

Codifica dell'informazione

Moreno Marzolla

moreno.marzolla@pd.infn.it

Copyright © 2006 Moreno Marzolla

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 Italy License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.5/it/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Ricordiamo...

- Il docente
 - Moreno Marzolla
 - Email: moreno.marzolla@pd.infn.it
 - Web: <http://www.dsi.unive.it/~marzolla>
 - Ufficio: Dipartimento di Fisica, via Marzolo 8
- Orario delle lezioni
 - Mercoledì 11:30—13:15, Aula A
 - Giovedì 11:30—13:15, Aula A

Ricordiamo...

- Testi di riferimento
 - Dispense *“Informatica di Base”* (Colussi, Filé, Rossi), presso la libreria Progetto
 - Donatella Sciuto, Giacomo Buonanno, Luca Mari, *“Introduzione ai sistemi informatici”*, terza edizione, McGraw-Hill, 2005, ISBN 88-386-6269-X
- Altri testi consigliati
 - J. Glenn Brookshear, *“Informatica—una panoramica generale”*, Addison-Wesley, 2004, ISBN 88-7192-184-4
 - Stefano Ceri, Dino Mandrioli, Licia Sbattella, *“Informatica: arte e mestiere”*, McGraw-Hill, 2004, ISBN 88-386-6140-5
- Lucidi delle lezioni e ulteriori risorse
<http://www.dsi.unive.it/~marzolla/teaching>

Riepilogo

- Fino ad ora abbiamo visto come è possibile rappresentare in base 2
 - Numeri positivi
 - Interi
 - Numeri reali
- Che dire di altri tipi di informazione?
 - Caratteri alfanumerici
 - Suoni
 - Immagini
 - ...
 - Le istruzioni eseguite dalla CPU

Codifica dei caratteri / 1

- Quanti simboli dobbiamo rappresentare?
- Per l'alfabeto inglese
 - 26 lettere minuscole
 - 26 lettere maiuscole
 - 10 numeri (0—9)
 - circa 30 simboli vari (% , \$, "...)
 - alcuni caratteri di controllo (Return, Canc, Insert...)
 - In totale, circa 120 simboli diversi
- Rappresentiamo ogni simbolo con una sequenza di bit di una certa dimensione fissata
 - Domanda: quanto lunga deve essere ogni sequenza di bit per poter rappresentare univocamente 120 simboli?

Codifica dei caratteri / 2

- Risposta: servono almeno 7 bit
 - Con 7 bit si possono scrivere $2^7 = 128$ combinazioni diverse
 - Da 0000000 a 1111111
- La codifica ASCII (*American Standard Code for Information Interchange*) definisce a quali simboli corrispondono le sequenze di bit
 - 0100001 \Rightarrow !
 - 1000001 \Rightarrow A
 - 1011010 \Rightarrow Z
 - ...e così' via

Codifica dei caratteri / 3

- In realtà in molti alfabeti si fa uso di più di 128 simboli
 - Lettere accentate ed altro
- Per questa ragione si fa uso della variante della codifica ASCII che usa 8 bit (=1 byte) per rappresentare un carattere
 - In tutto si possono rappresentare $2^8 = 256$ simboli diversi
- Però i simboli utilizzati nei vari linguaggi sono molto più che 256!
 - Alfabeti scandinavi, greco, lingue slave, cirillico, ebraico...

Codifica dei caratteri / 4

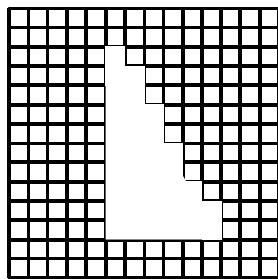
- UNICODE
 - Fa uso di 16 bit (=2 bytes) per codificare i caratteri
 - In tutto si possono codificare $2^{16} = 65536$ simboli diversi
 - I primi 256 simboli in UNICODE sono gli stessi dei caratteri ASCII, per compatibilità

Codifica di immagini / 1

- Le immagini non sono formate da sequenze di oggetti ben definiti come i numeri e le stringhe
- Bisogna quindi prima 'discretizzarle' ovvero trasformarle in un insieme di parti distinte che possono essere codificate separatamente con sequenze di bit
- Consideriamo prima *immagini fisse* (foto etc ...)

Codifica di immagini / 2

- Immagini 'bitmap':
 - l'immagine viene scomposta in una griglia di elementi detti *pixel* (da *picture element*)



Rappresentazione *bitmap*

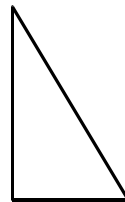
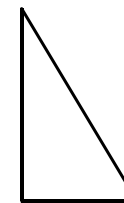
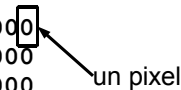


Immagine originale

Codifica di immagini / 3

- Immagini 'bitmap':
 - l'immagine viene scomposta in una griglia di elementi detti *pixel* (da *picture element*)

```
00000000000000000000
00000010000000000000
00000011000000000000
00000011100000000000
00000011110000000000
00000011111000000000
00000011111100000000
00000011111110000000
00000011111111000000
00000000000000000000
```

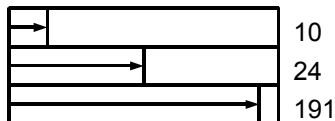


Rappresentazione *bitmap*

Immagine originale

Codifica di immagini / 4

- Come si rappresenta un pixel?
 - Immagini bianco e nero: 1 bit per pixel (0=pixel bianco, 1=pixel nero)
 - Immagini a toni di grigio: un byte per pixel (se il byte vale 0=bianco, se il byte vale 255=nero, gli altri valori rappresentano toni intermedi di grigio)
 - Immagini a colori: 3 bytes per pixel
 - 1 byte per la componente Rossa
 - 1 byte per la componente Verde
 - 1 byte per la componente Blu



Codifica di immagini / 5

- Problema:
 - la rappresentazione accurata di una immagine dipende
 - dal numero di pixel (*definizione, o risoluzione*)
 - dalla codifica del pixel
 - ...e richiede generalmente molta memoria, ad esempio :

<i>tipo</i>	<i>risoluzione</i>	<i>numero colori</i>	<i>num. byte</i>
imm. televisiva	720x625	256	440 KB
SVGA	1024x768	65536	1.5 MB
foto	15000x10000	16milioni	430 MB

Codifica di immagini / 6

- Quindi si cerca di 'risparmiare' memoria :
 - con l'uso di una 'tavolozza' (*palette*) che contiene il sottoinsieme dei colori rappresentabili che compare in una foto
 - ogni pixel codifica un indice all'interno della tavolozza
 - con *tecniche di compressione* che non codificano ogni pixel in modo autonomo ma cercano di raggruppare i le aree che hanno caratteristiche comuni
- Formati più usati:
 - TIFF (tagged image file format),
 - GIF (graphics interchange format),
 - JPEG (Joint photographers expert group)

Codifica di immagini / 7

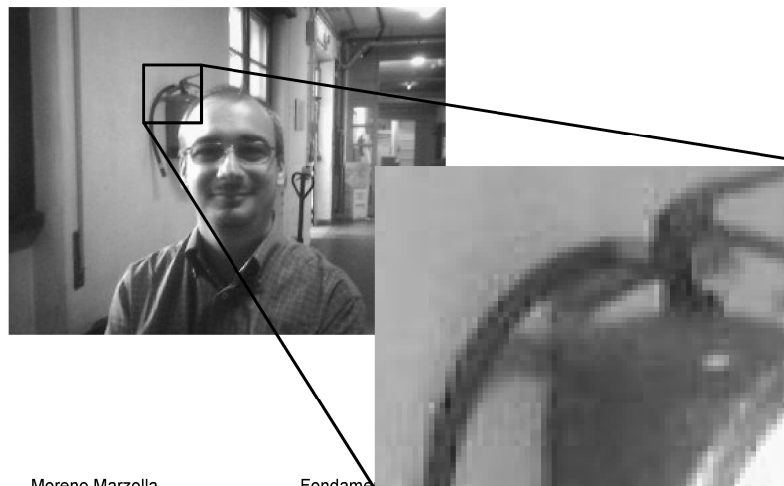
- Algoritmi *lossless* (senza perdita di informazione):
 - Operano un cambiamento di codifica dei dati che permette di diminuire il numero di bit necessari alla rappresentazione
- Esempio:
 - sequenza di 1 milione di caratteri, A=00, B=01, C=10, D=11, totale 2 milioni di bit di codifica
 - se A compare il 90% delle volte posso 'comprimere' la codifica nel seguente modo A=0, B=100, C=110, D=111 ottenendo una lunghezza di :

$$900\,000 * 1 + 100\,000 * 3 = 1\,200\,000 \text{ bit}$$

Codifica di immagini / 8

- Algoritmi *lossy* (con perdita di informazione)
 - generalmente sono specifici di un certo campo e sfruttano le caratteristiche degli oggetti da rappresentare per 'buttare via' informazione poco importanti
 - gli algoritmi di compressione usati nei formati GIF e JPEG per immagini fisse sfruttano la caratteristica dell'occhio umano di *essere poco sensibile a lievi cambiamenti di colore in punti contigui*, e quindi eliminano questi lievi cambiamenti appiattendolo il colore dell'immagine
 - generalmente è possibile specificare quanto siamo disposti a perdere attraverso alcuni parametri

Esempio

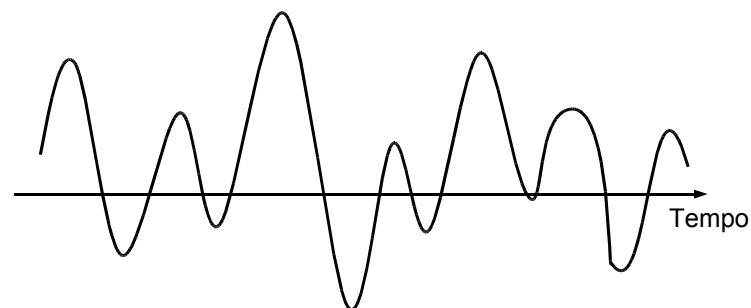


Codifica di immagini / 9

- Immagini in movimento (video ...)
 - il movimento è rappresentato già in modo discreto nei media : con un numero abbastanza alto di fotogrammi fissi (24-30 al secondo) l'occhio umano percepisce il movimento come un continuo
 - potrei in principio codificare separatamente ogni fotogramma come immagine fissa, ma lo spazio di memoria richiesto sarebbe enorme (650 MB, un intero CD per un minuto di proiezione ...)
 - sono stati quindi sviluppati metodi di codifica che economizzano, codificando solo le 'differenze' fra un fotogramma e l'altro (MPEG)

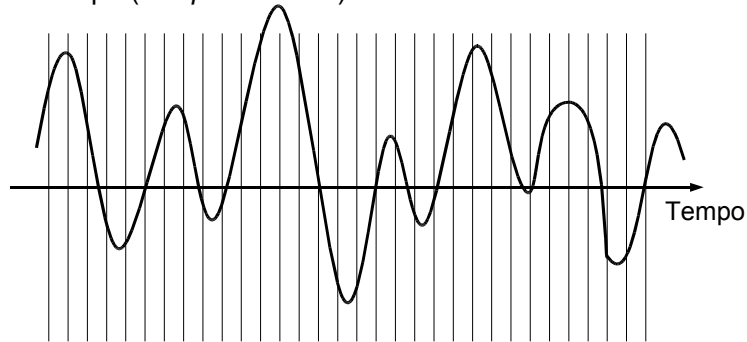
Codifica di suoni / 1

- Un generico suono (o segnale analogico) è rappresentato da un'onda continua



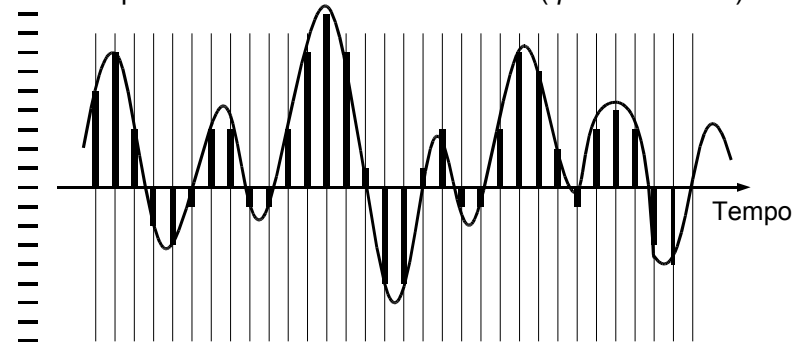
Codifica dei suoni / 2

- Primo passo
 - Il segnale viene misurato solo ad intervalli discreti di tempo (*campionamento*)



Codifica dei suoni / 3

- Secondo passo
 - Per ogni campione, il valore assunto dal segnale viene espresso con un numero finito di bit (*quantizzazione*)



Codifica dei suoni / 4

- L'accuratezza della ricostruzione dipende:
 - da quanto sono piccoli gli intervalli di campionamento
 - da quanti bit vengono utilizzati per descrivere il suono in ogni campione nella fase di quantizzazione
 - al solito *maggiore accuratezza significa maggior quantità di memoria occupata!*
- Anche per i suoni si possono utilizzare tecniche di compressione per migliorare l'occupazione di memoria della sequenza di campioni

Codifica dei suoni / 5

- Algoritmi lossy per suoni
 - Sfruttano il fatto che per l'orecchio umano suoni a basso volume sovrapposti ad altri di volume maggiore sono poco udibili e possono essere eliminati
 - E' quello che accade nello standard MPEG Layer 3, detto anche MP3