4. Rappresentazione dell'informazione

4.1 Unità di misura della memoria

4.1.1 Unità elementari di memorizzazione

I sistemi di elaborazione utilizzano **numeri** per *codificare* qualsiasi tipo di informazione, sia essa proprio un numero, un immagine oppure un filmato. Ogni dato viene quindi codificato e trasformato in una serie di cifre per poter essere memorizzato ed elaborato dai computer.
I sistemi elettronici sono però in grado di distinguere solo due diversi stati fisici: acceso o spento, tensione alta o tensione bassa, passaggio di corrente o assenza di corrente, etc. In pratica, essi sono capaci di memorizzare solo cifre binarie o **bit** (***bi****nary digi****t***) e di gestire sequenze di 0 e 1.
La matematica **binaria** è altrettanto efficace di quella *decimale* che ci è più nota o di quella che utilizzi un numero qualsiasi di cifre; la scelta del nostro normale sistema decimale è dettata dalle *dieci dita* delle mani che ogni persona possiede, mentre l'utilizzo del minor numero possibile di simboli, consente la semplificazione e quindi la miniaturizzazione dei circuiti fisici che dovranno eseguire la memorizzazione e le successive elaborazioni.
Il bit costituisce quindi l'**unità elementare** di memorizzazione; visto che la quantità di informazione che può essere contenuta in un singolo bit è minima, per poter codificare dati complessi è necessario lavorare su gruppi di bit.
Un gruppo di 8 bit viene detto *ottetto* o **byte** e consente di codificare **256** (2^8) simboli o dati elementari diversi; ad esempio un byte può essere usato per codificare tutti i simboli (caratteri) presenti in un normale testo, quali lettere maiuscole e minuscole, numeri e segni di interpunzione.
Per definizione il bit viene abbreviato con **b**, mentre il byte con **B**.
Il termine **word** indica, invece, una serie di bit (in numero tale da essere potenza di 2) che hanno un particolare significato; si parla quindi di word di 4, 16, 32 o 64 bit. Normalmente, una *word* è la dimensione minima di bit su cui un calcolatore può eseguire operazioni elementari; i vecchi PC lavoravano con word di 8 o 16 bit, mentre gli attuali elaboratori hanno word di 64 bit.
In definitiva, se per noi è semplice eseguire calcoli nella forma 12+15 o ricercare la parola "ciao" in un testo, per un elaboratore elettronico è molto più facile sommare 1100 a 1110 (intesi come numeri binari) o cercare la sequenza di bit "01110101011101100101011011001101" (possibile codifica della parola "ciao") in una serie di un milione di cifre binarie.

4.1.2 Multipli utilizzati

La memoria utilizzata per codificare una pagina di testo è di qualche migliaio di byte, quella usata per una immagine può raggiungere il milione di byte, mentre un lungo filmato può richiedere miliardi di byte per essere memorizzato.
Si avverte la necessità, come nel *sistema metrico decimale*, di utilizzare dei simboli per rappresentare i multipli delle grandezze elementari; nella terminologia informatica sono stati quindi adottati gli stessi simboli del sistema decimale, ma visto che la misurazione della memoria ha come sua base principale il 2, il loro significato è *leggermente diverso*.
La seguente tabella riassume i simboli e i valori dei multipli più usati in campo informatico:

Multiplo	Sigla	Valore	Approssimazione
Kilo	k	$2^{10} = 1024$	$\approx 10^3$
Mega	M	$2^{20} = 1024^2 = 1024$ k	$\approx 10^6$
Giga	G	$2^{30} = 1024^3 = 1024$ M	$\approx 10^9$
Tera	T	$2^{40} = 1024^4 = 1024$ G	$\approx 10^{12}$

Se per salvare un documento sono necessari 100 kB, significa che la sua occupazione di memoria è di circa centomila byte (esattamente 102400); un floppy disk da 1,44 MB può contenere circa un milione e mezzo di caratteri (1 byte = 1 carattere), mentre un hard disk da 10 GB ha la capacità di memorizzare più di 10 miliardi di caratteri.

4.2 Codifica di strutture elementari

4.2.1 Sistema binario

Come già accennato nel precedente paragrafo, l'elaboratore elettronico trova più semplice operare su *stringhe* di cifre binarie piuttosto che su numeri decimali o caratteri alfanumerici.
Anche se per l'utente finale la situazione è *trasparente*, ogni comando, dalla ricerca di una parola in un testo al ridimensionamento di un immagine, viene tradotto dal computer in una lunga serie di calcoli *elementari* con numeri binari.
Sarà quindi utile conseguire alcune basi del calcolo *non decimale* per poter comprendere, ad esempio, come tale modo di lavorare condizioni l'*approssimazione* dei risultati forniti dall'elaboratore.
Nel sistema binario, a differenza di quello decimale, esistono **solo due cifre**, lo zero e l'uno; il numero di cifre usate da un sistema numerico prende il nome di **base**. Le basi più usate in informatica sono, oltre alla base 2, la base 8 e la base 16 (le potenze di 2 più vicine al 10).
In una qualsiasi base, i primi numeri sono formati dalla serie **ordinata** delle cifre *disponibili*; il numero successivo sarà 10, cioè il più piccolo numero con 2 cifre, e, quando saranno esaurite le possibilità con 2 cifre si passerà a 100 e così via. Mentre nel sistema decimale si può contare fino a 9 prima di passare ai numeri con 2 cifre, in una base **b** qualsiasi si può contare fino a **b-1**; nel sistema binario, ad esempio, il numero che precede il 10 è 1, cioè la maggiore fra le cifre della base.
La successiva tabella riporta i primi 20 numeri nel sistema decimale, binario, ottale e esadecimale (in quest'ultima base le cifre dopo il 9 sono rappresentate dalle prime lettere maiuscole dell'alfabeto):

Decimale	Binario	Ottale	Esadecimale
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13

Il sistema numerico da noi adottato è posizionale, cioè il valore assunto della cifra dipende dalla posizione in cui si trova all'interno del numero; è evidente che nel numero 555, la cifra 5 assume valori diversi, indicando prima le *unità* (10^0), poi le *decine* (10^1) e, infine, le *centinaia* (10^2). Ogni posizione, a partire da destra, implica un **peso** legato alla base. In pratica, se numeriamo le posizioni a partire *da destra* dallo 0, il *peso* di ogni cifra è uguale alla base elevata alla numero della posizione.
Ad esempio, il valore del numero 321 è uguale a $1\times10^0+2\times10^1+3\times10^2 = 1\times1+2\times10+3\times100$. Questo tipo di scrittura si chiama **forma polinomiale** del numero.
In generale, il valore di un numero in una base qualsiasi sarà *uguale alla somma delle sue cifre moltiplicate per il proprio peso*; il numero 253 in base ottale (che può venire più brevemente indicato con 253_8) è uguale a $3\times8^0+5\times8^1+2\times8^2 = 3\times1+5\times8+2\times64 = 3+40+128 = 171$ in base decimale.
La conversione inversa, da base 10 ad una base qualsiasi, può essere effettuata tramite il metodo delle divisioni successive, composto dai seguenti passi:

1) si divide il numero decimale per b; il resto viene scritto a destra della divisione;
2) si divide il quoziente precedente per b; il resto viene scritto a destra della divisione;
3) si ripete l'operazione 2 fino ad ottenere un quoziente nullo.

Il numero in base b sarà formato dai resti letti dal basso verso l'alto.
Ad esempio, il numero 26 decimale sarà uguale al numero 11010 in binario; infatti:

26 : 2 = 13	resto	0
13 : 2 = 6	resto	1
6 : 2 = 3	resto	0
3 : 2 = 1	resto	1
1 : 2 = 0	resto	1

Le conversioni fra le basi 2, 8 e 16 sono particolarmente semplici visto che si tratta di basi tutte potenze di 2; per passare, ad esempio da un numero binario ad uno ottale, basta procedere secondo il seguente metodo:
si raggruppano a tre a tre ($8 = 2^3$) le cifre binarie, a partire da destra; ad ogni gruppo di cifre binarie si sostituisce la cifra ottale corrispondente (vedi la tabella precedente).
Consideriamo, ad esempio, il numero 100110111_2; suddividiamolo in gruppi: 100 110 111 e sostituiamo le corrispondenti cifre ottali: 100 = 4, 110 = 6 e 111=7; il numero ottale sarà 467_8.
Se si vuole passare dalla base 2 alla base 16 basterà semplicemente raggruppare le cifre **a quattro a quattro,** perché $16 = 2^4$.
Le operazioni fondamentali in base 2 (similmente a qualsiasi altra base) si eseguono con le stesse regole usate per il calcolo decimale, tenendo conto delle cifre disponibili, dei *riporti* e dei *prestiti*.
Vediamo degli esempi di operazioni in base 2:

Addizione:

Primo addendo		1	0	1	0	
Secondo addendo		1	0	1	1	
Riporti	1		1			
Risultato	1	0	1	0	1	

Sottrazione:

Prestiti			2	2			
Minuendo			1	1	0	1	1
Sottraendo				1	1	0	1
Risultato					1	1	0

Moltiplicazione:

Primo fattore				1	0	1	1
Secondo fattore				1	0	1	0
				0	0	0	0
			1	0	1	1	
		0	0	0	0		
	1	0	1	1			
Risultato	1	1	0	1	1	1	0

4.2.2 Numeri binari relativi

Per rappresentare numeri interi positivi, il calcolatore usa il sistema binario puro; quando, però, si vogliano utilizzare insiemi numerici più estesi, risultano essere necessarie delle tecniche di codifica più raffinate.
Se, ad esempio, i calcoli coinvolgono i numeri relativi interi, la loro rappresentazione dovrà essere fatta in modo da essere economica in termini di **spazio** e di **tempo di esecuzione** delle operazioni.
Esistono quattro tecniche per organizzare la codifica dei numeri binari negativi:

- **Modulo e segno**: il *modulo* del numero viene rappresentato normalmente, mentre un bit posto a sinistra indica il *segno*: 0 positivo e 1 negativo. Ad esempio, la sequenza 1 1010 indica il numero decimale –10.
  E' il metodo più semplice, ma presenta due problemi: lo zero è rappresentato due volte come –0 e +0; inoltre sommando con le normali regole due numeri relativi qualsiasi codificati con modulo e segno, non sempre si ottiene un risultato corretto. Con 8 bit si possono rappresentare i numeri da –127 a +127.
- **Complemento alla base diminuita**: per rappresentare nella base *b* il numero negativo di modulo *N* con un numero *n* di cifre, sarà necessario compiere il calcolo $\mathbf{b^n - N}$; ad esempio, la rappresentazione di –23 in base decimale con 4 cifre sarà $10^4 - 23 = 10000 - 23 = 9977$.
  Nel caso in cui la base sia 2, il calcolo $b^n - N$ diventa particolarmente semplice perché basta *invertire* i bit di N (da 0 a 1 e viceversa); non viene risolto il problema del *doppio zero*, ma tutti i calcoli con i numeri relativi possono essere correttamente eseguiti usando solo addizioni.
- **Complemento alla base**: metodo simile al precedente, modifica la formula di rappresentazione in $\mathbf{b^n - N - 1}$; utilizzando 4 cifre, il numero decimale –150 sarà rappresentato con $10^4 - 150 = 10000 - 150 - 1 = 9849$.
  Nel caso di numeri binari, la modifica consente di eseguire comunque in modo facile la conversione ed elimina il problema del doppio zero. I numeri rappresentabili con 8 bit vanno da –128 a +127, ma in alcuni casi il risultato della somma di due numeri relativi non è corretto per il superamento dei limiti di rappresentazione; in questi casi si parla di **overflow**.
  E' il metodo più usato (in modo quasi esclusivo) per la rappresentazione degli interi relativi.
- **Eccesso M**: il numero relativo N viene rappresentato usando la formula N + M con M numero opportuno; i numeri –5 e +5 decimali, vengono rappresentati con 495 e 505 se M = 500.

Nel caso binario, il metodo di trasformazione non è elementare (dipende da M), ma consente di avere rappresentazioni *asimmetriche* rispetto allo zero conservando i vantaggi della precedente rappresentazione; se vogliamo invece ottenere una equa distribuzione di numeri positivi e negativi, allora M deve essere uguale a b^{n-1} o b^{n-1}-1, dove b è la base e n sono il numero delle cifre disponibili per la rappresentazione.

Nella seguente tabella vengono riassunti i metodi sopra descritti relativamente alla rappresentazione binaria a 4 cifre.

Sequenza Binaria	**Modulo e segno**	**Complemento a 1**	**Complemento a 2**	**Eccesso 8**
0000	0	0	0	-8
0001	1	1	1	-7
0010	2	2	2	-6
0011	3	3	3	-5
0100	4	4	4	-4
0101	5	5	5	-3
0110	6	6	6	-2
0111	7	7	7	-1
1000	-0	-7	-8	0
1001	-1	-6	-7	1
1010	-2	-5	-6	2
1011	-3	-4	-5	3
1100	-4	-3	-4	4
1101	-5	-2	-3	5
1110	-6	-1	-2	6
1111	-7	-0	-1	7

4.2.3 Rappresentazione in virgola mobile

I metodi precedenti sono utilizzati per operare con numeri **relativi interi**; con piccole modifiche (introduzione del *punto decimale*) potrebbero essere usate anche per i numeri **decimali**, ma con scarsa efficacia. Nei calcoli che un elaboratore si trova ad affrontare, possono comparire, infatti, quantità *molto grandi* e *molto piccole* che richiederebbero un elevato numero di bit per essere rappresentate con la giusta precisione.

Per ovviare a tale problema si ricorre ad una particolare *notazione* detta **esponenziale** o in **virgola mobile** (*floating point*) simile alla notazione *scientifica* fornita da alcune calcolatrici tascabili.

In questa notazione, ogni numero viene espresso come il prodotto di una certa quantità (detta **mantissa**) per una **base** specifica elevata ad una **potenza**.

Nel caso di base decimale saranno utili i seguenti esempi:

471 000 000 000 può essere scritto in notazione esponenziale come $4.71 \bullet 10^{11}$

0. 000 000 012 può essere scritto in notazione esponenziale come $1.2 \bullet 10^{-8}$

Dagli esempi si nota il notevole risparmio in termini di cifre utilizzate, che si ottiene senza sacrificare la **precisione** dei valori; ciò viene ottenuto eliminando le cifre che servono solo ad esprimere l'ordine di grandezza del numero (che viene condensato nell'esponente).

Un numero frazionario binario viene quindi rappresentato da una serie di bit suddivisi in tre gruppi (viene infatti sottointeso che la *base* è 2):

- un bit per il *segno* del numero
- alcuni bit per la *mantissa*
- alcuni bit per l'*esponente*

L'organismo di standardizzazione internazionale **ANSI**, ha emesso alcune specifiche (IEEE 754-1985) per uniformare il modo con cui vengono rappresentati i numeri *floating point* evitando incomprensioni dovute, ad esempio, ad una diversa distribuzione dei bit fra i vari gruppi.
Lo standard prevede che un numero N sia espresso nella seguente forma:

$$N = (-1)^{\mathbf{S}} \bullet 2^{\mathbf{E-M}} \bullet 1.F$$

Dove S è il segno (0 = positivo, 1 = negativo), E è l'esponente espresso in eccesso M e F è la mantissa frazionaria; per recuperare un bit, si sceglie di usare come mantissa i numeri da 1.000… a 1.111… e quindi si sottointende l'uno iniziale.
Lo standard prevede tre tipi di numeri *floating point* che possono rappresentare numeri via via più **precisi**; ciò si ottiene, naturalmente, aumentando il numero di bit utilizzati.
Nei numeri in **singola precisione** vengono usati 32 bit (4 byte) suddivisi in 8 bit di esponente, 23 bit di frazione oltre al bit di segno; dei 64 bit (8 byte) della **doppia precisione**, 11 sono riservati all'esponente, mentre 52 formano la parte frazionaria; nel caso di rappresentazione in **quadrupla precisione**, i bit per l'esponente sono 15, quelli per la frazione 111, per un totale di 128 bit (16 byte), includendo il bit di segno.
Per rappresentare, ad esempio, in *singola precisione* il numero **N = -157.6875** (utilizzando per la rappresentazione dell'esponente l'eccesso 128) avremo:

$N = -157.6875 = -\ 10011101.1011_2 = -\ 1.00111011011_2 \bullet 2^7$
S = 1 (segno negativo) (1 bit)
$E = 7 + 128\ (M) = 135 = 10000111_2$ (8 bit)
$F = 00111011011000000000000_2$ (23 bit)

Esiste anche un altro metodo per codificare i numeri decimali, diverso dal binario puro; si tratta del codice **BCD** (*Binary Coded Decimal*) che, utilizzato solo in particolari casi, trasforma ogni cifra decimale nel suo corrispondente binario.
In pratica, ogni cifra che forma il numero decimale di partenza viene trasformata in un gruppo di **quattro bit** che rappresentano l'equivalente binario della cifra decimale; con tale sistema il numero 382 decimale ha come equivalente BCD la serie di bit 0011 1000 0010.
Il vantaggio del sistema è una *facile traduzione* dal decimale: è sufficiente utilizzare una tabellina con i primi 10 numeri binari; per contro i calcoli risultano farraginosi e l'occupazione di memoria non è ottimale per il fatto che delle 16 possibili combinazioni ottenibili con 4 bit ne vengono sfruttate solo 10.

4.2.4 Valori di verità

La moderna matematica (e quindi l'informatica) non può prescindere dall'utilizzare, oltre ai numeri, anche altri **oggetti** su cui possono essere compiute delle generiche *operazioni*.
Una particolare branca della matematica detta **algebra di Boole**, ad esempio, è formata da un insieme di funzioni che operano su delle quantità variabili o costanti che possono assumere solo **due** valori (vero o falso, 0 o 1, alto o basso, ecc.); l'interpretazione, suggerita dalla *logica*, dei due valori come **vero** e **falso** ha portato al nome di **valori di verità**, mentre per le quantità si parla di grandezze di tipo **booleano**.
Dal punto di vista della rappresentazione non vi è alcuna differenza tra un valore di verità e una cifra binaria; per tanto una variabile *booleana* può essere memorizzata in un **singolo bit**.
Dal punto di vista teorico, invece, la diversità è notevole, visto che le operazioni che possono essere effettuate sulle variabili booleane sono totalmente diverse dalle operazioni numeriche e fanno parte del calcolo **logico** o **proposizionale**.

4.2.5 Codice ASCII

I moderni calcolatori sono in grado di manipolare, con la stessa facilità con cui operano sui numeri, anche elementi diversi come **caratteri** o sequenze di caratteri (parole, frasi, paragrafi, ecc.).
Ovviamente i circuiti fisici che attuano effettivamente i comandi eseguono *sempre e solo* calcoli su numeri binari, ma una opportuna **codifica** e, successivamente, un software di livello superiore, consentono di eseguire *operazioni* sul testo quali ricerca e/o sostituzione di parole o di intere frasi, controlli ortografici e sintattici, ecc.
Per poter codificare i simboli che compaiono in un testo (lettere, numeri, segni di interpunzione, operatori matematici, ecc.) si possono utilizzare delle **tabelle** che, ad un determinato numero fanno corrispondere un determinato simbolo alfanumerico.
Dato che non esiste un ordine riconosciuto all'interno dei caratteri alfanumerici, si è deciso, per evitare il formarsi di diverse interpretazioni della stessa sequenza di bit, di adottare un codice unificato determinato dall'ANSI detto **ASCII** (pronuncia *as-kii* o, italianizzato *asci*).
Il codice **ASCII Standard** utilizza **7 bit** per ogni carattere; è così possibile codificare **128 simboli** diversi che vengono suddivisi in alcuni gruppi: *caratteri di controllo* (0-31), simboli di interpunzione, cifre arabe, lettere maiuscole e minuscole e altri simboli particolari.
Dato che l'unità base di memorizzazione è il *byte* (8 bit), un bit non viene utilizzato dal precedente codice; sono così nati i codici **ASCII Estesi** che, sfruttando il bit residuo, consentono di memorizzare altri 128 simboli. Gli ulteriori simboli non sono standardizzati e sono usati per caratteri grafici locali (ad esempio le *lettere accentate*) o per interessi particolari del costruttore (*set grafico* di Apple).
Esiste anche un altro codice per la memorizzazione di caratteri alfanumerici, che però sta andando rapidamente in disuso; si tratta dell'**EBCDIC** (*Extended Binary Coded Decimal Interchange Code*), un codice adottato dall'IBM per i suoi computer di classe superiore (mini e mainframe).
Tale codice non viene più utilizzato per la mancanza di *standardizzazione internazionale* e per il fatto di non avere un ordine *strettamente sequenziale e contiguo* per i vari gruppi di caratteri.

4.3 Codifica di strutture complesse

4.3.1 Codifica di stringhe e vettori

Dopo aver illustrato la problematica della codifica di *semplici* informazioni (quali numeri e caratteri alfanumerici), consideriamo il caso della memorizzazione di strutture più *complesse*, a partire da quelle formate da *collezioni ordinate* di dati semplici.
Attualmente i computer vengono usati, oltre che per trattare l'informazione numerica, anche per quella che viene chiamata elaborazione testi, la possibilità cioè di manipolare insiemi di caratteri per ottenere lettere commerciali, relazioni, articoli di giornale, ecc.
Una **stringa** è una serie ordinata di caratteri alfanumerici; ogni singola parola e ogni frase complessa sono quindi da considerarsi delle stringhe.
Per codificare una stringa si può far uso delle tecniche viste per i singoli caratteri, tenendo conto che sarà necessario almeno un ulteriore byte, oltre a quelli necessari per l'informazione da memorizzare, per indicare o il numero di caratteri componenti la stringa (che in tal caso dovrà essere più piccola di 256 caratteri) o la fine della stringa medesima.
Se la stringa che andiamo a memorizzare è un file di testo complesso come quello generato da *programmi di videoscrittura*, il numero di byte aggiuntivi alla codifica dei semplici caratteri *aumenta*; infatti dovranno essere fornite indicazioni relativamente al tipo di stile usato, alla dimensione dei caratteri, al loro colore, etc.
In questo caso una pagina di un migliaio di caratteri andrà ad occupare una decina di Kbyte, contro il Kbyte richiesto dal testo puro (senza *formattazioni*).
Una seconda struttura complessa è il **vettore** (*array*), che costituisce una generalizzazione del concetto di stringa; infatti, si definisce come vettore una sequenza omogenea di informazioni di uno

stesso tipo (numeri, caratteri o altro). Ad esempio, le temperature rilevate alle ore 13 di ogni giorno del mese di Agosto a Udine, costituiscono un vettore di 31 numeri.
Per la reale memorizzazione dell'informazione, ci si basa sulla codifica dei singoli dati che formano il vettore, tenendo conto del fatto che saranno necessari ulteriori byte per specificare la natura del vettore e la sua dimensione.
Su questa base è possibile, ovviamente, costruire strutture più complesse quali **matrici** *bidimensionali*, *tridimensionali* o, in generale, n-*dimensionali*. Si tratterà infatti di impostare vettori che hanno per loro elementi altri vettori e così via fino ad ottenere la dimensione desiderata.

4.3.2 Codifica di immagini

L'utilizzo delle immagini sugli elaboratori è stato reso possibile dall'aumentata potenza di calcolo e di memoria dei computer che finalmente sono riusciti a gestire la grossa mole di dati contenuta in una semplice immagine.
La codifica delle immagini richiede, infatti, un ulteriore passaggio *logico* rispetto alla manipolazione di cifre e caratteri alfanumerici; in questi ultimi casi esiste un'unità minima di riferimento, mentre una immagine è, per sua natura, un *insieme continuo di informazioni*.
Si sono così ideate due strade per rendere **digitale** una informazione prettamente **analogica**: 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.
Nel primo caso si parla di **immagini bitmap**: infatti l'immagine è scomposta in un reticolo di punti, detti **pixel** (*picture element*) e per ogni punto vengono memorizzate alcune caratteristiche; se, ad esempio, permetto che ogni punto sia o bianco o nero, mi servirà un bit per ogni pixel, mentre se voglio sfruttare le sfumature di grigio o il colore, l'informazione da associare ad ogni singolo punto sarà maggiore.
La qualità dell'immagine dipende dal numero di punti in cui viene suddivisa e dai toni di colore permessi dalla codifica; formati tipici di risoluzione di immagini sullo schermo sono 640x480, 800x600, 1024x768, mentre i colori ammessi vanno da 2 a 16,8 milioni di colori.
L'occupazione di memoria delle immagini è notevole; la seguente tabella ne dà una idea.

Tipo di immagine	Definizione	Colori	Occupazione di memoria
Televisiva	720 x 625	256 (8 bit)	440 KB
Monitor VGA	640 x 480	16 (4 bit)	150 KB
Monitor SVGA	1024 x 768	65536 (16 bit)	1,5 MB
Monitor XVGA	1280 x 1024	16,8 milioni (24 bit)	3,8 MB
Fotografia	15000 x 10000	16,8 milioni (24 bit)	430 MB

Le immagini bitmap vengono quindi memorizzate come una lunga sequenza di bit il cui significato dipende dalla particolare codifica adottata; esistono diversi tipi standard di *file di immagine* che possono contemplare anche la possibilità di **comprimere** (come vedremo in un prossimo paragrafo) l'immagine di partenza.
I formati file più comuni sono il **TIFF** (*Tagged Image File Format*), il **GIF** (*Graphics Interchange Format*), il **JPEG** (*Joint Photographers Expert Group*) e il **BMP** (*Bit MaP*).
Quando le immagini da memorizzare hanno caratteristiche geometriche ben definite, come nel *disegno tecnico*, è possibile adottare una tecnica più efficiente per codificare le figure. Nel caso di progettazione architettonica, meccanica o elettronica, il disegno da memorizzare può essere facilmente *scomposto in elementi base* come una linea o un arco di circonferenza.
La memorizzazione dell'intera immagine avviene tramite la codifica di ogni singola parte: per definire un segmento basteranno, infatti, le coordinate dei due estremi, così come per una circonferenza saranno sufficienti raggio e coordinate del centro.

Questo tipo di codifica viene detto **vettoriale** e le sue principali caratteristiche sono:

- *indipendenza dalla risoluzione* a cui dovranno essere visualizzate le immagini; con questo metodo vengono memorizzati degli enti astratti che potranno essere mostrati con la massima risoluzione disponibile per il dispositivo usato.
- *possibilità di manipolare* facilmente l'immagine; semplici funzioni matematiche potranno essere usate per spostare mutuamente i vari elementi consentendo, in modo semplice, la creazione, ad esempio, di circonferenze concentriche (stesse coordinate per il centro).
- *applicabilità limitata*; non tutte le immagini possono essere scomposte efficacemente in parti elementari (ad esempio la fotografia di un paesaggio).
- *Necessità di software specifico*; l'immagine può essere ricostruita solo se si dispone di un software che riconosca il particolare formato.

I principali formati utilizzati sono il **DXF** (*Drawing eXchange Format*), usato anche dal programma di disegno tecnico *AutoCad*, e lo **IGES** (*Initial Graphicc Exchange Specifications*), di più elevato livello ed associato alla modellazione tridimensionale.
Ulteriori, importanti formati per la memorizzazione delle informazioni grafiche sono il **PostScript** e il **PDF** (*Portable Document Format*) che consentono di trattare le immagini con entrambi i metodi, bitmap e vettoriale, in quasi tutte le tipologie di elaboratori.
Il problema della memorizzazione delle *immagini in movimento* (**video**) viene risolto allo stesso modo in cui il cinema o la televisione lo hanno affrontato; sfruttando 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 (25 al secondo).
Un video viene quindi trattato come una sequenza di immagini (e di suoni associati) e come tale viene memorizzato; lo standard principale, il **MPEG** (*Moving Picture Expers Group*), associa alla semplice codifica di ciascuna immagine anche tecniche per il suono e, soprattutto, **modalità di compressione** che sfruttano il fatto che la differenza tra un frame e il successivo è minima. Senza la compressione, un filmato di **1 minuto** in qualità televisiva occuperebbe oltre **600 MB** (un intero CD).

4.3.3 Codifica audio

Un **segnale audio** è, similmente ad una immagine, una informazione **continua**; in *ogni istante* l'**ampiezza** del segnale può variare *liberamente* in maniera *analogica*.
Per poter **digitalizzare** (e quindi memorizzare) il suono sono quindi necessarie due operazioni: il **campionamento** e la **quantizzazione**; la prima permette di analizzare il segnale ad intervalli di tempo determinati, mentre la seconda costringe l'ampiezza ad assumere solo un determinato numero di valori.
L'operazione di *campionamento* va a misurare il segnale con una particolare **frequenza** (dell'ordine delle decine di KHz, cioè migliaia di volte al secondo) e, poi, un particolare circuito elettronico ricostruirà il segnale originale; in pratica, per poter ottenere il segnale iniziale **senza perdite** è necessario avere una elevata **frequenza di campionamento**, pari ad almeno due volte la frequenza massima del segnale stesso.
La *quantizzazione* del segnale è un procedimento attraverso il quale si attribuisce ad ogni campione di segnale individuato nella precedente fase un valore di ampiezza tra una serie di possibili valori; la consistenza numerica di tale serie sarà stabilità dalla quantità di bit utilizzati per tale operazione: con 3 bit potremo avere 8 (2^3) livelli diversi, mentre con 16 il numero di valori distinguibili sale a 65536 (2^{16}).
La quantizzazione è, diversamente dal campionamento, un processo **irreversibile** che conduce ad una sicura *perdita di informazioni*; tanto più l'operazione sarà accurata tanto più la qualità del suono sarà conservata riducendo al minimo quello che viene detto **rumore di quantizzazione**.

Il metodo scelto per digitalizzare l'informazione audio risente, oltre che della memoria a disposizione, della *velocità del flusso dei dati* nei dispositivi usati visto che questo tipo di informazione deve, normalmente, essere trasmessa, ricevuta ed elaborata in **tempo reale**.
I formati di memorizzazione più usati sono l'**AIFF** (*Audio Interchange File Format*), il **Midi** e il più recente **MP3**, che consente riproduzioni audio di alto livello.
Una registrazione audio di alcuni minuti occupa uno spazio di memoria che va dalle **decine di KB** alle **decine di MB**, a seconda dei formati adottati e della presenza o dell'assenza di *compressione* dell'informazione.

4.3.4 Compressione dei dati

I dati che un elaboratore deve trattare, soprattutto se si tratta di **informazioni multimediali**, richiedono a volte *elevati spazi di memoria* per poter essere registrati e canali trasmissivi ad *alta velocità* per essere trasferiti ad altri sistemi.
Nasce l'esigenza di limitare, quindi, la dimensione di tali informazioni attraverso delle **tecniche di compressione** che basano la loro efficacia sulla considerazione, ad esempio, che non tutti gli elementi dell'informazione hanno la stessa *frequenza statistica* (ad esempio è molto più semplice trovare in un testo un carattere "a" piuttosto che il carattere "y") oppure sulla presenza di *elementi ripetuti* (immagini con zone dello stesso colore o tabelle con dati replicati).
I dati compressi *non sono immediatamente utilizzabili* dall'elaboratore, ma richiedono un processo di *decodifica* per poter essere usati dall'utente; esistono centinaia di tecniche di compattamento e relativa **decompressione** che si dividono in due grandi gruppi: **tecniche senza perdita** e **tecniche con perdita**.
Nel primo caso viene garantito che, tramite il processo di decompressione, vengono ottenuti **esattamente** gli stessi dati di partenza; per tale ragione il **fattore di compressione** raggiungibile non è elevato e si attesta, in genere, tra 2:1 e 4:1, variando sensibilmente in relazione al tipo di dati da comprimere.
Tra le codifiche di tale tipo, le più usate sono la **RLE** (*Run Length Encoding*), che lavora su sequenze di **elementi uguali**, sostituendo ad esse l'elemento stesso, ripetuto tre volte, seguito dal suo numero di occorrenze, e la famiglia delle **codifiche adattative** in base alla frequenza; queste ultime lavorano rappresentando i diversi caratteri con stringhe di bit di lunghezza **inversamente proporzionale** alla frequenza con cui compaiono, riducendo così il numero globale di bit impegnati (il carattere "a" sarà, ad esempio, codificato con 4 bit, mentre il carattere "y", meno frequente, con 10). Le codifiche adattative più raffinate sfruttano anche le frequenze statistiche legate ai caratteri **successivi**; sarà così possibile codificare una "u" con molti bit in generale e con pochi nel caso che il carattere precedente sia una "q".
Altra tecnica di compattamento del tipo *senza perdita,* è la **LZW** (*Lempel Ziv Welch*), che trova applicazione a fianco di codifiche come GIF, PostScript e PDF. In tal caso *intere stringhe* di caratteri vengono compresse e decompresse utilizzando particolari **tabelle** che andranno a far parte dei dati trasmessi o memorizzati in modo compattato.
Le tecniche senza perdite sono particolarmente adatte alla compressione di dati principalmente testuali o numerici, ma hanno una *efficienza limitata*; nel caso in cui l'informazione sia comprensibile anche se sottoposta a *limitate trasformazioni* (suoni ed immagini, ad esempio), si può sacrificare l'esattezza dei dati a favore di rapporti di compressione di alcuni ordini di grandezza superiori a quelli precedentemente visti.
Le tecniche di **compressione con perdita**, sfruttano *trasformazioni* dell'informazione legate al comportamento dei nostri sistemi sensoriali (vista e udito), in modo da ridurre in maniera sensibile la quantità di memoria necessaria senza avere *grandi alterazioni* dei dati.
Queste tecniche, molto efficienti, sono però **irreversibili** e non consentono facili elaborazioni dell'informazione ottenuta, per l'esponenziale aumento delle perdite introdotte. Per contro, tali tecniche consentono la scelta del **miglior compromesso** tra *perdita di dati* e *rapporto di compressione*.

Le tecniche di compressione con perdita vengono attualmente utilizzate per *immagini fisse*, per *immagini in movimento* e per l'*audio*; per le immagini fisse, la tecnica **JPEG** già descritta, viene affiancata da una compressione ottenuta dividendo il reticolo di pixel in blocchi di dimensione 8x8, *normalizzando* tali blocchi e, in seguito, codificandoli.
Per le immagini in movimento la tecnica più usata è inserita nella codifica **MPEG**; si tratta di una modalità di compressione asimmetrica, cioè il compattamento è molto più complesso dell'operazione di decompressione.
La compressione in MPEG è ottenuta generando tre tipi di frame (immagini): **intraframe** (normali immagini fisse), **predicted frame** (immagini dedotte dall'immagine precedente) e **bidirectional frame** (immagini ottenute per interpolazione tra una immagine e la successiva); i frame più onerosi in termini di memoria, gli *intraframe*, sono presenti in quantità inferiori al 10%.
La compressione di informazioni **audio** è consentita da studi *psico-acustici* che evidenziano come, in presenza di suoni forti e di particolari frequenze, l'orecchio umano non ne percepisca altri più deboli che vengono, in pratica, *mascherati*.
La tecnica **PASC** (*Philips Precision Sub-band Adaptive Coding*) utilizzata nei **DCC** (*Digital Compat Cassette*) e la codifica **ATRAC** (*Adaptative TRasform Acoustic Coding*) usata nei *MiniDisc*, sono applicazioni di tali concetti e consentono rapporti di compressione di 5:1.
Per la trasmissione o la memorizzazione **efficiente** dell'informazione sono necessarie, oltre alle tecniche di compressione, anche metodologie finalizzate alla **ricerca** e alla **correzione degli errori**; tali obiettivi sono realizzati attraverso i **codici a correzione d'errore** (**ECC**, *Error Correcting Code*) al prezzo di un incremento della memoria utilizzata.
Il metodo più semplice si base sul **bit di parità**: ad ogni byte viene aggiunto un bit il cui valore rende *pari* il numero di 1 presenti nel byte; se in ricezione (o in lettura) si trova un numero dispari di 1 vuole dire che si è verificato un *errore* nella trasmissione del byte. Per la correzione di tale errore è necessario aggiungere un *byte di parità* che potrà correggere il bit errato precedentemente individuato.
Esaminiamo, ad esempio, il seguente caso:

0 1 0 1 0 1 1 0 0	0 1 0 1 0 1 1 0 0	0 1 **0** 1 0 1 1 0 0
1 1 0 0 1 0 1 1 1	1 1 0 0 1 0 1 1 1	1 1 **0** 0 1 0 1 1 1
0 1 1 1 0 1 1 1 0	0 1 1 1 0 1 1 1 0	**0 1 0 1 0 1 1 1 0**
1 1 0 0 1 1 0 1 1	1 1 0 0 1 1 0 1 1	1 1 **0** 0 1 1 0 1 1
0 0 0 1 0 1 0 1 1	0 0 0 1 0 1 0 1 1	0 0 **0** 1 0 1 0 1 1
1 0 0 0 0 1 1 0 1	1 0 0 0 0 1 1 0 1	1 0 **0** 0 0 1 1 0 1
1 0 0 1 0 1 0 1 0	1 0 0 1 0 1 0 1 0	1 0 **0** 1 0 1 0 1 0
0 0 1 0 0 0 1 0 0	0 0 1 0 0 0 1 0 0	0 0 **1** 0 0 0 1 0 0
	0 0 0 0 0 0 1 1 0	0 0 **0** 0 0 0 1 1 0
Con bit di parità	Con byte parità	Determinazione dell'errore

Il metodo, molto usato, funziona però solo con un **singolo errore** per ogni blocco di dati; *errori multipli* possono mascherarsi a vicenda o introdurre falsi errori.
Le tecniche attualmente in uso si basano sulla più complessa codifica **Reed-Salomon** (dal nome dei due studiosi che la progettarono), datata anni '60 e in grado di correggere un numero arbitrario di bit o di byte.