



Rappresentazione dell'Informazione

Luca Abeni

February 25, 2014



- Informatica:
 - ◆ **Scienza della rappresentazione e dell'elaborazione dell'informazione**
 - Studio degli algoritmi che descrivono e trasformano l'informazione
 - ◆ Oppure: **Scienza dei Computer** (Computer Science)
- Come funzionano i computer?
- Come si rappresenta l'informazione (in un computer)?
- Come istruire un computer per risolvere un determinato problema?

- Scienza della **rappresentazione dell'informazione**: come rappresenta l'informazione un computer?
 - ◆ Rappresentare numeri
 - ◆ Rappresentare testo
 - ◆ Rappresentare immagini
 - ◆ Rappresentare suoni
 - ◆ Rappresentare *algoritmi*
- Computer: insieme di circuiti elettronici
 - ◆ Tutto è 0 o 1...
 - ◆ ...numeri, testo, immagini, suoni, etc... Tutto va rappresentato come sequenza di 0 e 1!!!
 - ◆ Come operare su tali sequenze???

In Questo Corso...

■ Vedremo:

- ◆ Cos'è un computer
- ◆ Cos'è un algoritmo
- ◆ Cos'è un linguaggio di programmazione (esempio: C)

■ Nel dettaglio:

- ◆ Architettura (struttura di un computer, linguaggio macchina)
- ◆ Codifica dell'elaborazione
- ◆ Struttura di un algoritmo
- ◆ Dall'algoritmo al programma

- Un computer è un insieme di circuiti elettronici...
- ...In ogni circuito, la corrente può **passare** o **non passare**
 - ◆ Passa corrente: 1
 - ◆ Non passa corrente: 0
- ⇒ un computer memorizza (e manipola) solo sequenze di 0 e 1
 - ◆ Sequenze di 0 e 1 devono poter rappresentare tutti i tipi di informazione che un computer gestisce: valori numerici, caratteri o simboli, immagini, suoni, **programmi**, ...
- Una singola cifra (0 o 1) è detta **bit**; sequenze di 8 bit sono chiamate **byte**

- Per fare sì che una sequenza di bit (cifre 0 o 1) possa essere interpretata come informazione utile, deve essere stabilita una **codifica** (rappresentazione digitale)
- Tutto deve essere *codificato* opportunamente per rappresentarlo in questo modo
 - ◆ Numeri (interi con e senza segno, razionali, reali, ...)
 - ◆ Caratteri / testo
 - ◆ Programmi (sequenze di istruzioni macchina)
 - ◆ Immagini e Suoni
- Sì, ma come codifichiamo tutta questa roba???

- Codifica: funzione che associa ad un oggetto / simbolo una sequenza di bit
- Codifica su n bit: associa una sequenza di n bit ad ogni entità da codificare
 - ◆ Permette di codificare 2^n entità / simboli distinti
- Esempio: per codificare i semi delle carte bastano 2 bit
 - ◆ Picche \rightarrow 00
 - ◆ Quadri \rightarrow 01
 - ◆ Fiori \rightarrow 10
 - ◆ Cuori \rightarrow 11

- Rappresentiamo un numero come sequenza di 0 e 1...
- ...Cominciamo con cose semplice: **numeri naturali** (interi positivi) $n \in \mathcal{N}$
- Per capire come fare, consideriamo i numeri “che conosciamo” (base 10)
 - ◆ Se scrivo 501, intendo $5 * 100 + 0 * 10 + 1(*10^0)$
 - ◆ Sistema **posizionale**: il valore di ogni cifra dipende dalla sua posizione
 - ◆ Sistema **decimale** (base 10): ho 10 cifre (0...9)

- In generale, un numero naturale è una sequenza di cifre
- B possibili cifre (da 0 a $B - 1$)
 - ◆ B è detto *base di numerazione*
 - ◆ Siamo abituati a 10 cifre (sistema decimale, o in base 10), ma nulla vieta di usarne di più o di meno...
- Il valore di una sequenza di cifre $c_i c_{i-1} c_{i-2} \dots c_0$ si calcola moltiplicando ogni cifra per una potenza della base B , che dipende dalla posizione della cifra
$$c_i c_{i-1} c_{i-2} \dots c_0 = c_i * B^i + c_{i-1} * B^{i-1} + \dots + c_0 * B^0$$
- Una sequenza di cifre da interpretarsi come numero in base B è spesso indicata con pedice B (501_{10} , 753_8 , 110110_2 , $1AF_{16}$...)

Esempi con Basi di Numerazione Diverse da 10

- Base 2 (cifre: 0 e 1): sistema **binario**

- ◆ Esempio: $1101_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13_{10}$

- Base 8: sistema **ottale**

- ◆ Esempio: $170_8 = 1 * 8^2 + 7 * 8^1 + 0 * 8^0 = 120_{10}$

- Base 16: sistema **esadecimale**

- ◆ Esempio: $1AF_{16} = 1 * 16^2 + 10 * 16^1 + 15 * 16^0 = 431_{10}$

- Le cifre da 10 a 16 sono rappresentate dalle lettere da *A* ad *F* ($A_{16} = 10_{10}$, $B_{16} = 11_{10}$, ... $F_{16} = 15_{10}$)

Hai Detto... Binario?

- Sistema binario: 2 sole cifre (0 e 1)
 - ◆ Sembra fatto apposta per i computer!
- Un numero naturale è rappresentato in binario su k cifre
 - ◆ Cifra binaria: **binary digit** — bit
 - ◆ Numero binario su k cifre: valori fra 0 e $2^k - 1$
- Valori comuni per k : 8, 16, 32, 64
 - ◆ Le potenze di 2 rivestono un ruolo molto importante!
- Ricordi? Bit raggruppati in sequenze di 8, dette **byte**
 - ◆ Ogni byte può assumere 2^8 valori, da 0 (= 00000000₂) a 255 (= 11111111₂)

- La base 2 è la base “nativa” dei computer...
- ...Ma i numeri in base 2 possono richiedere un gran numero di cifre!
 - ◆ Esempio: $1000_{10} = 1111101000_2$
- Spesso si usa la base 16 (numerazione esadecimale) per ridurre il numero di cifre
 - ◆ Ogni cifra esadecimale corrisponde a 4 cifre binarie
 - ◆ La conversione fra binario ed esadecimale (e viceversa) è molto semplice
 - ◆ Esempio: $(0011\ 1110\ 1000)_2 = 3E8_{16}$

Conversioni fra Basi

- Base 2 \leftrightarrow 16: come visto (si convertono 4 bit in una cifra esadecimale e viceversa)
- Base $B \rightarrow 10$: già visto anche questo (si moltiplica ogni cifra c_i per B^i , dipendentemente dalla sua posizione)
- Base 10 $\rightarrow B$: si divide iterativamente il numero per B
 1. Per convertire x_{10} in base B :
 2. Divisione intera fra x e B
 3. Il resto e' la cifra da inserire a sinistra nel numero convertito
 4. Il quoziente viene assegnato ad x
 5. Ritorna al punto 2

Esempio di conversione

$1000_{10} = ?_2$

X	$X/2$	$X\%2$
1000	500	0
500	250	0
250	125	0
125	62	1
62	31	0
31	15	1
15	7	1
7	3	1
3	1	1
1	0	1

$1000_{10} = 1111101000_2$

$1000_{10} = ?_{16}$

X	$X/16$	$X\%16$
1000	62	8
62	3	14 (<i>E</i>)
3	0	3

$1000_{10} = 3E8_{16}$

Scienza della *rappresentazione* e dell'*elaborazione* dell'informazione

- Abbiamo visto come i computer rappresentano l'informazione...
 - ◆ Sequenze di bit (esempio: per i numeri naturali, rappresentazione binaria)
- ...vediamo ora come elaborare tale informazione!
 - ◆ Operazioni su bit o sequenze di bit

Algebra Booleana

■ Tecnicamente, “Algebra di Boole a 2 valori” ...

◆ 2 soli possibili valori: 0 e 1

■ Se x è una variabile, $x = 1 \Leftrightarrow x \neq 0$, $x = 0 \Leftrightarrow x \neq 1$

◆ 3 operazioni: *and* (\wedge), *or* (\vee) e *not* (\neg)

◆ 0: elemento neutro per \vee ; $1 = \neg 0$: elemento neutro per \wedge

■ Operazioni fondamentali:

$$0 \vee 0 = 0$$

$$0 \vee 1 = 1$$

$$1 \vee 0 = 1$$

$$1 \vee 1 = 1$$

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 0$$

$$1 \wedge 0 = 0$$

$$1 \wedge 1 = 1$$

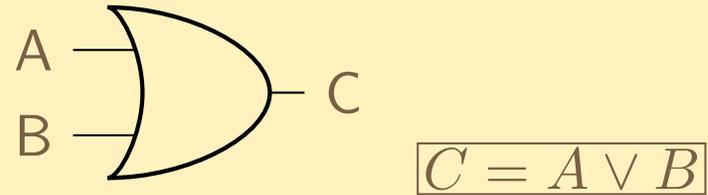
$$\neg 0 = 1$$

$$\neg 1 = 0$$

Algebra Booleana: Assiomi

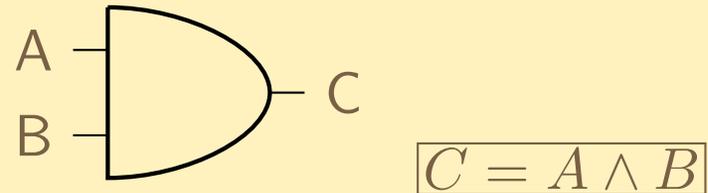
- Proprietà commutativa: $a \wedge b = b \wedge a$; $a \vee b = b \vee a$
- Elementi neutri: $a \wedge 1 = a$; $a \vee 0 = a$
- $a \wedge \neg a = 0$; $a \vee \neg a = 1$
- Proprietà distributiva: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$;
 $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

Or in Dettaglio



- Il risultato è 1 se almeno uno dei due operandi è 1
 - ◆ Per certi versi simile ad una somma
- Detto anche *somma logica*
 - ◆ Talvolta indicato col simbolo +
- Estendibile ad n operandi: la somma logica di n variabili è 1 se il valore di almeno una variabile è 1

And in Dettaglio

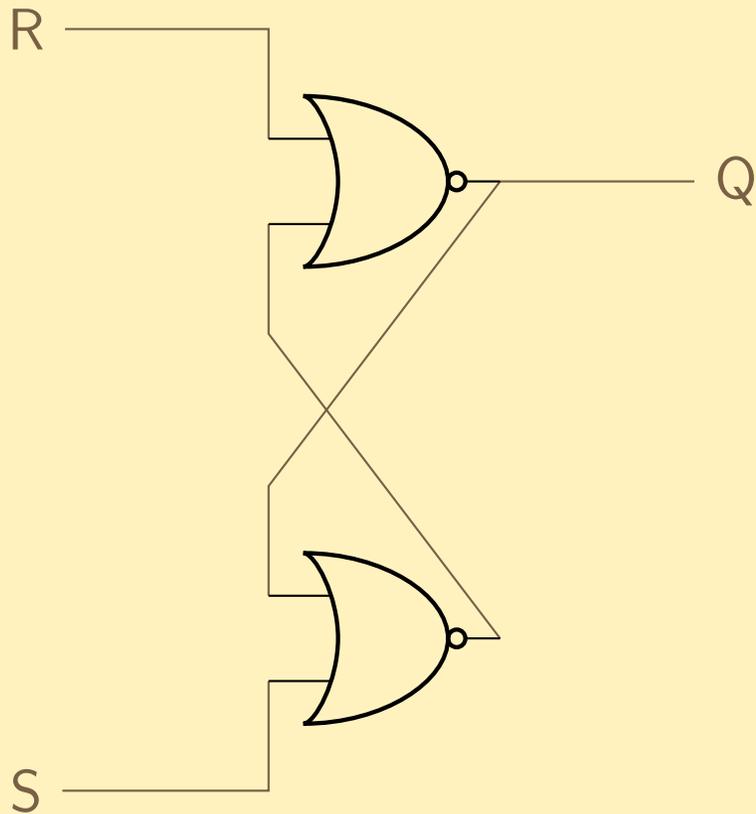


- Il risultato è 1 se tutti e due gli operandi sono 1
 - ◆ Per certi versi simile ad un prodotto
- Detto anche *prodotto logico*
 - ◆ Talvolta indicato col simbolo \cdot
- Estendibile ad n operandi: il prodotto logico di n variabili è 1 se il valore di tutte le variabili è 1

Scienza della *rappresentazione* e dell'*elaborazione* dell'informazione

- Per poter **elaborare** l'informazione, deve essere **memorizzata**!
 - ◆ Dove? Nella memoria principale (RAM: Random Access Memory)
 - Vedi slide “Architettura del Calcolatori”
- ...vediamo ora come può funzionare tale memoria!
 - ◆ Memorizzazione di bit o sequenze di bit
 - ◆ Varie possibilità: per esempio Flip-Flop e memorie dinamiche

Flip Flop SR



- Input S (Set) e R (Reset)
- S e R mai contemporaneamente a 1
- Output: Q — Dipende da se stesso...

Flip Flop SR - Come Funziona

S	R	Q precedente	Q successivo
0	0	0	0
1	0	0	1
0	1	0	0
0	0	1	1
1	0	1	1
0	1	1	0

- $S=1$: La porta NOR in basso ha output **sempre 0**
 - ◆ $Q = \text{not } R = 1$ (perché se $S=1$ allora $R=0$)
- $R=1$: La porta NOR in alto ha output **sempre 0**: $Q=0$
- $S=R=0$: La porta NOR in basso ha output **not Q**
 - ◆ $Q=Q$
- Utile per memorizzare un bit!!!

Memoria Dinamica

- Grosso numero di **condensatori** collegati a circuiti elettronici
- Ogni condensatore può essere in 2 stati:
 - ◆ Condensatore carico: memorizza il bit 1
 - ◆ Condensatore scarico: memorizza il bit 0
- Ogni condensatore tende naturalmente a scaricarsi
 - ◆ Lo stato di carica deve essere periodicamente ripristinato (refresh)
 - ◆ Per questo si parla di memoria *dinamica*
- La memoria si cancella quando il computer viene spento (come per i Flip-Flop)

- Flip-Flop, condensatori o altri elementi di memoria sono raggruppati in gruppi di 8
 - ◆ 8 bit = 1 byte
- Ogni gruppo di 8 elementi (byte) è chiamato cella (o locazione) di memoria
- Ogni locazione di memoria è identificata da un numero, chiamato **Indirizzo di Memoria**
- L'intera memoria è una sequenza di locazioni (celle) accessibili tramite il loro indirizzo
- $1024(= 2^{10})$ byte = 1KB, $1024KB(= 2^{20})$ byte = 1MB, $1024MB(= 2^{30})$ byte = 1GB, etc...

- Come rappresentare il testo tramite sequenze di 0 e 1?
 - ◆ Testo: sequenza di caratteri
 - ◆ Quindi, il problema è rappresentare caratteri come sequenze di 0 e 1...
- Ricorda: n bit possono codificare 2^n simboli diversi
- Quanti possibili caratteri si devono rappresentare?
- Alfabeto “anglosassone”: 7 bit ($2^7 = 128$ diversi caratteri)
 - ◆ Lettere maiuscole e minuscole, numeri, punteggiatura, ...
- ASCII (American Standard Code for Information Interchange)

Lo Standard ASCII

- Specifica come codificare lettere, numeri e punteggiatura su 7 bit
 - ◆ Ma un byte è composto da 8 bit...
 - ◆ Bit più significativo sempre a 0
- Cosa fare per caratteri accentati o “strani”?
 - ◆ Ci sono altre 128 combinazioni di bit disponibili...
- **Extended ASCII**: usa 8 bit per codificare caratteri aggiuntivi
 - ◆ Non esiste un unico standard “esteso” ...
 - ◆ Varie estensioni per supportare vari alfabeti (europa dell’est, ovest, etc...)

Tabella ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

■ Codifichiamo la parola “Ciao”

- ◆ C è codificata come $67 = 0x43 = 01000011$
- ◆ i è codificata come $105 = 0x69 = 01101001$
- ◆ a è codificata come $97 = 0x61 = 01100001$
- ◆ o è codificata come $111 = 0x6F = 01101111$

01000011	01101001	01100001	01101111
C	i	a	o

- Un byte con valore < 128 (bit più significativo a 0) si interpreta in **modo univoco** come carattere
 - ◆ Esempio: 01000001 è **sempre** “A”
- L’interpretazione di byte col bit più significativo ad 1 non è univoca
 - ◆ ISO 8859-1 (Latin1): caratteri dell’Europa Occidentale (lettere accentate, etc...)
 - ◆ ISO 8859-2: caratteri dell’Europa Orientale
 - ◆ ISO 8859-5: per i caratteri cirillici
 - ◆ ...
- Esempio: il valore 224 è “à” per ISO 8859-1, “ř” per ISO 8859-2, etc...

- ASCII esteso: codifica non univoca di caratteri “non standard”
- Problemi nella condivisione di documenti
 - ◆ Se utilizzo una “è” in un documento testo e lo trasmetto ad altre persone...
 - ◆ ...Devo assicurarmi che i computer delle altre persone utilizzino ISO 8859-1 come il mio computer...
 - ◆ ...Altrimenti strani simboli possono essere visualizzati al posto della mia “è”
- E cosa dire degli alfabeti asiatici (cina, giappone, ...)?

- Codifica **univoca** di tutti i possibili caratteri: 8 bit non bastano!!!
- **Unicode**: fino a 2^{32} simboli!!!
 - ◆ Possono servire 32 bit (4 byte) per carattere...
- I simboli unicode possono essere codificati in vari modi
 - ◆ UTF-32: ogni simbolo è composto da 32 bit
 - ◆ UTF-16: ogni simbolo è composto da 16 o più bit (simboli a lunghezza variabile)
 - ◆ UTF-8: ogni simbolo è composto da 8 o più bit
 - Compatibile con ASCII

- Esistono vari tipi di codifica

- ◆ Campionamento

- Monocromatico (bianco e nero) o a colori
- Adatto per immagini di tipo fotografico

- ◆ Codifica vettoriale

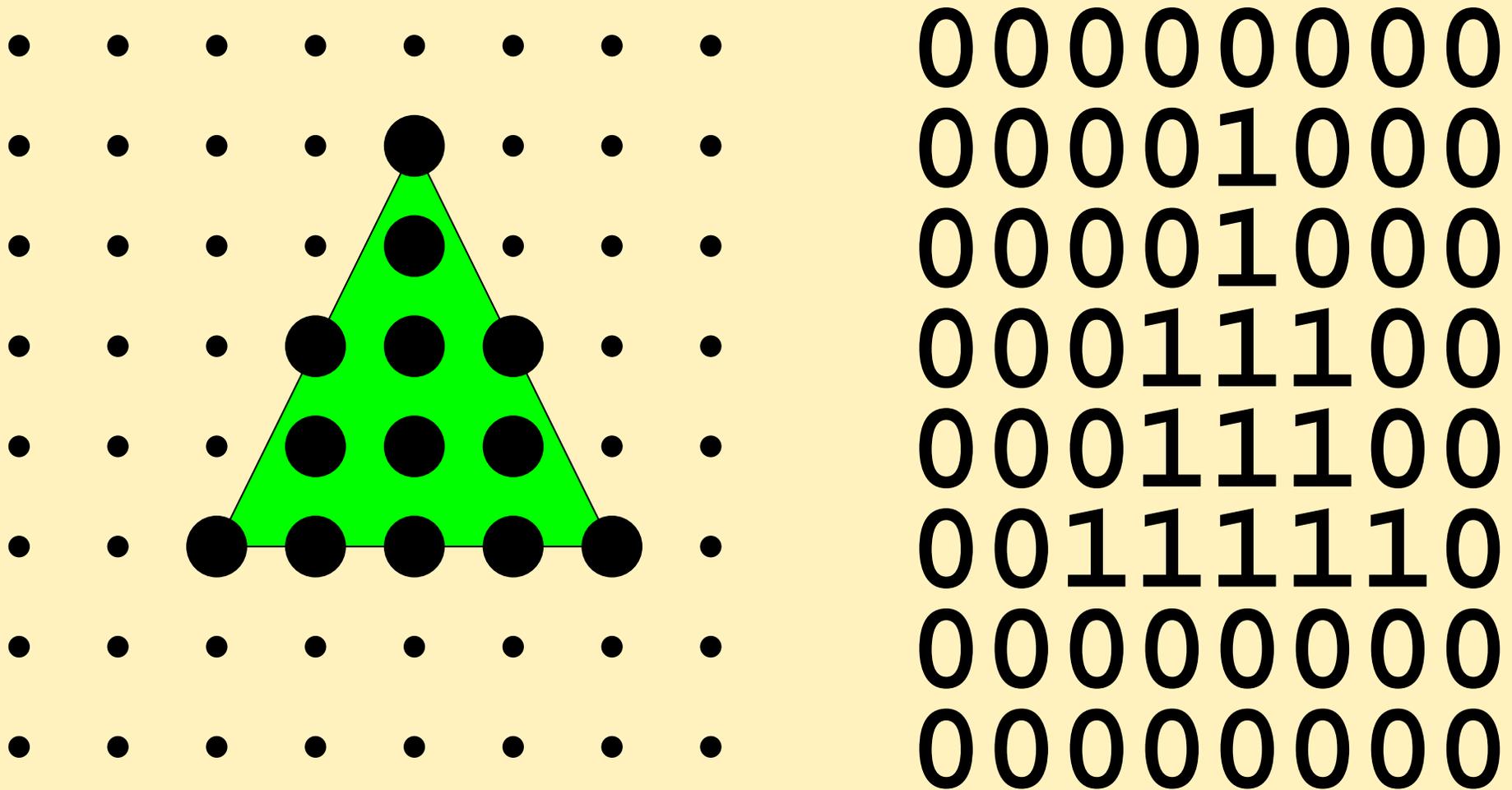
- Rappresenta un'immagine come insieme di primitive geometriche semplici (linee, cerchi, poligoni, ...)
- Adatta per immagini “geometriche”, più facile da riscalarare

Campionamento di un'Immagine

- Immagini quadrate o rettangolari
- Si prelevano campioni (**pixel**) distribuiti uniformemente lungo i due assi (verticale ed orizzontale)
 - ◆ Immagine monocromatica: ogni campione vale 0 (immagine assente) o 1 (immagine presente)
 - ◆ Immagine a colori: ogni campione rappresenta un colore (codificato su n bit)
 - ◆ Numero di bit per campione: **profondità**
 - Esempi di profondità: 1 bit per campione (monocromatico), 8 bit, 24 bit (true color), ...
- Numero di campioni nelle due direzioni: **risoluzione**
 - ◆ Esempi: 800×600 , 1024×768 , 720×576 (tv), ...

Esempio di Campionamento

8 campioni in orizzontale e 8 in verticale, 1 bit per pixel



La codifica è quindi:

00000000000010000000100000011100000111000011111000...

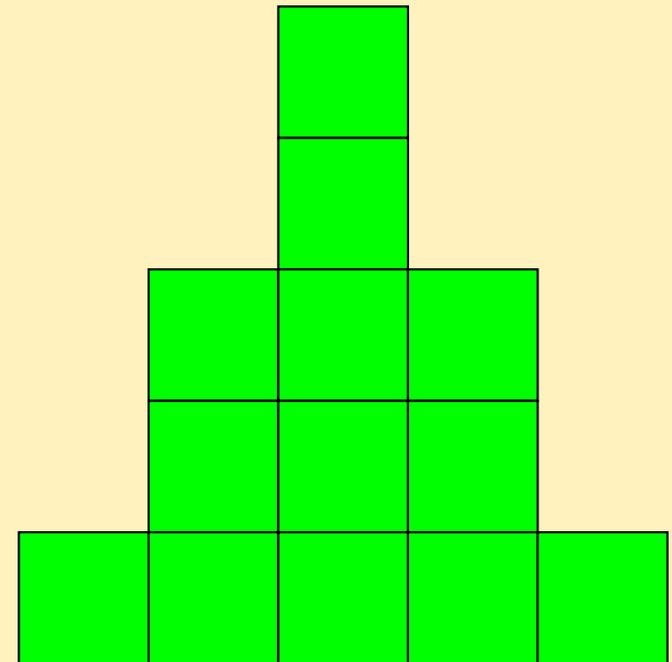
- Dati $n \times m$ campioni, come si ricostruisce l'immagine?
 - ◆ Teorema del campionamento: l'immagine può essere ricostruita con una certa approssimazione
 - ◆ Tecnicamente, si applica un **filtro** alla sequenza dei campioni
 - ◆ In pratica, si usano piccole figure geometriche (quadrati, cerchi, etc...)
- Esempio: in caso di immagine monocromatica, se un campione è 1, si disegna un quadrato al suo posto
 - ◆ Si possono usare anche altri filtri / forme geometriche

Esempio di Decodifica

8 campioni in orizzontale e 8 in verticale, 1 bit per pixel

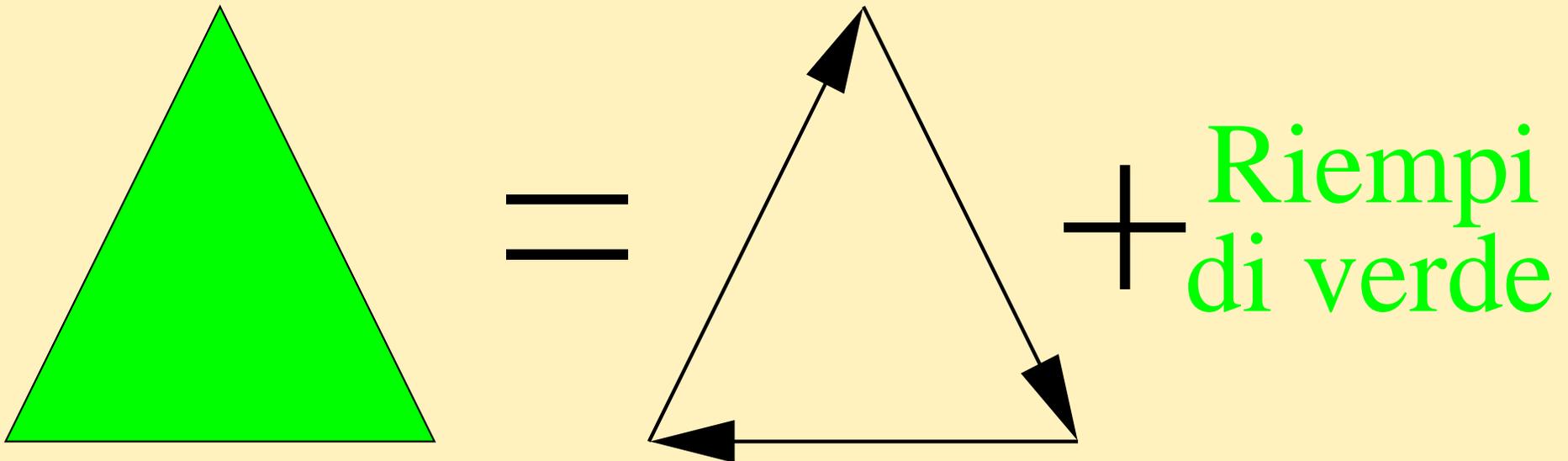
00000000000010000000100000011100000111000011111000...

0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	0
0	0	1	1	1	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Immagini Vettoriali

- Immagine descritta come insieme di linee, curve e figure geometriche
- Esempio:



```
<polygon points="1,6 3,2 5,6 1,6"  
  style="stroke:#000000;stroke-width:8;  
  stroke-linejoin:miter;  
  stroke-linecap:butt;  
  fill:#00ff00;"/>
```

- Sia per immagini campionate che vettoriali...
 - ◆ Come visto, colori codificati su n bit $\rightarrow 2^n$ colori
- Possibili vari tipi di codifica
 - ◆ Toni di grigio
 - ◆ Palette
 - ◆ Basate su componenti
- Toni di grigio: n bit codificano la “luminosità” del pixel
 - ◆ $0 \rightarrow$ nero
 - ◆ $2^n - 1 \rightarrow$ bianco
- Palette: campione = indice in una [tabella dei colori](#)

Componenti dei Colori

- Codifica per componenti: ogni colore è un punto in uno spazio tridimensionale
 - ◆ Spazio vettoriale di dimensione 3
 - ◆ Identificato da 3 valori (ognuno codificato su $n/3$ bit) su una base
- Esempio: **RGB** (Red Green Blue)
 - ◆ Ogni colore è identificato da un'intensità di rosso, un'intensità di verde ed un'intensità di blu
 - ◆ Colori fondamentali
 - ◆ Altre basi ortonormali sono possibili (YUV, ...)
- 8 bit rosso, 8 bit verde, 8 bit blu \rightarrow 24 bit (2^{24} circa 16 milioni di colori): true color!

Dimensione di un'Immagine

- Consideriamo un'immagine campionata...
- Quanta memoria è necessaria per contenerne la codifica?
 - ◆ $w \times h$ campioni; n bit per campione...
 - ◆ Esempio: 352×288 , true color: $352 * 288 * 24 = 2433024$ bit (304128 byte)
 - ◆ TV: $720 * 576 * 24 = 9953280$ bit = 1244160 byte (circa 1.2MB)
- Spesso i dati vengono **compressi** per salvare spazio
 - ◆ Compressione senza perdita (lossless)
 - ◆ Compressione con perdita (lossy)

- Ancora una volta, campionamento o somma/sequenza di “primitive di base” (note...)
- Un suono è rappresentato da una forma d’onda (segnale analogico)
- Campionamento del segnale analogico
 - ◆ Frequenza di campionamento (campioni al secondo)
 - ◆ Numero di bit per campione
- Esempio: qualità CD → 44100 campioni al secondo, 16 bit per campione

- Invece che campionare la forma d'onda, una musica può essere codificata codificando le note (o pause, rumori, etc...) che la compongono
- CD vs spartito musicale
- Esempio: MIDI propone una sorta di “linguaggio” che codifica le varie note (con intensità, altezza, durata, ...) suonate da vari strumenti (specificati tramite un “timbro”)
- Il computer riproduce un brano “suonando” campioni di vari strumenti musicali secondo durate, intensità, altezze specificate dal file MIDI