c_0$ (ovvero, data una rappresentazione, trovare il numero rappresentato), esso può essere trovato applicando la seguente formula

$$\text{Valore}(c_n \dots c_0) = \sum_{i=0}^{n} b^i c_i$$

- per esempio, il valore di 123_{10} è $\text{Valore}(123_{10}) = \sum_{i=0}^{2} 10^i c_i = 3 \cdot 10^0 + 2 \cdot 10^1 + 1 \cdot 10^2 = 3 + 20 + 100 = 123$
- con il sistema binario, si hanno solo le cifre binarie 0 e 1, dette *bit* dall'inglese *b(inary dig)it*
- vale sempre la stessa formula di sopra per conoscere il valore di una stringa di bit
- ad esempio, il valore di 1111011_2 è $\text{Valore}(1111011_2) = \sum_{i=0}^{6} 2^i c_i = 1{\cdot}2^0{+}1{\cdot}2^1{+}0{\cdot}2^2{+}1{\cdot}2^3{+}1{\cdot}2^4{+}1{\cdot}2^5{+}1{\cdot}2^6 = 1{+}2{+}8{+}16{+}32{+}64 = 123$
- tale regola può quindi essere usata, in pratica, per convertire un numero (intero e positivo) da binario a decimale
- per fare il passo inverso, si può usare l'algoritmo 1 per convertire un numero dalla rappresentazione in base 2 a quella in base 10
- l'algoritmo 1 non dà nessun output per il caso in cui $n = 0$
- può essere corretto, ottenendo l'algoritmo 2

- ecco un esempio di come possa essere applicato l'algoritmo 2 per trovare la rappresentazione in base 2 di 123_{10}

	$H = 123$	output $= \emptyset$
$K = 123 \bmod 2 = 1$	$H = 123 \operatorname{div} 2 = 61$	output $= 1_2$
$K = 61 \bmod 2 = 1$	$H = 61 \operatorname{div} 2 = 30$	output $= 11_2$
$K = 30 \bmod 2 = 0$	$H = 30 \operatorname{div} 2 = 15$	output $= 011_2$
$K = 15 \bmod 2 = 1$	$H = 15 \operatorname{div} 2 = 7$	output $= 1011_2$
$K = 7 \bmod 2 = 1$	$H = 7 \operatorname{div} 2 = 3$	output $= 11011_2$
$K = 3 \bmod 2 = 1$	$H = 3 \operatorname{div} 2 = 1$	output $= 111011_2$
$K = 1 \bmod 2 = 1$	$H = 1 \operatorname{div} 2 = 0$	output $= 1111011_2$

- per rappresentare un numero n in base 2, occorrono $\lfloor \log_2 n \rfloor + 1$ bits
  - per un numero (reale) r, fare $\lfloor r \rfloor$ vuol dire prendere il più grande intero minore od uguale ad r (in pratica, si arrotonda sempre all'intero più basso, e se è già intero non occorre far nulla), quindi $\lfloor 3.999999 \rfloor = 3$, $\lfloor 10.000001 \rfloor = 10$, $\lfloor 4 \rfloor = 4$, ...
  - analogamente, per un numero (reale) r, fare $\lceil r \rceil$ vuol dire prendere il più piccolo intero maggiore od uguale ad r (in pratica, si arrotonda sempre all'intero più alto, e se è già intero non occorre far nulla), quindi $\lceil 3.999999 \rceil = 4$, $\lceil 10.000001 \rceil = 11$, $\lceil 4 \rceil = 4$, ...
  - quindi $\lfloor r \rfloor + 1 = \lceil r \rceil$ se r non è intero, altrimenti $\lfloor r \rfloor = \lceil r \rceil$ se r è intero
- e servono $\lfloor \log_{10} n \rfloor + 1$ cifre decimali per rappresentarlo in base 10
  - in generale, servono $\lfloor \log_k n \rfloor + 1$ cifre in base k per rappresentare n in base k
- dato che $\log_2 n > \log_{10} n$, ci possono volere troppi bit per rappresentare numeri relativamente piccoli
  - per rappresentare un milione, occorrono 20 bit (e solo 7 cifre decimali)
- è possibile usare le basi 8 e 16 (non a caso potenze di 2...) per accorciare le rappresentazioni di numeri in binario, usando una conversione diretta
  - per rappresentare un milione, occorrono 20 bit
  - con la notazione in base 16, si riduce la lunghezza delle rappresentazioni di un fattore 4 (perché $2^4 = 16$...); con la notazione in base 8, si riduce la lunghezza delle rappresentazioni di un fattore 3 (perché $2^3 = 8$...)
  - quindi, occorrono $\lceil 20/4 \rceil = 5$ cifre esadecimali per rappresentare un milione, mentre occorrono $\lceil 20/3 \rceil = 7$ cifre ottali per lo stesso numero
  - per la conversione da base 2 a base 16, basta dividere il numero in pezzetti da 4 bit, per poi convertire singolarmente ciascuna cifra

        - quindi, 0000_2 diventa 0_{16}, 0001_2 diventa 1_{16}, ..., 1001_2 diventa 9_{16}, 1010_2 diventa A_{16}, 1011_2 diventa B_{16}, 1100_2 diventa C_{16}, 1101_2 diventa D_{16}, 1110_2 diventa E_{16}, 1111_2 diventa F_{16}
    - esempio: $10011010101111100100012 = 26AF91_{16} = 11527621_8$, dal momento che, suddividendo a gruppi di 4, si ha 0010 0110 1010 1111 1001 0001, e $0001_2 = 1_{16}$, $1001_2 = 9_{16}$, $1111_2 = F_{16}$, $1010_2 = A_{16}$, $0110_2 = 6_{16}$ e $0010_2 = 2_{16}$
    - per la conversione inversa, basta trasformare ogni cifra esadecimale in un gruppo di 4 bit
        - quindi, 0_{16} diventa 0000_2, 1_{16} diventa 0001_2, ..., 9_{16} diventa 1001_2, A_{16} diventa 1010_2, B_{16} diventa 1011_2, C_{16} diventa 1100_2, D_{16} diventa 1101_2, E_{16} diventa 1110_2, F_{16} diventa 1111_2
    - esempio: $FA3_{16} = 111110100011_2$, dal momento che $3_{16} = 0011_2$, $A_{16} = 1010_2$ e $F_{16} = 1111_2$.
    - per la base 8, vale lo stesso discorso, ma si usano gruppi di 3 bit

- Caratteri: codice ASCII
    - nel codice ASCII ogni carattere è un *byte*, ovvero una sequenza di 8 bit
    - questo ha fatto sì che il byte diventasse l'unità di misura delle memorie
    - cioè, dato che tutto dentro un computer è sotto forma di bit, anche le memorie (RAM, dischi o altro) sono fatte di bit
    - ovvero, ciò che vi viene "ricordato" sono bit
    - perché, come stiamo per vedere, praticamente ogni cosa è rappresentabile coi bit
    - normalmente, però, non si dice però che una memoria può memorizzare 16 bit, ma si dice che può memorizzare 2 bytes: ovvero l'unità di misura delle memorie (non solo RAM, ma anche dischi rigidi, CD, DVD etc) è il byte
    - in realtà, le memorie memorizzano molti bytes
    - limitandosi alla RAM e al disco: le dimensioni tipiche sono di miliardi di bytes, e di centinaia di miliardi di bytes, rispettivamente
    - siccome parlare di migliaia di miliardi e cose simili è un po' scomodo, si usano le sigle
    - già nel Sistema Metrico Internazionale ci sono dei prefissi (vedere Figura 3, tratta da Wikipedia), ma sono poco usati (mai sentito parlare di gigametri?)
    - nell'informatica, invece, se ne fa un grand'uso, ma il loro valore è leggermente diverso (vedere Figura 4, ancora tratta da Wikipedia)

Prefissi del Sistema Internazionale

10^n	Prefisso	Simbolo	Nome	Equivalente decimale
10^{24}	yotta	Y	Quadrilione	1 000 000 000 000 000 000 000 000
10^{21}	zetta	Z	Triliardo	1 000 000 000 000 000 000 000
10^{18}	exa	E	Trilione	1 000 000 000 000 000 000
10^{15}	peta	P	Biliardo	1 000 000 000 000 000
10^{12}	tera	T	Bilione	1 000 000 000 000
10^{9}	giga	G	Miliardo	1 000 000 000
10^{6}	mega	M	Milione	1 000 000
10^{3}	kilo	k	Mille	1 000
10^{2}	hecto	h	Cento	100
10^{1}	deca	da	Dieci	10
10^{0}			Uno	1
10^{-1}	deci	d	Decimo	0,1
10^{-2}	centi	c	Centesimo	0,01
10^{-3}	milli	m	Millesimo	0,001
10^{-6}	micro	µ	Milionesimo	0,000 001
10^{-9}	nano	n	Miliardesimo	0,000 000 001
10^{-12}	pico	p	Bilionesimo	0,000 000 000 001
10^{-15}	femto	f	Biliardesimo	0,000 000 000 000 001
10^{-18}	atto	a	Trilionesimo	0,000 000 000 000 000 001
10^{-21}	zepto	z	Triliardesimo	0,000 000 000 000 000 000 001
10^{-24}	yocto	y	Quadrilionesimo	0,000 000 000 000 000 000 000 001

Figure 3: Prefissi del sistema Metrico Internazionale

IEC prefix		Representations				Customary prefix	
Name	Symbol	Base 2	Base 1024	Value	Base 10	Name	Symbol
kibi	Ki	2^{10}	1024^1	1 024	$\approx 1.02 \times 10^3$	kilo	k, K
mebi	Mi	2^{20}	1024^2	1 048 576	$\approx 1.05 \times 10^6$	mega	M
gibi	Gi	2^{30}	1024^3	1 073 741 824	$\approx 1.07 \times 10^9$	giga	G
tebi	Ti	2^{40}	1024^4	1 099 511 627 776	$\approx 1.10 \times 10^{12}$	tera	T
pebi	Pi	2^{50}	1024^5	1 125 899 906 842 624	$\approx 1.13 \times 10^{15}$	peta	P
exbi	Ei	2^{60}	1024^6	1 152 921 504 606 846 976	$\approx 1.15 \times 10^{18}$	exa	E
zebi	Zi	2^{70}	1024^7	1 180 591 620 717 411 303 424	$\approx 1.18 \times 10^{21}$	zetta	Z
yobi	Yi	2^{80}	1024^8	1 208 925 819 614 629 174 706 176	$\approx 1.21 \times 10^{24}$	yotta	Y

Figure 4: Prefissi del sistema Metrico Internazionale nel caso binario

ASCII Code Chart

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

Figure 5: Il codice ASCII

  - 1 KB vale $1024 = 2^{10}$ bytes, 1 MB vale $1048576 = 2^{20}$ bytes, e così via
  - il codice ASCII può essere riassunto come in Figura 5
    * ciascun carattere è rappresentato su un byte, ovvero 8 bit
    * quindi servono 2 cifre esadecimali per rappresentare un carattere
    * nella tabella del codice ASCII data in Figura 5, l'intestazione della riga dà la prima cifra esadecimale, quella della colonna la seconda
    * ad esempio, la "A" maiuscola ha come codice 41_{16}, che corrisponde a 65 in decimale
    * le prime 2 righe (quindi 32 caratteri) sono quelli non (direttamente) stampabili
    * tra gli altri, da notare il carattere **LF** (*line feed*, corrispondente a $0A_{16} = 10_{10}$), che rappresenta l'andata a capo (tasto Return o Invio)
  - ASCII fatto apposta per semplificare alcune operazioni: tutti i caratteri vicini, facile il passaggio alle maiuscole

- Caratteri: standard UNICODE e codifica UTF-8
  - l'ASCII fa il minimo indispensabile (niente lettere accentate, niente lettere greche, ...)
  - nello UNICODE, tutti i caratteri delle lingue conosciute al mondo trovano un posto (anche arabo, cinese, giapponese, cirillico, Braille...)
  - per rappresentare lo UNICODE con stringhe di bit si usa per lo più il codice UTF-8 (usato da quasi tutte le pagine del WWW)

- qui basti sapere che nell'UTF-8 i caratteri "semplici" (quelli dell'ASCII) hanno la stessa codifica che avevano in ASCII
- per gli altri, si usano 2 o più bytes
- esistono però altri metodi
- un file di testo è fatto di soli caratteri, seconda una delle possibili codifiche; se un programma usa un codice mentre un file è stato scritto con l'altro, i caratteri speciali verranno mal visualizzati