Codifica dell’informazione

- Il calcolatore memorizza ed elabora vari tipi di informazioni
  - Numeri, testi, immagini, suoni
- Occorre rappresentare tale informazione in formato facilmente manipolabile dall’elaboratore

Rappresentazione delle informazioni

Idea di fondo

- usare presenza/assenza di carica elettrica
- usare passaggio/non passaggio di corrente/luce

Usiamo cioè una rappresentazione binaria (a due valori) dell'informazione

L'unità minimale di rappresentazione è il **BIT** (**BI**nary digi**T** – cifra digitale): **0** o **1**

Informazioni complesse

Con 1 bit rappresentiamo solo 2 diverse informazioni:

si/no - on/off - 0/1

Mettendo insieme più bit possiamo rappresentare più informazioni:

00 / 01 / 10 / 11

Informazioni complesse si memorizzano come sequenze di bit

Informazioni complesse

- Per codificare i nomi delle 4 stagioni bastano 2 bit

- Ad esempio:
  - **0 0** per rappresentare **Inverno**
  - **0 1** per rappresentare **Primavera**
  - **1 0** per rappresentare **Estate**
  - **1 1** per rappresentare **Autunno**

- Quanti bit per codificare i nomi dei giorni della settimana?

Informazioni complesse

In generale, con **N** bit, ognuno dei quali può assumere **2** valori, possiamo rappresentare $\mathbf{2^N}$ informazioni diverse (**tutte le possibili combinazioni di 0 e 1 su N posizioni**)

viceversa

Per rappresentare **M** informazioni dobbiamo usare **N** bit, in modo che $\mathbf{2^N >= M}$

Esempio

Per rappresentare **57** informazioni diverse dobbiamo usare gruppi di almeno **6** bit. Infatti:

$$2^6 = 64 > 57$$

Cioè un gruppo di 6 bit può assumere 64 configurazioni diverse:

000000 / 000001 / 000010 …/ 111110 / 111111

Il Byte

- Una sequenza di **8 bit** viene chiamata ***Byte***
  - 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 1
  - .....

byte = 8 bit = 2^8 = 256 informazioni diverse

Usato come unità di misura per indicare

- le dimensioni della memoria
- la velocità di trasmissione
- la potenza di un elaboratore

Usando sequenze di byte (e quindi di bit) si possono rappresentare caratteri, numeri immagini, suoni.

Altre unità di misura

- KiloByte (**KB**), MegaByte (**MB**), GigaByte (**GB**)
- Per ragioni storiche in informatica Kilo, Mega, e Giga indicano però le ***potenze di 2*** che più si avvicinano alle corrispondenti potenze di 10
- Più precisamente
  - 1 KB = 1024 x 1 byte = $2^{10} \sim 10^3$ byte
  - 1 MB = 1024 x 1 KB = $2^{20} \sim 10^6$ byte
  - 1 GB = 1024 x 1 MB = $2^{30} \sim 10^9$ byte
- I multipli del byte vengono utilizzati come unità di misura per la capacità della memoria di un elaboratore

Il sistema decimale

- 10 cifre di base: 0, 1, 2, ..., 9
- **Notazione posizionale:** la posizione di una cifra in un numero indica il suo **peso** in potenze di **10.** I pesi sono:
  - unità = $10^0 = 1$ (posiz. 0-esima)
  - decine = $10^1 = 10$ (posiz. 1-esima)
  - centinaia = $10^2 = 100$ (posiz. 2-esima)
  - migliaia = $10^3 = 1000$ (posiz. 3-esima)
  - ... .

Esempio di numero rappresentato in notazione decimale

Il **numerale** 2304 in notazione decimale (o in base 10) rappresenta la quantità:

$2304 = 2*10^3 + 3*10^2 + 0*10^1 + 4*10^0 =$

$2000 + 300 + 0 + 4 = 2304$ (**numero**)

Nota: numero e numerale qui coincidono, perché il sistema decimale e quello adottato come sistema di riferimento.

Il sistema binario

- 2 Cifre di base: 0 e 1.
- **Notazione posizionale:** la posizione di una cifra in un numero binario indica il suo **peso** in potenze di **2.** I pesi sono:
  - $2^0 = 1$ (posiz. 0-esima)
  - $2^1 = 2$ (posiz. 1-esima)
  - $2^2 = 4$ (posiz. 2-esima)
  - $2^3=8$; $2^4=16$; $2^5=32$; $2^6=64$; $2^7=128$; $2^8=256$; $2^9=512$; $2^{10} = 1024$; $2^{11}=2048$, $2^{12}=4096$;...

Esempio di numero rappresentato in notazione binaria

Il **numerale** 10100101 in notazione binaria (o in base 2) rappresenta la quantità:

$$10100101$$

$$1*2^7+0*2^6+1*2^5+0*2^4+0*2^3+1*2^2+0*2^1+1*2^0$$

$$128 + 0 + 32 + 0 + 0 + 4 + 0 + 1 =$$

$$165 \text{ (}\mathbf{\text{numero}}\text{)}$$

Il numero più grande rappresentato con N cifre

- Sist. Decimale = 99...99 = 10^N - 1
- Sist. Binario= 11..11 = 2^N - 1
- **Esempio**: 11111111 (8 bit binari) = 2^8 -1 = 255. Per rappresentare il n. 256 ci vuole un bit in più: 100000000 = $1*2^8$ = 256.

Quindi…

Fissate quante cifre (bit) sono usate per rappresentare i numeri, si fissa anche il numero più grande che si può rappresentare:

- con 16 bit: $2^{16} - 1 = 65.535$
- con 32 bit: $2^{32} - 1 = 4.294.967.295$
- con 64 bit: $2^{64} - 1 =$ circa $1{,}84 * 10^{19}$

Conversione da base 2 a base 10

Basta moltiplicare ogni bit per il suo peso e sommare il tutto:

Esempio:

$$10100$$

$$1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 =$$

$$16 + 4 = 20$$

la conversione e' una **somma di potenze**

(N.B. se il numero binario termina per 1 e' dispari altrimenti e' pari).

Conversione da base 10 a base 2

- Dividere il numero per 2 ripetutamente finche` il risultato non e` 0
- Scrivere i resti in ordine inverso.

Esempio: conversione del numero 12

Divisioni: 12/2 = 6/2 = 3/2 = 1/2 = 0

Resti: 0 0 1 1

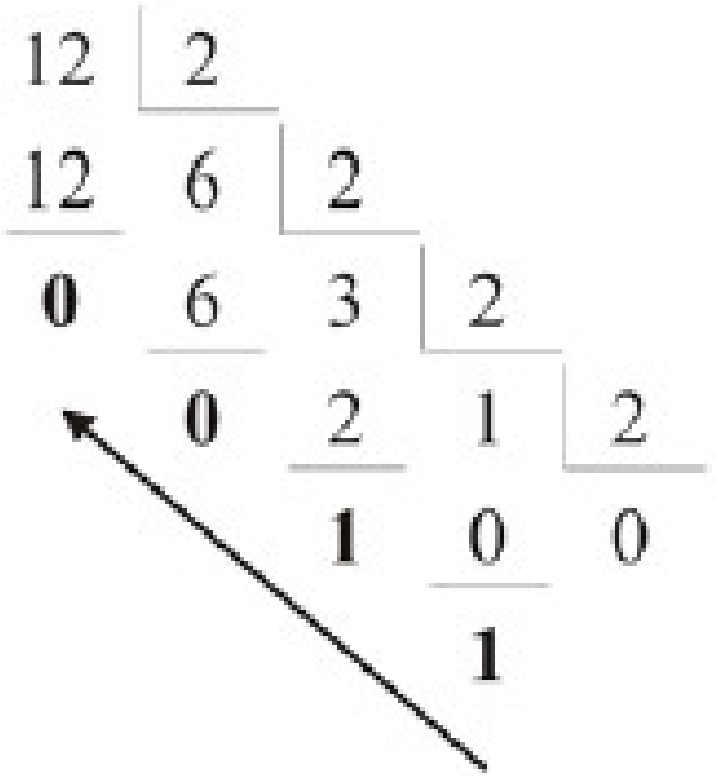
12 = 1100

Conversione da base 10 a base 2

Idea di fondo: usare le potenze di 2 che, sommate, danno il numero **N** da convertire:

- Prendere le potenze di 2 <= di N nell'ordine dalla più grande alla più piccola (cioè 2^0)
- Associare il bit 1 alle potenze che vengono usate nella somma per ricostruire **N**
- Associare il bit 0 alle potenze non usate.

Conversione da base 10 a base 2



Esistono anche altre basi di numerazione

- **CODICE OTTALE**
  - cifre: 0, 1, 2, 3, 4, 5, 6, 7
  - 10 (ottale) = 8 (decimale)

- **CODICE ESADECIMALE**
  - cifre: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - 10 (esadecimale) = 16 (decimale); B = 11;
    $2B=2*16^1+B*16^0 = 32+11 = 43$

Rappresentazione di numeri positivi e negativi

- Il bit più a sinistra rappresenta il segno del numero:

  0 = ‘+’ 1 = ‘-’

  1101 = -5

- E’ indispensabile indicare il numero **N** di bit utilizzati:
  - **1** bit per il segno e **N-1** bit per il modulo
- Con un byte possiamo rappresentare tutti i numeri compresi tra

  +127 (01111111) e –127 (11111111)
  - In generale con **N** bit si rappresentano i valori da

    $-2^{N-1}-1$ a $+2^{N-1}-1$

Rappresentazione di numeri positivi e negativi

Complemento a 2

- Se N sono i bit da utilizzare e x il numero da rappresentare si utilizza il valore binario pari a

$$2^N + x$$

Es. con 4 bit

$+7 = 2^4 + 7 = 16 + 7 = 23 = \underline{\mathbf{1}}0111 = 0111$

$-7 = 2^4 - 7 = 16 - 7 = 9 = 1001$

si scarta

Rappresentazione di numeri positivi e negativi

Complemento a 2

- In alternativa, per i numeri negativi si eseguono i seguenti passi:
  - Si rappresenta in binario il corrispondente numero positivo
  - Si invertono tutti i bit
  - Si aggiunge 1

Es. con 4 bit il numero –7: $+7_2 = 0111 \Rightarrow$

$1000+1=1001 = -7_2$

Rappresentazione di numeri frazionari in Virgola fissa

Un numero frazionario è rappresentato come una coppia di numeri interi: la parte intera e la parte decimale.

12,52 <12; 52>

<1100; 110100>

Numeri in virgola mobile (Floating point)

Idea: $12{,}52 = 1252/100 = 1252 * 10^{-2}$

Un numero decimale è rappresentato come un intero moltiplicato per una opportuna potenza di 10, cioè con una coppia:

<1252; -2>

mantissa esponente

Numeri floating point

E’ necessario stabilire quanti bit assegnare alla mantissa e all’esponente.

Ad esempio, con 16 bit a disposizione possiamo usarne 12 per la mantissa e 4 per l’esponente

(in realtà dovremmo anche tener conto dei segni)

Numeri floating point

Con lo stesso metodo possiamo rappresentare numeri molto grandi. Ad esempio, con 8 bit:

5 bit di mantissa: 11111 = 31

3 bit di esponente: 111 = 7

11111 111 = $31 * 10^7$ = 310 milioni

Mentre, con la notazione classica, con 8 bit rappresentiamo al massimo il n. 255

Numeri floating point

Ma allora, perchè non usare sempre la notazione floating point?

Perchè si perde in precisione

Esempio: 5 cifre (decimali) : 4 per la mantissa, 1 per l'esponente.
Rappresentare

312,45

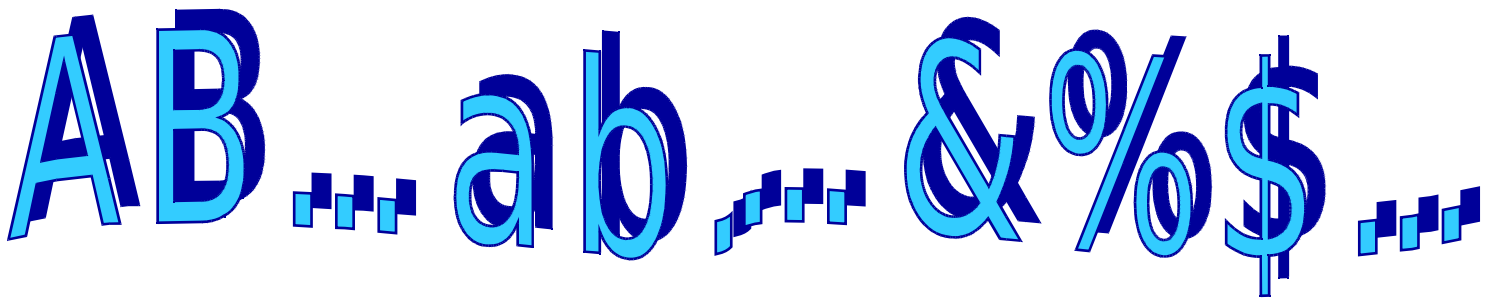
<3124; -1> = [312,4 .. 312,5]???

Numeri floating point

Quindi: possiamo rappresentare numeri molto grandi o con molti decimali al costo di una perdita di precisione

Perchè? Perchè i computer permettono solo rappresentazioni **finite**, e così dobbiamo approssimare alcuni numeri (ad esempio gli irrazionali), ma anche **immagini e suoni**

La Codifica dei Caratteri



Codici per i simboli dell'alfabeto

- Per rappresentare i simboli dell'alfabeto anglosassone (0 1 2 ... A B ... a b ...) bastano 7 bit
  - Nota: *B* e *b* sono simboli diversi
  - 26 maiuscole + 26 minuscole + 10 cifre + 30 segni di interpunzione+… -> circa 120 oggetti

- Per l'alfabeto esteso con simboli quali &, %, $, ... bastano 8 bit come nella codifica accettata universalmente chiamata ASCII esteso
- Per manipolare un numero maggiore di simboli si utilizza la codifica UNICODE a 16 bit

Codifica ASCII

- La codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nterchange **C**ode) utilizza codici su 7 bit
  (2^7 = 128 caratteri diversi)
- Ad esempio
  - 1 0 0 0 0 0 1 rappresenta A
  - 1 0 0 0 0 1 0 rappresenta B
  - 1 0 0 0 0 1 1 rappresenta C

- Le parole si codificano utilizzando sequenze di byte
  - 1000010 1000001 1000010 1000001
    B A B A

Altri codici di codifica

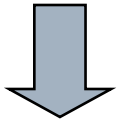
- **ASCII ESTESO**
  - Usa anche il primo bit di ogni byte
  - 256 caratteri diversi
  - non è standard (cambia con la lingua usata)
- **UNICODE**
  - standard proposto a 16 bit (65.536 caratteri)
- **EBCDIC**
  - altro codice a 8 bit della IBM (quasi in disuso)

ASCII esteso

00000000	Null	00100000	Spc	01000000	@	01100000	`
00000001	Start of heading	00100001	!	01000001	A	01100001	a
00000010	Start of text	00100010	"	01000010	B	01100010	b
00000011	End of text	00100011	#	01000011	C	01100011	c
00000100	End of transmit	00100100	$	01000100	D	01100100	d
00000101	Enquiry	00100101	%	01000101	E	01100101	e
00000110	Acknowledge	00100110	&	01000110	F	01100110	f
00000111	Audible bell	00100111	'	01000111	G	01100111	g
00001000	Backspace	00101000	(	01001000	H	01101000	h
00001001	Horizontal tab	00101001	)	01001001	I	01101001	i
00001010	Line feed	00101010	*	01001010	J	01101010	j
00001011	Vertical tab	00101011	+	01001011	K	01101011	k
00001100	Form Feed	00101100	,	01001100	L	01101100	l
00001101	Carriage return	00101101	-	01001101	M	01101101	m
00001110	Shift out	00101110	.	01001110	N	01101110	n
00001111	Shift in	00101111	/	01001111	O	01101111	o
00010000	Data link escape	00110000	0	01010000	P	01110000	p
00010001	Device control 1	00110001	1	01010001	Q	01110001	q
00010010	Device control 2	00110010	2	01010010	R	01110010	r
00010011	Device control 3	00110011	3	01010011	S	01110011	s
00010100	Device control 4	00110100	4	01010100	T	01110100	t
00010101	Neg. acknowledge	00110101	5	01010101	U	01110101	u
00010110	Synchronous idle	00110110	6	01010110	V	01110110	v
00010111	End trans. block	00110111	7	01010111	W	01110111	w
00011000	Cancel	00111000	8	01011000	X	01111000	x
00011001	End of medium	00111001	9	01011001	Y	01111001	y
00011010	Substitution	00111010	:	01011010	Z	01111010	z
00011011	Escape	00111011	;	01011011	[	01111011	{
00011100	File separator	00111100	<	01011100	\	01111100	|
00011101	Group separator	00111101	=	01011101	]	01111101	}
00011110	Record Separator	00111110	>	01011110	^	01111110	~
00011111	Unit separator	00111111	?	01011111	_	01111111	Del

Numeri in ASCII

Le cifre 0..9 rappresentate in Ascii sono simboli o caratteri **NON** quantità numeriche



Non possiamo usarle per indicare quantità e per le operazioni aritmetiche. (Anche nella vita di tutti giorni usiamo i numeri come simboli e non come quantità: i n. telefonici)

Il concetto di FILE

FILE: sequenza di byte conosciuta nel computer con un certo nome.

TIPI DI FILE:

- file di dati (es: immagini o suoni)
- programmi (es: word o explorer)
- file di testo (cioè caratteri ascii)

ESTENSIONI DI UN FILE (windows95/98)

- nome.**gif**, nome.**exe**, nome.**doc**, ...

La struttura dei file

E’ meglio vedere un file come una sequenza di byte. E’ il programma che usa il file a gestirne il contenuto in modo corretto.

Ad esempio, nei file di testo quando viene incontrato il byte = <new line> si devono visualizzare i caratteri successivi nella riga sottostante.

Codifica di immagini

Codifica di immagini

- Un’immagine è un insieme continuo di informazioni
  - A differenza delle cifre e dei caratteri alfanumerici, per le immagini non esiste un'unità minima di riferimento
- Problema: rendere **digitale** una informazione prettamente **analogica**

Codifica di immagini

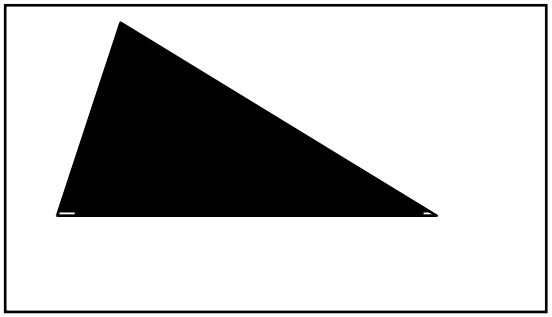
- Esistono numerose tecniche per la memorizzazione digitale e l'elaborazione di un'immagine
  - una prevede la scomposizione dell'immagine in una *griglia* di tanti elementi (**punti**) che sono l'unità *minima* di memorizzazione;
  - La seconda strada prevede la presenza di strutture elementari di natura più complessa, quali *linee*, *circonferenze*, *archi*, etc.

Codifica delle immagini B/N

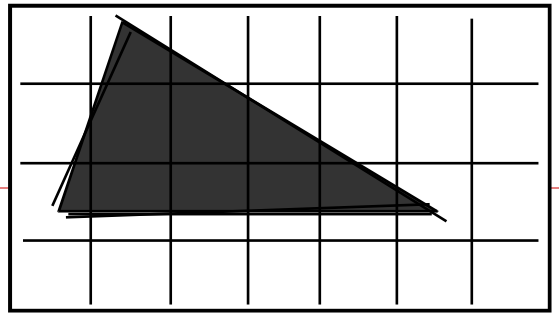
- Dividere l'immagine in una griglia a righe orizzontali e verticali
- Ogni quadratino della griglia è un **pixel** (**picture element**)
- Codificare ogni pixel con:
  - 0 se il pixel è bianco
  - 1 se il pixel è nero
- Convenire un ordinamento per i bit usati nella codifica

Codifica delle immagini B/N

- Consideriamo un'immagine in bianco e nero, senza ombreggiature o livelli di chiaroscuro



- Suddividiamo l’immagine mediante una griglia formata da righe orizzontali e verticali a distanza costante



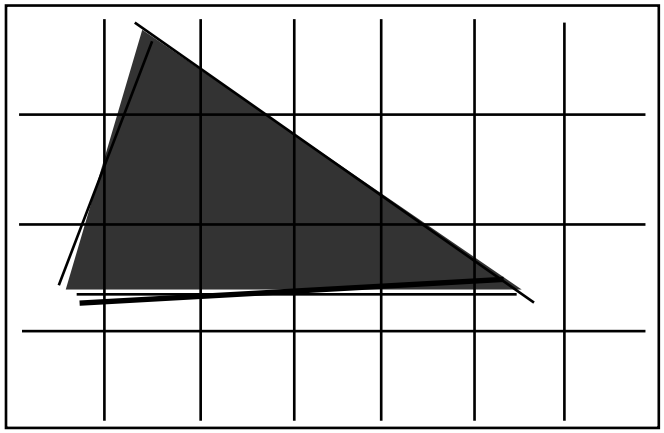
Codifica delle immagini B/N

- Ogni quadratino derivante da tale suddivisione prende il nome di **pixel** (picture element) e può essere codificato in binario secondo la seguente convenzione:
  - il simbolo “**0**” viene utilizzato per la codifica di un pixel corrispondente ad un quadratino bianco (in cui il bianco è predominante)
  - il simbolo “**1**” viene utilizzato per la codifica di un pixel corrispondente ad un quadratino nero (in cui il nero è predominante)

Codifica delle immagini B/N

Poiché una sequenza di bit è lineare, si deve definire una convenzione per **ordinare** i pixel della griglia

Hp: assumiamo che i pixel siano ordinati dal basso verso l'alto e da sinistra verso destra

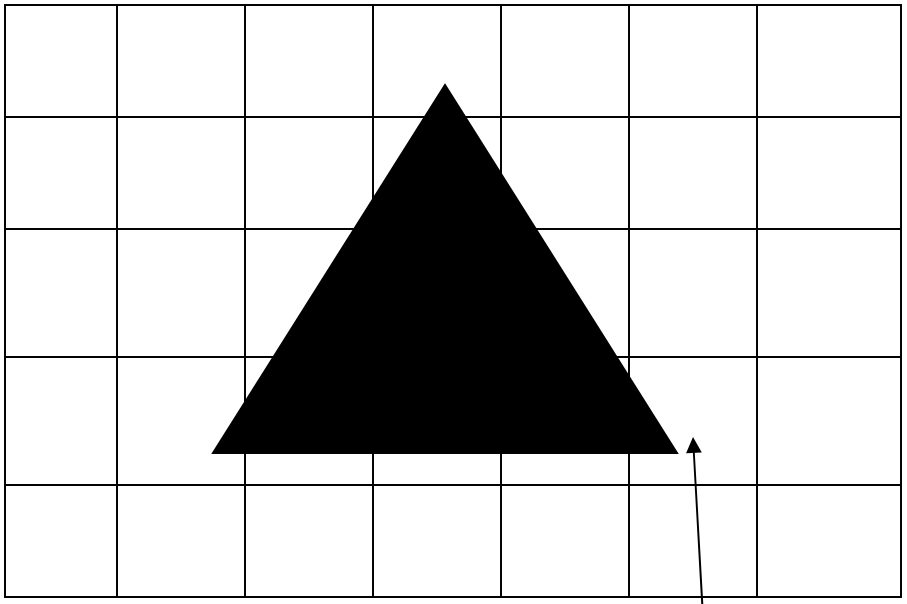


0	1	0	0	0	0	0
22	23	24	25	26	27	28
0	1	1	0	0	0	0
15	16	17	18	19	20	21
0	1	1	1	1	0	0
8	9	10	11	12	13	14
0	0	0	0	0	0	0
1	2	3	4	5	6	7

La rappresentazione della figura è data dalla stringa binaria

0000000 0111100 0110000 0100000

Codifica di un’immagine B/N



Pixel = 1

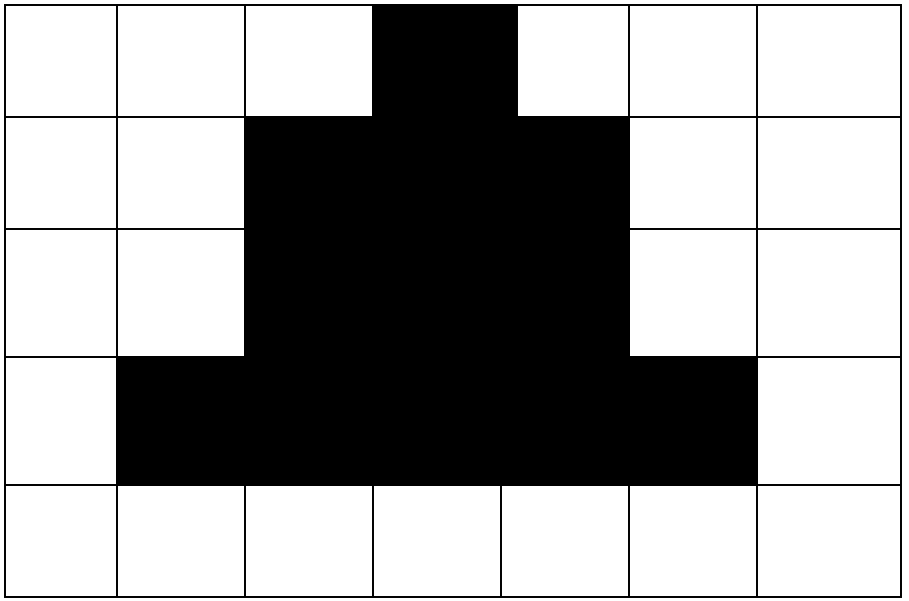
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 0
0 0 0 0 0 0 0

codifica

Decodifica

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

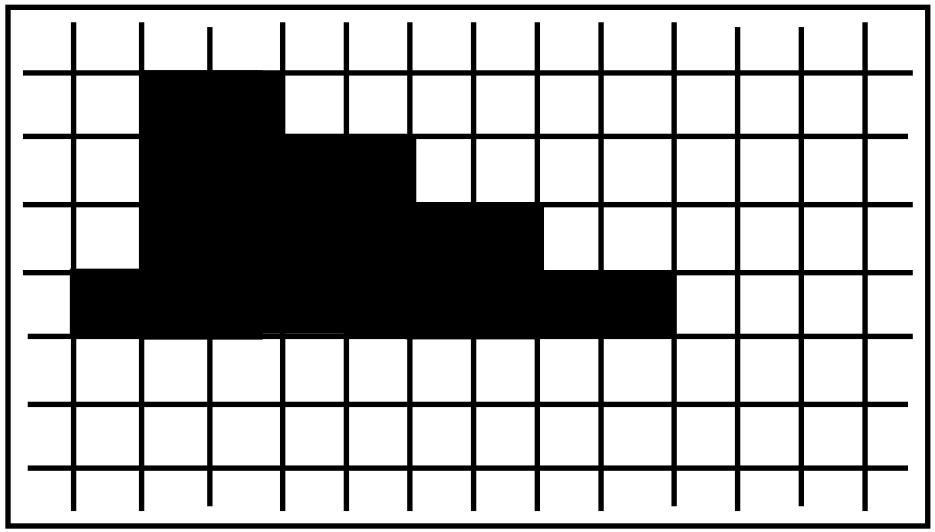
Codifica



Immagine

Codifica delle immagini B/N

- Non sempre il contorno della figura coincide con le linee della griglia
  - nella codifica si ottiene un'approssimazione della figura originaria
- La rappresentazione sarà più fedele all'aumentare del numero di pixel
  - ossia al diminuire delle dimensioni dei quadratini della griglia in cui è suddivisa l'immagine



Codifica delle immagini B/N

Quindi: le immagini sono rappresentate con un certo livello di approssimazione, o meglio, di **risoluzione**, ossia il numero di pixel usati per riprodurre l'immagine.

Risoluzioni tipiche

- 640 x 480 pixel; 800 x 600 pixel
- 1024 x 768 pixel; 1280 x 1024 pixel

Immagini in toni di grigio

- Le immagini in bianco e nero hanno delle sfumature, o **livelli di intensità di grigio**

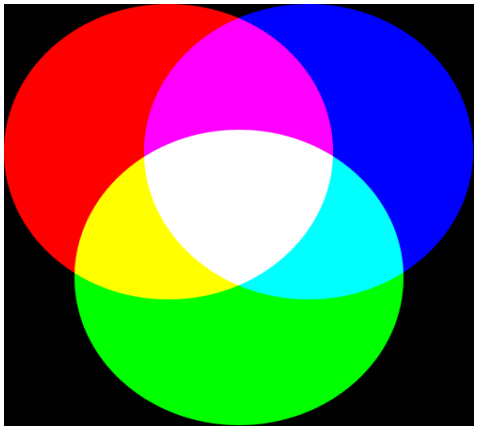
- Per codificare immagini con sfumature:
  - si fissa un insieme di livelli (*toni*) di grigio, cui si assegna convenzionalmente una rappresentazione binaria
  - per ogni pixel si stabilisce il livello medio di grigio e si memorizza la codifica corrispondente a tale livello

- Per memorizzare un pixel non è più sufficiente 1 bit.
  - con **4** bit si possono rappresentare $\mathbf{2^4=16}$ livelli di grigio
  - con **8** bit ne possiamo distinguere $\mathbf{2^8=256}$,
  - con **K** bit ne possiamo distinguere $\mathbf{2^K}$

Immagini a colori

- Analogamente possono essere codificate le immagini a colori:
  - bisogna definire un insieme di sfumature di colore differenti e rappresentarle mediante una opportuna sequenza di bit

- Nella codifica **RGB** si utilizzano tre colori
  - **rosso** (Red), **verde** (Green) e **blu** (Blue)



- Ad ogni colore si associa un certo numero di sfumature codificate su N bit
  (2^N possibili sfumature)

- Esempio
  - con 2 bit per colore si ottengono 4 sfumature per colore
  - con 8 bit per colore si ottengono 256 sfumature per colore e 256^3 (16 milioni) possibili colori

Immagini a colori

- La qualità dell'immagine dipende
  - dal numero di punti in cui viene suddivisa (*risoluzione*)
  - dai toni di colore permessi dalla codifica;

Bitmap (o immagine raster)

- La rappresentazione di un'immagine mediante la codifica a pixel viene chiamata **bitmap**
- Il numero di byte richiesti per memorizzare un bitmap dipende dalla risoluzione e dal numero di colori
- Esempio
  - se la risoluzione è 640x480 con 256 colori occorrono 2.457.600 bit = 300 KB

Bitmap

- I formati bitmap più conosciuti sono
  - **BITMAP** (.bmp),
  - **GIF** (.gif),
  - **JPEG** (.jpg)
  - **TIFF** (.tiff)

  } compressi

- In tali formati si utilizzano metodi di ***compressione*** per ridurre lo spazio di memorizzazione
  - Aree dello stesso colore si rappresentano in modo “abbreviato”.

- E’ in genere possibile passare da un formato ad un altro

Codifica dei filmati

- Immagini in movimento sono memorizzate come sequenze di fotogrammi
  - Si sfrutta la limitatezza della capacità percettiva dell'occhio umano
  - la sequenza continua di immagini viene *discretizzata* ottenendo una serie di immagini (**frame**) che variano velocemente, ma a intervalli stabiliti

- In genere si tratta di sequenze compresse di immagini
  - ad esempio si possono registrare solo le variazioni tra un fotogramma e l’altro

Codifica dei suoni

- Si effettuano dei campionamenti su dati analogici
  - L'onda sonora viene misurata (campionata) ad intervalli regolari
- Si rappresentano i valori campionati con valori digitali
- La frequenza del campionamento determina la fedeltà della riproduzione del suono
  - Minore è l'intervallo di campionamento e maggiore è la qualità del suono

CD musicali: 44000 campionamenti al secondo, 16 bit per campione