

Rappresentazione dell'informazione

- rappresentazione binaria: si basa su due simboli (es. 0/1)
- tecnologicamente pratica (è facile rappresentare due stati: acceso/spento, due polarizzazioni magnetiche, etc)
- i computer attuali si basano unicamente su informazione rappresentata in modo binario a vari livelli (processori, memorie, periferiche....)

Notazione posizionale

- Cosa significa un numero come lo scriviamo noi?
- abbiamo una serie di dieci simboli ordinati: "0", "1", ..., "9"
- il loro significato dipende dalla posizione che assumono nella "parola" che codifica un numero:
- $s_{n-1}s_{n-2}...s_1s_0 = s_{n-1} \cdot 10^{n-1} + s_{n-2} \cdot 10^{n-2} + ... + s_1 \cdot 10^1 + s_0 \cdot 10^0$
- Es: $1967 = 1 \cdot 10^3 + 9 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0$
- "10" è la **base b** della rappresentazione, ovvero il numero di simboli usati; potrebbe essere diversa:
- $s_{n-1}s_{n-2}...s_1s_0 = s_{n-1} \cdot b^{n-1} + s_{n-2} \cdot b^{n-2} + ... + s_1 \cdot b^1 + s_0 \cdot b^0$

Esempio: rappresentazione ottale

- base 8, simboli 0,1,2,3,4,5,6,7
- $1367_8 = 1 \cdot 8^3 + 3 \cdot 8^2 + 6 \cdot 8^1 + 7 \cdot 8^0 = 512 + 192 + 48 + 7 = 759_{10}$

Esempio: rappresentazione binaria

- base 2, simboli 0,1
- $1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1 = 11_{10}$

Conversioni tra basi

- da base b a decimale: già vista,

$$N_{10} (= s_{n-1}s_{n-2}\dots s_1s_0)_b = s_{n-1} \cdot 10^{n-1} + s_{n-2} \cdot 10^{n-2} + \dots + s_1 \cdot 10^1 + s_0 \cdot 10^0$$
- da decimale a base b ?

$$N_b (= s_{n-1}s_{n-2}\dots s_1s_0)_{10} = s_{n-1} \cdot b^{n-1} + s_{n-2} \cdot b^{n-2} + \dots + s_1 \cdot b^1 + s_0 \cdot b^0 =$$

$$= b \cdot (s_{n-1} \cdot b^{n-2} + s_{n-2} \cdot b^{n-3} + \dots + s_1 \cdot b^0) + s_0 = b \cdot (s_{n-1} \cdot b^{n-2} + s_{n-2} \cdot b^{n-3} + \dots + s_1) + s_0$$

$$\rightarrow N_{10}/b = (s_{n-1} \cdot b^{n-2} + s_{n-2} \cdot b^{n-3} + \dots + s_1 \cdot b^0) \text{ con il resto di } s_0;$$
- iterativamente troviamo tutti i simboli

Aritmetica in base b

- è quella usuale (che si basa sulla posizione)
- Esempi:
 - somma ottale ($b=8$):
 $2+3=5$; $4+7 = 13$; $17+2 = 21$; $144+43=207$
 - somma binaria ($b=2$):
 $0+1=1$; $1+0=1$; $1+1=10$;
 $00101100+00011001= 01000101$
 - somma esadecimale ($b=16$):
 $FF1+2F=1020$
 - prodotto: bisogna conoscere le tabelline...

Quanta informazione codifico?

- L'entità minima di informazione è la singola cifra della rappresentazione in base b ;
- ogni cifra può assumere b valori diversi;
- quindi:
 - con una cifra si rappresentano b informazioni diverse;
 - con due cifre $b \cdot b = b^2$ informazioni;
 - con N cifre b^N informazioni diverse
- Esempi:
 - 4 bit (\rightarrow base 2) : 2^4 informazioni diverse
 - 1 byte: $2^8 = 256$ informazioni diverse
 - 8 caselle sulla lavagna di lavoro: 10^8 informazioni

Quante cifre mi servono?

- problema inverso: devo rappresentare M diverse informazioni, quante cifre mi servono in una determinata rappresentazione?
- in generale, visto che N cifre rappresentano b^N informazioni, dovrò scegliere N tale che: $b^N \geq M$ (se è $>$, non si usano tutte le configurazioni possibili)
- Esempi (devo rappresentare 57 informazioni diverse):
 - in base 2: $2^6=64 > 57$ ($2^5=32 < 57$) -> 6 cifre
 - in base 5: $5^3=125 > 57$ ($5^2=25 < 57$) -> 3 cifre
 - in base 10: $10^2=100 > 57$ -> 2 cifre

I numeri naturali

- la rappresentazione binaria dei numeri permette di rappresentare numeri naturali (cioè interi positivi, incluso lo zero)
- nei calcolatori però non abbiamo a disposizione un numero illimitato di bit:
 - di solito il numero di cifre (bit) per rappresentare un numero è fissato (8, 16, 32);
 - avendo N bit,
 - si possono rappresentare numeri da 0 a 2^N-1
 - se durante un'operazione aritmetica si va oltre 2^N-1 , si genera il cosiddetto errore di **overflow**
 - Esempio: 4 bit, $1011+0111=10010$ (overflow)

I numeri "negativi"

- sono indispensabili nell'aritmetica usuale
 - come possiamo codificarli?
 - Idea:
 - N bit rappresentano 2^N numeri diversi;
 - per i numeri naturali, avevamo "deciso" che la numerazione partiva da 0;
 - scegliamo invece un intervallo a cavallo dello zero, che includa quindi numeri positivi e negativi
- $-2^{N-1} \quad 0 \quad 2^{N-1}-1 \quad 2^{N-1}$

 naturali
 negativi
- idealmente la codifica dovrebbe mantenere la possibilità di fare le operazioni come al solito...

Per esempio... una codifica

- dati N bit per rappresentare un numero,
 - il primo bit indica il segno (0=positivo, 1=negativo),
 - gli altri N-1 il valore assoluto del numero
 - Esempio a 4 bit:

0000 = +0	1000 = -0
0001 = +1	1001 = -1
0010 = +2	1010 = -2
.....	
0111 = +7	1111 = -7
- Due problemi:
 - due rappresentazioni dello zero (0000, 1000)
 - non si possono fare le operazioni aritmetiche con le regole usuali
- Non ci piace...

Una codifica che funziona: il complemento a due

- il primo bit è il segno (0=positivo, 1=negativo);
- numeri positivi:
 - si codifica il valore con gli altri N-1 bit
- numeri negativi:
 - si rappresenta in complemento a 2 il corrispondente numero positivo;
 - si invertono tutti i bit;
 - si aggiunge 1
- Esempi (N=4 bit):
 - +5 = 0101
 - -5 : +5=0101 -> 1010 -> 1010+0001=1011
- le operazioni aritmetiche usuali sono rispettate

I numeri "con la virgola"

- **NB: non potremo mai rappresentare numeri con infinite cifre all'interno di un calcolatore!**
- è comodo ricondurre un numero con la virgola alla combinazione di due numeri interi.
- Un numero decimale generico può essere scritto come numero intero diviso per un'opportuna potenza di 10, cioè moltiplicato per una potenza negativa di 10:
 - $0.0190 = 19/1000 = 19 \cdot 10^{-3}$
 - $565.2300 = 56523/100 = 56523 \cdot 10^{-2}$
 - $N = \text{mantissa} \cdot 10^{\text{esponente}}$
- dando per scontata la base 10, la coppia di numeri interi è **<mantissa, esponente>**

Rappresentazione "floating point" (o "virgola mobile")

- si basa su quanto appena visto: un numero viene rappresentato nella **notazione esponenziale** come coppia <mantissa, esponente>
- ciò permette anche di rappresentare in modo approssimato numeri più grandi di quanto sarebbe permesso dal numero di bit disponibili:
- esempio: 5 cifre (decimali) a disposizione,
 - 312545 ha 6 cifre -> overflow
 - $312545 = 3125.45 * 10^2 = <3125,2>$
- si perde in precisione, ma possiamo codificare numeri "grandi"

floating point nel calcolatore

- nella pratica, sia mantissa che esponente vengono rappresentati con un numero fissato di bit, a cui si aggiungono i 2 bit di segno (non si usa il complemento a due per comodità).
- Avendo in tutto 16 bit, se ne useranno: 2 per i segni, 9 per il valore assoluto del numero, 5 per il valore assoluto dell'esponente
- quindi i numeri rappresentabili saranno: $-511 * 10^{32} \leq N \leq 511 * 10^{32}$
- Avendo 32 bit, se ne useranno: 2 per i segni, 20 per il valore assoluto del numero, 10 per il valore assoluto dell'esponente
- si può arrivare a 64 o 128 bit

Rappresentazione del testo

- Vogliamo rappresentare tutto ciò che può entrare in un testo:
 - lettere dell'alfabeto maiuscole e minuscole,
 - segni di punteggiatura,
 - cifre,
 - indicatori di fine della riga o pagina, etc.
- Idea: un numero per ogni carattere
- In modi diversi: es ASCII, EBCDIC, ...
 - Il più diffuso è **ASCII** (American Standard Code for Information Interchange), che usa 7 bit per codificare i caratteri (inclusi in un byte con il primo bit a 0)
 - **ASCII esteso**: codifica anche simboli speciali (es *èâü*), con il primo bit a 1; non è realmente standard
 - **UNICODE**: 16 bit, 65536 caratteri, tutti gli alfabeti

Rappresentazione dell'informazione multimediale

- computer più potenti, prezzi ridotti
-> possiamo trattare informazione più complessa (multimediale)
- come?
 - rappresentazione digitale delle informazioni:
 - immagini fisse
 - suoni
 - video
 - problemi di ingombro della rappresentazione
-> tecniche di compressione

Codifica *bitmap* delle immagini

- un'immagine viene suddivisa tramite una griglia in quadratini detti **pixel** (picture element)
- ogni pixel assumerà come valore il colore medio dell'area che rappresenta (secondo un qualche modello del colore)
- la griglia sarà ordinata dal basso verso l'alto e da sinistra verso destra, e corrisponderà ad una matrice costituita dai valori dei pixel
- l'insieme dei valori dei pixel è una approssimazione dell'immagine
- la precisione della codifica dipende dal numero di pixel nella griglia e dal numero di valori che il pixel può assumere (b/n, toni di grigio, colore)

Un esempio

- codifica di immagini in b/n senza sfumature
- ogni pixel è rappresentato con 1 bit:
 - 0 = bianco ;
 - 1 = nero
- griglia composta da 7x4 pixel

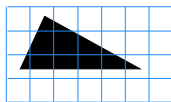


immagine e griglia

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

sua rappresentazione in pixel

0000000011110001100000100000
rappresentazione numerica

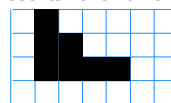
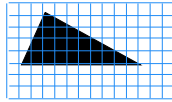


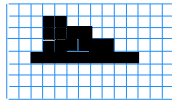
immagine digitale (approssimata)

Precisione della codifica: risoluzione spaziale

- Maggiore è il numero di pixel che compongono la griglia, migliore è l'approssimazione dell'immagine (e l'ingombro)



griglia 14x8



Precisione della codifica: livelli di grigio

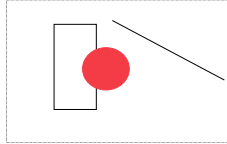
- con 1 bit per pixel possiamo codificare solo bianco e nero;
- normalmente le immagini presentano una serie continua di sfumature dal bianco al nero;
- se associamo più bit ad ogni pixel, possiamo codificare il corrispondente numero di sfumature (livelli) di grigio; per esempio:
 - 4 bit -> 16 livelli di grigio
 - 8 bit -> 256 livelli di grigio
- anche questa è un'approssimazione, in quanto discretizziamo un valore continuo

Precisione della codifica: immagini a colori

- qualsiasi colore può essere ottenuto da almeno tre colori di base, secondo un modello del colore:
 - RGB: rosso, verde e blu;
 - CMYK: ciano, magenta, giallo, nero
 - il colore è rappresentato dai valori delle componenti
- codifiche:
 - a palette: si codificano un certo numero di sfumature di colore con valori dei pixel (es 256 colori diversi) le cui componenti sono descritte in una tabella (palette); serviranno 8 bit per pixel + la palette (es. $8 \times 3 \times 256$)
 - si codificano i parametri relativi ai colori secondo un determinato modello del colore (es. RGB) per ogni pixel (es 8×3 per il numero di pixel nell'immagine)

Codifica vettoriale

- a volte le immagini -sintetiche- possono essere memorizzate come insieme di primitive geometriche che le compongono
- primitive: punto, linea, rettangolo, cerchio, etc codificate con codice + parametri
- es. WMF, PICT,



Esempio di codifica:

- punto: 0, x1, x2
- linea: 1, x1, y1, x2, y2
- rettangolo: 2, x1, y2, x2, y2
- cerchio: 3, c1, c2, r (, <255, 0, 0>)

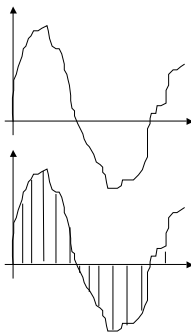
Codifica dell'immagine:

2 30 15 50 75 3 55 45 7 1 5510 110 50

eventuale colore

Codifica del suono

- fisicamente i suoni sono dati da variazioni nella pressione dell'aria che possono essere rilevate (dall'orecchio, dal microfono) e trasformate in impulsi elettrici
- questi impulsi possono venire misurati a intervalli determinati e trasformati in un'approssimazione digitale come succede nelle immagini



Suoni: precisione della codifica

- come nelle immagini, dipende da due fattori:
 - frequenza di campionamento, ovvero l'intervallo di tempo tra una misurazione e l'altra;
 - rappresentazione binaria del campione: anche il valore misurato viene trasformato in un valore discreto
- concretamente, se il suono deve essere ascoltato, possiamo basarci sui limiti del nostro udito: 8-20000 Hz, percezione di differenze fino ad una certa soglia
- 44 KHz come frequenza di campionamento
 - 16 bit/campione (=65536 valori diversi)

Codifiche video

- un'immagine in movimento viene memorizzata (non solo nei calcolatori...) come sequenza di fotogrammi, catturati a intervalli fissati (es. 25/s)
- la codifica digitale è simile: una sequenza video corrisponde ad una sequenza di immagini
- il problema dell'ingombro in memoria è particolarmente pesante: si cerca quindi di codificare solo le differenze tra fotogrammi.
- Precisione della codifica: risoluzione dell'immagine, bit/pixel, fotogrammi/secondo (tenendo presente che l'occhio ha un tempo di latenza di 1/25 s)
- Esempi: MPEG 1-2-4, Quicktime, AVI

Ingombro in memoria

- Alcuni esempi:
 - testo equivalente a libro da 200 pagine: 400 KB
 - immagine b/n 512x512: 32 KB
 - immagine 640x480, 256 colori: 300 KB
 - immagine 1024x768, milioni di colori: 2.3 MB
 - 1 minuto di suono, 44 KHz 16 bit: 5.1 MB
 - 1 minuto di video 320x240, 25 f/s, 256 colori: 112.5 MB
- è utile adottare delle strategie per ridurre queste dimensioni

Tecniche di compressione

- si applicano a dati generici, quando possibile
- due categorie:
 - *tecniche reversibili* (lossless), in cui con metodi statistici e matematici viene ridotta la ridondanza dell'informazione dai dati, per cui è possibile riottenere integralmente i dati iniziali; spesso sono tecniche generiche (es. Huffman, LZW)
 - *tecniche irreversibili* (lossy), in cui si elimina dai dati una parte dell'informazione, ritenuta non significativa, per cui non è possibile ricostruire esattamente i dati di partenza, ma se ne ottiene un'approssimazione; spesso sono specifiche per una determinata tipologia di dati (es JPEG)

Esempio: tecnica reversibile

- partendo da una sequenza di bit, ne ricaviamo una nuova codificando le serie di bit uguali con 1 bit per il valore, più 4 bit per il numero di volte che si ripete.

0000000001111110000110000000111111111

rappresentazione numerica

0|1001|10|1100|0100|100100|0111|1001

rappresentazione compressa

- Formati di compressione lossless: GIF, TIFF, ZIP

Tecniche irreversibili

- Per ridurre lo spazio, a volte ci si accontenta di un'approssimazione dei dati iniziali
 - immagini, suoni, video
 - NON per memorizzare programmi, basi di dati, etc
- Esempio: nelle immagini, memorizzare un pixel sì ed uno no, ricostruendo i pixel mancanti per interpolazione
- di solito queste tecniche utilizzano un modello della sorgente e dell'utilizzatore per eliminare informazione che si suppone poco significativa
- JPEG (immagini), basato sul modello della telecamera e dell'occhio
 - MPEG (video): memorizza aree con differenze maggiori di una soglia tra un frame e il successivo
