

Codifica dell'Informazione

In questa lezione:

- Cosa è l'informazione
- La codifica dei numeri
- La codifica dei caratteri
- La codifica dei suoni
- La codifica delle immagini
- La codifica dei filmati

Cosa è l'informazione

Cosa è l'informazione

Se l'informatica è la gestione automatica dell'informazione, è bene chiarire cosa si intende per informazione e prima ancora per dato.

I **dati** sono insiemi di valori quantitativi o qualitativi relativi a un fatto (o un evento, un fenomeno). Sono rappresentati mediante simboli o combinazioni di simboli. Non sono interpretati.

Es., la stringa (sequenza di caratteri) “**Mario**” il numero **20** (combinazione delle cifre 2 e 0) sono due dati.

L'**informazione** è una relazione tra un insieme di dati, per interpretare e comunicare un fatto.

«**Mario ha 20 anni**» è un'informazione che mette in relazione i dati “Mario” e 20 e ne dà una interpretazione.

Codifica dell'informazione

Le forme in cui si presenta l'informazione sono molto diverse tra loro. Ad esempio:

- un numero
- una frase
- un suono
- un'immagine
- un filmato

Per tutte queste forme possiamo trovare una **codifica**, cioè una rappresentazione tale che l'informazione possa essere elaborata.

Tipologie di Informazione

Potremmo scegliere di rappresentare l'informazione in forma **analogica**, ma si preferisce la rappresentazione **digitale**.

Rappresentazione Analogica: l'informazione viene rappresentata da grandezze continue (es., la traccia su un disco in vinile)

Rappresentazione Digitale: l'informazione viene rappresentata sfruttando combinazioni di simboli (es., le cifre decimali per rappresentare i numeri)

La rappresentazione analogica ha lo svantaggio di essere più soggetta al rumore. Questo limita la trasmissione dell'informazione. Quindi si preferisce una rappresentazione digitale con pochi simboli chiaramente distinguibili.

In informatica si usano due soli simboli, per convenzione **0** e **1**.

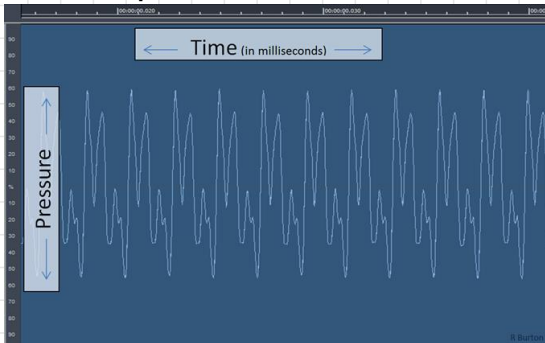
La rappresentazione di un fenomeno

Possiamo rappresentare tutti i fatti o tutti i fenomeni?

Come possiamo rappresentare un fenomeno come il suono?

In fisica, il suono è una vibrazione che si propaga tramite un'onda di pressione attraverso un mezzo (come l'aria o l'acqua).

Possiamo pensare di rappresentare l'onda di pressione come una funzione del tempo:





Possiamo rappresentare l'onda come una sequenza dei valori di pressione presi ad intervalli regolari nel tempo. A questo punto abbiamo una **sequenza di valori numerici**.

È quindi importante poter **codificare i numeri**.

La codifica dei numeri

La rappresentazione dei numeri

Il numero è un concetto astratto. Dato un numero ogni sua rappresentazione viene detta **numerale**. Il numero **5** può essere rappresentato con diversi numerali

- con sassi  o con le dita di una mano 
- in antico romano **V**
- come **5** in decimale
- **101** in binario(!)
- come **€** in greco
- ...

La rappresentazione dei numeri

Un numero è comunemente rappresentato da una sequenza di cifre decimali (cioè i simboli $0, 1, \dots, 9$) dove ogni cifra assume un valore a seconda della posizione in cui si trova nella stringa (notazione posizionale).

Es.:

$$\begin{aligned} 2729 &= 2000 + 700 + 20 + 9 \\ &= 2 \cdot 1000 + 7 \cdot 100 + 2 \cdot 10 + 9 \cdot 1 \\ &= 2 \cdot 10^3 + 7 \cdot 10^2 + 2 \cdot 10^1 + 9 \cdot 10^0 \end{aligned}$$

In generale un numero di n cifre x_i , $i = 0, 1, \dots, n-1$, è rappresentato come

$$\begin{aligned} x_{n-1}x_{n-2}x_{n-3} \dots x_1x_0 &= \\ x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + x_{n-3} \cdot 10^{n-3} + \dots + x_1 \cdot 10^1 + x_0 \cdot 10^0 \end{aligned}$$

dove $0 \leq x_i < 10$. La base di rappresentazione è 10.

La rappresentazione posizionale

Il valore della cifra dipende dalla posizione nella sequenza e dal valore della base.

Fissata la base b :

$$\begin{aligned} 0 \leq x_i < b \\ x_{n-1}x_{n-2} \dots x_0 &= x_{n-1} \cdot b^{n-1} + x_{n-2} \cdot b^{n-2} + \dots + x_0 \cdot b^0 \end{aligned}$$

Inoltre, se $b=10$:

L'insieme delle cifre $A = \{0, 1, \dots, 9\}$ si dice alfabeto

$$b = |A|$$

$$x_i \in A$$

La rappresentazione binaria $b=2$

Se abbiamo a disposizione solo i simboli **0** e **1** allora:

$$A = \{0, 1\}$$

$$b = |A| = 2$$

$$x_{n-1}x_{n-2} \dots x_0 = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \dots + x_0 \cdot 2^0$$

Esempi:

$$(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 0 + 1 = (5)_{10}$$

$$(1010)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 0 + 2 + 0 = (10)_{10}$$

$$(11101)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16 + 8 + 4 + 0 + 1 = (29)_{10}$$

È quindi chiaro come convertire un numero in base 2 in uno in base 10.

Da base 10 a base 2

Se dividiamo un numero per 10, il resto sarà la cifra meno significativa.

Es., $1729/10 = 172$ con resto 9

In generale se $(x_{n-1}x_{n-2} \dots x_0)_b$ è un numero in base b allora

$$x_{n-1}x_{n-2} \dots x_0 / b = x_{n-1}x_{n-2} \dots x_1 \text{ con resto } x_0$$

Se ripetiamo l'operazione sul quoziente $(x_{n-1}x_{n-2} \dots x_1)_b$ otteniamo x_1

Reiterando finché non troviamo un quoziente pari a 0, possiamo ottenere tutte le cifre x_0, x_1, \dots, x_{n-1} .

Esempio di conversione da base 10 a base 2

Convertire in base 2 il numero $(25)_{10}$

1° passo:	25	/	2	=	12	resto	1
2° passo:	12	/	2	=	6	resto	0
3° passo:	6	/	2	=	3	resto	0
4° passo:	3	/	2	=	1	resto	1
5° passo:	1	/	2	=	0	resto	1

Si devono prendere i resti dal basso verso l'alto.

Quindi $(25)_{10}$ è uguale a $(11001)_2$.

Verifica:

$$(11001)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16 + 8 + 0 + 0 + 1 = (25)_{10}$$

Rappresentazione in esadecimale

In informatica si usa una codifica particolare in base 16 detta **esadecimale**.

$$A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

$$b = |A| = 16$$

$$x_{n-1}x_{n-2} \dots x_0 = x_{n-1} \cdot 16^{n-1} + x_{n-2} \cdot 16^{n-2} + \dots + x_0 \cdot 16^0$$

Esempio: il numero $(1729)_{10}$ si scrive $(6C1)_{16}$ in esadecimale.

Infatti:

$$(6C1)_{16} = 6 \cdot 16^2 + 12 \cdot 16^1 + 1 \cdot 16^0 = 6 \cdot 256 + 12 \cdot 16 + 1 = 1536 + 192 + 1 = (1729)_{10}$$

Rappresentazione in esadecimale

L'esadecimale è utile per rappresentare lunghe sequenze di bit. Ad ogni 4 bit si può associare una cifra esadecimale secondo il seguente schema:

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Se dobbiamo rappresentare il contenuto di una cella di memoria di 32 bit, es.:

110000101111001110111101000011011

Possiamo separare i byte:

11000010 | 11110011 | 01111010 | 00011011

e associare due cifre esadecimali a ciascuno di essi:

1100 0010 | 1111 0011 | 0111 1010 | 0001 1011
C 2 | F 3 | 7 A | 1 B

Il contenuto della cella di memoria è quindi

(C2F37A1B)₁₆

Operazioni con numeri in base diversa da 10

Gli algoritmi per calcolare il risultato di operazioni aritmetiche (addizione, sottrazione, moltiplicazione e divisione) funzionano indipendentemente dalla base.

Es.

Addizione

$$\begin{array}{r}
 1 1 1 1 \\
 1 0 1 1 0 + \\
 1 1 1 0 0 1 1 = \\
 \hline
 1 0 1 0 1 0 0 1
 \end{array}$$

Verifica:

$$(110110)_2 = 32 + 16 + 4 + 2 = (54)_{10}$$

$$(1110011)_2 = 64 + 32 + 16 + 2 + 1 = (115)_{10}$$

$$(10101001)_2 = 128 + 32 + 8 + 1 = (169)_{10}$$

Rappresentazione dei numeri relativi

Finora abbiamo rappresentato i numeri naturali. I numeri relativi presentano un segno per distinguere i negativi dai positivi.

Es.: -6 , $+13$.

Potremmo pensare di aggiungere un simbolo s alla rappresentazione di un numero che vale 0 se il numero è positivo e 1 se il numero è negativo. Un numero di n cifre si rappresenterebbe così:

$$s x_{n-1} x_{n-2} \dots x_0$$

$$\begin{array}{lcl} \text{Es.:} & 1110 & = -(4 + 2) = (-6)_{10} \\ & 01101 & = +(8 + 4 + 1) = (13)_{10} \end{array}$$

Così però ci sarebbero due rappresentazioni per lo zero: $+0$ e -0 . Per ovviare a questo inconveniente si utilizzano anche altre rappresentazioni per i numeri relativi.

Rappresentazioni con un numero finito di bit

Premessa

Abbiamo visto che le celle di memoria hanno un numero finito (e piccolo!) di bit.

Quanti numeri naturali possiamo rappresentare con n bit?

Se abbiamo a disposizione 3 bit, i numeri rappresentabili sono 8:

Conf.	Num.	
000	0	In generale con n bit posso ottenere 2^n differenti configurazioni e quindi distinguere tra 2^n oggetti.
001	1	
010	2	
011	3	Con n bit posso rappresentare i naturali da 0 a $2^n - 1$.
100	4	
101	5	Viceversa: dato un numero naturale, di quanti bit ho bisogno per rappresentarlo?
110	6	
111	7	

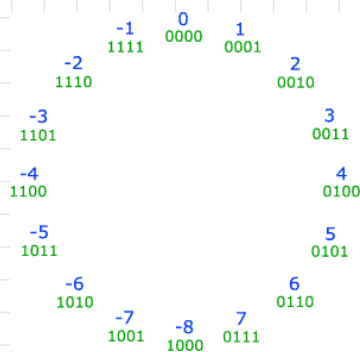
Rappresentazioni con un numero finito di bit

Conseguenza

Molti linguaggi di programmazione, anche di alto livello, definiscono interi solo quei numeri rappresentabili con i bit di una parola.

A ciascuna delle configurazioni possibili associano un intero diverso.

Esempio con 4 bit:



Con un **byte** (8 bit), rappresentiamo gli interi nell'intervallo $-128, 127$, con **16 bit** quelli tra $-32768, 32767$, con **32 bit** quelli tra $-2147483648, 2147483647$ e in generale, con n bit, quelli nell'intervallo aperto $[-2^{n-1}, +2^{n-1})$.

Alcuni linguaggi più recenti, tra cui il Python, rappresentano gli interi con un numero di cifre qualsiasi utilizzando più celle di memoria fino ad esaurimento della stessa.

Rappresentazioni numeri frazionari

Un numero frazionario rappresenta una frazione dell'unità ed è un reale tra 0 e 1.

Si presenta nella forma di uno zero seguito da un punto (una virgola in italiano) e da una sequenza di cifre: $x_{-1}x_{-2} \dots x_{-n}$.

Il valore della cifra dipende dalla posizione nella stringa.

Fissata la base b :

$$0 \leq x_i < b$$

$$0.x_{-1}x_{-2} \dots x_{-n} = x_{-1} \cdot b^{-1} + x_{-2} \cdot b^{-2} + \dots + x_{-n} \cdot b^{-n}$$

Esempi:

$$(0.587)_{10} = 5 \cdot 10^{-1} + 8 \cdot 10^{-2} + 7 \cdot 10^{-3}$$

$$(0.1011)_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{1}{2} + 0 + \frac{1}{8} + \frac{1}{16} = (0.6875)_{10}$$

Conversione di frazionari da base 10 a base 2

Se moltiplichiamo un numero frazionario per 10, la parte intera del risultato sarà la prima cifra dopo la virgola.

Es., $0.729 \cdot 10 = 7.29$ con 7 come parte intera.

In generale se $(0.x_{-1}x_{-2} \dots x_{-n})_b$ è un numero frazionario in base b allora

$$0.x_{-1}x_{-2} \dots x_{-n} \cdot b = x_{-1}.x_{-2} \dots x_{-n} \text{ con parte intera } x_{-1}$$

Se ripetiamo l'operazione sulla parte frazionaria $(0.x_{-2} \dots x_{-n})_b$ otteniamo la seconda cifra dopo la virgola x_{-2} .

Reiterando per n volte possiamo ottenere tutte le cifre $x_{-1}, x_{-2}, \dots, x_{-n}$.

Esempio di conversione di frazionari da base 10 a base 2

Convertire in base 2 il numero $(0.8125)_{10}$

1° passo:	0.8125	·	2	=	1.625	parte intera	1
2° passo:	0.625	·	2	=	1.25	parte intera	1
3° passo:	0.25	·	2	=	0.5	parte intera	0
4° passo:	0.5	·	2	=	1.0	parte intera	1

Si devono prendere le parti intere risultati dall'alto verso il basso.

Quindi $(0.8125)_{10}$ è uguale a $(0.1101)_2$.

Verifica:

$$(0.1101)_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16} = (0.8125)_{10}$$

Rappresentazioni dei numeri reali

I numeri reali rappresentabili in un calcolatore sono in realtà razionali, poiché non è possibile rappresentare un numero infinito di cifre. Quindi un numero reale può essere visto come formato da una **parte intera** e una **frazionaria**.

Esempio:

$$(00101001011.10110)_2 = (331.6875)_{10}$$

Se abbiamo a disposizione 16 bit, possiamo pensare di usarne 11 per la parte intera e 5 per la parte decimale, come nell'esempio.

Quando destiniamo un numero fisso di bit alla parte intera e uno alla parte frazionaria parliamo di rappresentazione in **virgola fissa** (*fixed point*).

Per approssimare meglio i numeri reali si usa un'altra notazione detta a **virgola mobile** (*floating point*).

Rappresentazioni dei numeri reali

La rappresentazione **floating point** usa la notazione esponenziale per i reali. Un numero reale positivo r può essere rappresentato come:





$$r = m \cdot b^n$$

dove m è detto **mantissa** ed è un numero frazionario tra 0 e 1, e b è una base il cui esponente n è detto **caratteristica**.

Esempio:

$$687.5 = 0.6875 \cdot 10^3$$

Questa rappresentazione è stata standardizzata dall'IEEE 754 che prevede 32 bit per rappresentare un reale in base $b=2$ in **singola precisione**:

 un bit  di **segno**, 8 bit  di **caratteristica** e 23 bit  di **mantissa**

Per la **doppia precisione** si usano 64 bit (11 per la **caratteristica** e 52 per la **mantissa**). Lo standard verrà illustrato nel corso di Calcolatori Elettronici.

La codifica dei caratteri

La codifica più diffusa prevede l'uso di soli 7 bit per i caratteri. È la codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange).

ASCII control characters		ASCII printable characters			
00	NULL (Null character)	32	space	64	@
01	SOH (Start of Header)	33	!	65	A
02	STX (Start of Text)	34	"	66	B
03	ETX (End of Text)	35	#	67	C
04	EOT (End of Trans.)	36	\$	68	D
05	ENQ (Enquiry)	37	%	69	E
06	ACK (Acknowledgement)	38	&	70	F
07	BEL (Bell)	39	'	71	G
08	BS (Backspace)	40	(72	H
09	HT (Horizontal Tab)	41)	73	I
10	LF (Line feed)	42	*	74	J
11	VT (Vertical Tab)	43	+	75	K
12	FF (Form feed)	44	,	76	L
13	CR (Carriage return)	45	-	77	M
14	SO (Shift Out)	46	.	78	N
15	SI (Shift In)	47	/	79	O
16	DLE (Data link escape)	48	0	80	P
17	DC1 (Device control 1)	49	1	81	Q
18	DC2 (Device control 2)	50	2	82	R
19	DC3 (Device control 3)	51	3	83	S
20	DC4 (Device control 4)	52	4	84	T
21	NAK (Negative acknowl.)	53	5	85	U
22	SYN (Synchronous idle)	54	6	86	V
23	ETB (End of trans. block)	55	7	87	W
24	CAN (Cancel)	56	8	88	X
25	EM (End of medium)	57	9	89	Y
26	SUB (Substitute)	58	:	90	Z
27	ESC (Escape)	59	;	91	[
28	FS (File separator)	60	<	92	\
29	GS (Group separator)	61	=	93]
30	RS (Record separator)	62	>	94	^
31	US (Unit separator)	63	?	95	_
127	DEL (Delete)				

Il codice ASCII è insufficiente per rappresentare tutti i caratteri utilizzati nelle varie lingue. Recentemente si è diffuso l'**Unicode**, che può codificare oltre un milione di caratteri. Un carattere è indicato con *U+* seguito da 4, 5 o 6 cifre esadecimali.

Es. il **codice Unicode** per "€" è **U+20AC**.

Per evitare l'uso di più byte anche per i caratteri nel codice ASCII, è stata introdotta la codifica Unicode **Utf-8** che prevede un solo byte per i caratteri ASCII.

La codifica dei suoni

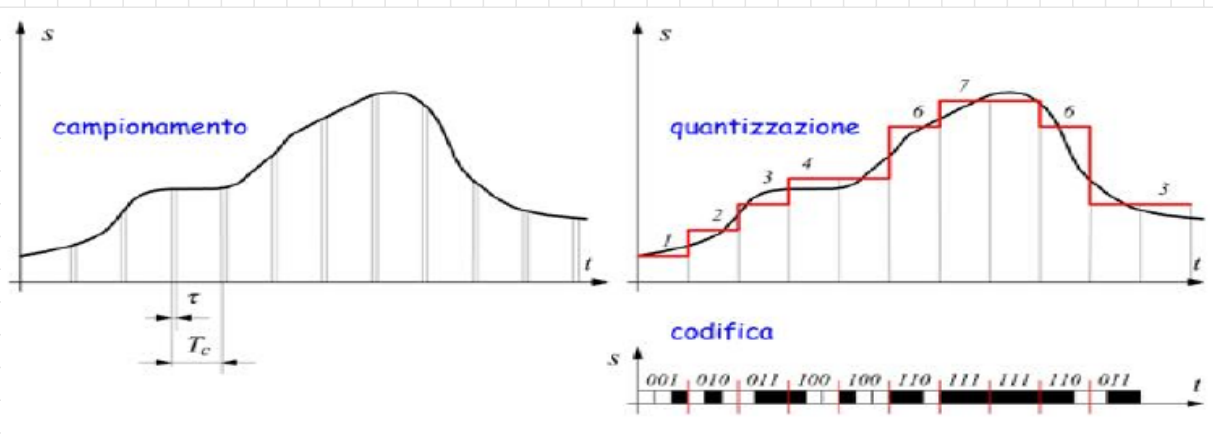
I segnali analogici possono essere visti come funzioni continue nel tempo. È continuo sia l'intervallo di tempo in cui prendiamo in considerazione il segnale sia l'insieme dei valori che può assumere.

Di conseguenza, per identificare l'elemento costitutivo di base si ricorre a due operazioni:

campionamento: consideriamo solo i valori che il segnale assume in determinati istanti di tempo

quantizzazione: approssimiamo l'insieme continuo dei valori con un insieme discreto (cioè, definire quanti valori diversi il segnale può assumere per definire il numero di bit necessario a identificare un campione)

La codifica dei suoni



Per un CD audio si usano:

- 44.100 campioni al secondo
- 2 byte per campione
- 2 canali (stereo)

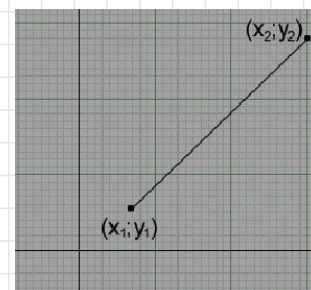
Quindi, per 10 secondi occorrono:

$$44.100 \times 10 \times 2 \times 2 = 1.764.000 \text{ byte} = 1,7 \text{ Mbyte}$$

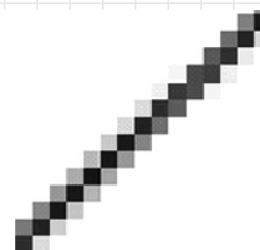
La codifica delle immagini

Esistono due tipi differenti di immagini che differiscono nella natura dell'elemento costitutivo di base:

Immagini vettoriali: descritte matematicamente come composizione di elementi di base come punti e linee

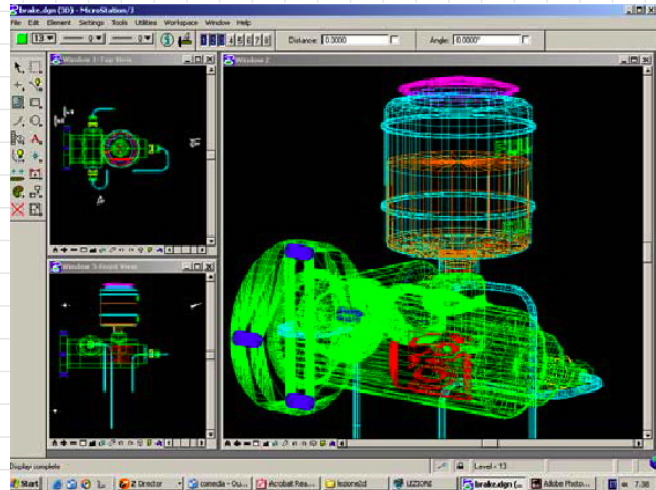


Immagini raster: descritte come griglie di *picture elements* (pixel)



Le immagini vettoriali

Le immagini vettoriali sono utilizzate nella progettazione meccanica, architettónica, elettronica (sistemi di Computer Aided Design, CAD), e dai sistemi di grafica professionale (sistemi di drawing)



Limiti:

- forte dipendenza dal programma applicativo usato per la creazione
- inadeguatezza alla rappresentazione di immagini molto complesse dal punto di vista delle sfumature cromatiche
- inadeguatezza alla rappresentazioni di immagini fotografiche

Le immagini raster

Le immagini raster sono ottenute da scanner o macchine fotografiche digitali.



Originale

Acquisizione

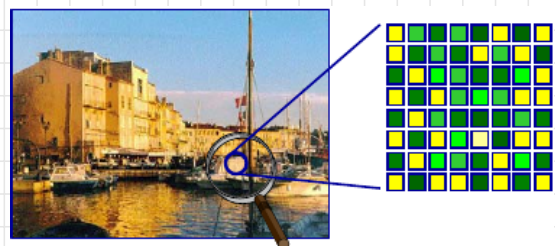


Immagine raster acquisita (matrice di pixel)

Nelle immagini in bianco e nero ad ogni pixel è associato un bit. Nelle immagini a colori ad ogni pixel possono essere associati 3 byte, uno per ogni colore fondamentale (rosso, verde, blu).

Le immagini richiedono Megabyte per essere memorizzate: esistono quindi molti formati per la loro compressione.

La codifica dei filmati

- il **segnale** è **discreto all'origine** (24 fotogrammi al secondo per il cinema, 25 o 30 fotogrammi al secondo per la tv, a seconda dello standard adottato)
- in linea teorica si potrebbero codificare tutti i fotogrammi (immagini statiche) in successione
- questo porterebbe ad una **occupazione di memoria** di 644 Mbyte per un filmato televisivo della durata di un minuto
- sono necessarie tecniche di **compressione**

Esempio: se in una scena lo sfondo rimane invariato da un fotogramma all'altro, per il primo fotogramma si memorizzano tutti i dati, per i fotogrammi successivi solo quelli che differiscono dai precedenti.

Questa tecnica è usata nel **formato MPEG** che lascia non compressi dei fotogrammi chiave (circa ogni mezzo secondo) e comprime quelli intermedi