Corso di INFORMATICA
Corso di Laurea Triennale in Matematica

Rappresentazione dell'Informazione

Rev. 2009 by Mario Alviano

Sommario

Rappresentazione dell'informazione

L'informatica è la scienza che studia i modi in cui l'informazione può essere rappresentata e manipolata in un sistema informatico (un computer). Per manipolazione intendiamo elaborazione: lo scopo dell'informatica è elaborare dati, automaticamente, per raggiungere un determinato obiettivo. Per meglio chiarire questo concetto consideriamo un programma che determini l'area di un quadrato data la lunghezza del lato. In questo caso, il programma elaborerà l'informazione "lunghezza del lato" al fine di determinare l'area del quadrato, ad esempio applicando la nota formula "lato × lato".
L'informazione può essere elaborata solo se rappresentata in modo opportuno. Ad esempio, è possibile leggere un libro solo se scritto in un linguaggio noto, ovvero se i "concetti" (le informazioni) che contiene sono rappresentate in modo comprensibile al lettore. Allo stesso modo, affinché un computer possa elaborare delle informazioni, queste devono essere rappresentate secondo determinate regole. L'insieme delle regole che definiscono come un'informazione debba essere rappresentata prende il nome di codifica.
Nel seguito approfondiremo questi concetti e discuteremo brevemente le codifiche delle informazioni fondamentali, concentrandoci in particolare sulla rappresentazione dei numeri (interi e non).

Concetti introduttivi: informatica, algoritmo, programma

Per informazione intendiamo una notizia o un dato che consente di avere conoscenza di fatti o situazioni. Ad esempio, un'informazione può essere la lunghezza del lato di un quadrato.
L'informatica, come già detto, è la scienza che studia la rappresentazione e l'elaborazione dell'informazione. L'elaborazione è volta alla risoluzione di un problema di interesse, che definisce le informazioni in input e le informazioni che dovranno essere fornite come risposta (l'output). Ad esempio, per il problema del calcolo dell'area del quadrato descritto sopra, l'input è costituito dalla lunghezza del lato del quadrato, e l'output che ci aspettiamo è l'area del quadrato.
L'elaborazione delle informazioni consiste in una sequenza determinata di operazioni eseguite sui dati di partenza (l'input). L'insieme di queste operazioni viene detto algoritmo. Più formalmente, un algoritmo è una sequenza di operazioni che risolvono un problema. In generale, un problema può essere risolto da diversi algoritmi. Ad esempio, il problema del calcolo dell'area di un quadrato può essere risolto dai seguenti algoritmi:

1. Sia L la lunghezza del lato del quadrato fornito in input; restituisci L×L.
2. Sia L la lunghezza del lato del quadrato fornito in input; restituisci L^2.

Come ogni altra informazione, un algoritmo può essere compreso (o meglio eseguito) dal computer solo se rappresentato in modo opportuno per mezzo di un programma. Per questo motivo si dice che un programma implementa un algoritmo, ad evidenziare il fatto che un programma non è altro che una rappresentazione di un algoritmo che un computer può eseguire. Come un problema può essere risolto da diversi algoritmi, un algoritmo può essere implementato da diversi programmi.

Introduzione alla rappresentazione dell'informazione

L'idea fondamentale alla base della rappresentazione dell'informazione è il BIT (Binary digiT, ovvero cifra digitale). Il BIT costituisce l'unità minimale di rappresentazione in quanto può rappresentare uno di due valori: 0 oppure 1. La semplicità del BIT ne consente una facile e naturale rappresentazione in un calcolatore elettronico. Potendo assumere solo due valori, infatti, un BIT può essere associato alla presenza/assenza di corrente elettrica. D'altro canto, potendo assumere solo i valori 0 e 1, le uniche informazioni rappresentabili da un BIT sono quelle a due valori. Ad esempio, con un BIT possiamo rappresentare lo stato di un interruttore, ON oppure OFF, associando 1 al primo stato e 0 al secondo. Lo stato di un interruttore ha infatti un dominio di soli due valori, ON e OFF appunto. In modo simile possiamo distinguere fra giorno e notte. Non potremmo invece rappresentare la stagione in cui ci troviamo. Questo tipo d'informazione ha infatti un dominio di quattro elementi: primavera, estate, autunno e inverno. In questi casi possiamo combinare più BIT allo scopo di rappresentare informazioni più complesse. Ad esempio, per rappresentare una stagione sono sufficienti 2 BIT:

1° BIT	**2° BIT**	**Stagione**
0	0	Primavera
0	1	Estate
1	0	Autunno
1	1	Inverno

Quindi, seguendo questa rappresentazione, per indicare che ci troviamo in autunno useremo la coppia di BIT (1,0).
Per rappresentare un giorno della settimana 2 BIT non bastano, ma 3 sono sufficienti. Un possibile schema di rappresentazione è il seguente:

1° BIT	**2° BIT**	**3° BIT**	**Giorno della settimana**
0	0	0	Lunedì
0	0	1	Martedì
0	1	0	Mercoledì
0	1	1	Giovedì
1	0	0	Venerdì
1	0	1	Sabato
1	1	0	Domenica
1	1	1	NON USATO

Seguendo questo schema, la tripla (0,1,0) è usata per indicare il giorno Mercoledì. Visto che i BIT possono assumere due soli valori, 0 oppure 1, una coppia, una tripla e più in generale una n-pla sono generalmente rappresentate come successioni di zeri e di uno, omettendo le virgole e le parentesi. Quindi, per indicare

Sabato useremo 101, mentre con 011 indicheremo Giovedì. Come è possibile osservare dallo schema, una delle 8 combinazioni (111) dei 3 BIT è di troppo. I giorni della settimana sono infatti 7. In questi casi le combinazioni in eccesso non saranno utilizzate in quanto non significative.
In generale, N BIT possono assumere 2^N combinazioni diverse, ognuna delle quali può essere associata a un valore che una specifica informazione può assumere. Ne consegue che con N BIT possiamo rappresentare informazioni con un dominio di al più 2^N valori. Viceversa, per rappresentare un'informazione con un dominio di M valori avremo bisogno di almeno $min_{N \in \mathbb{N}} 2^N \geq M$ BIT. Ad esempio, per rappresentare un'informazione che può assumere uno di 57 valori diversi avremo bisogno di almeno 6 BIT. Infatti, $2^6 = 64 > 57$, mentre $2^5 = 32 < 57$.
Particolarmente importanti sono le sequenze di 8 BIT, comunemente chiamate BYTE. Con un BYTE possiamo rappresentare informazioni con dominio di al più $2^8 = 256$ elementi. Generalmente, informazioni più complesse (ovvero con dominio di più di 256 elementi) vengono rappresentate con successioni di BYTE. Quindi, in informatica difficilmente si troveranno informazioni codificate in 10 BIT, preferendo in questi casi l'uso di 2 BYTE (ovvero 16 BIT).
Come per ogni altra unità di misura, i multipli del BYTE prendono nomi specifici. Generalmente i multipli delle unità di misura seguono le potenze di 10, che è la base che comunemente adottiamo. In informatica la base più comune è invece la 2 (come le combinazioni esprimibili da un BIT), e pertanto anche i multipli seguiranno questa base. Per non stravolgere il nostro modo di pensare, le potenze di 2 scelte per i multipli del BYTE sono quelle che più si avvicinano alle potenze di 10. In particolare, i primi multipli del BYTE sono KiloByte (KB), MegaByte (MB), GigaByte (GB), TeraByte (TB). Il loro valore è il seguente:

1 KB	$= 2^{10} \times 1$ BYTE	$= 1024 \times 1$ BYTE	$\approx 1000 \times 1$ BYTE	$= 10^3$ BYTE
1 MB	$= 2^{10} \times 1$ KB	$= 1024 \times 1$ KB	$\approx 1000 \times 1$ KB	$\approx 10^6$ BYTE
1 GB	$= 2^{10} \times 1$ MB	$= 1024 \times 1$ MB	$\approx 1000 \times 1$ MB	$\approx 10^9$ BYTE
1 TB	$= 2^{10} \times 1$ GB	$= 1024 \times 1$ GB	$\approx 1000 \times 1$ GB	$\approx 10^{12}$ BYTE

Cenni sulla conversione di base

Molte informazioni sono costituite da un numerale associato a un nome; in questi casi, il numerale indica in modo preciso la quantità numerica di qualcosa, costituendo quindi un'importante parte dell'informazione. Ad esempio, nell'informazione “tre mele”, il numerale *tre* indica la quantità di mele. Per ovvie ragioni siamo abituati a pensare ai numeri in base 10, quella che quotidianamente utilizziamo per esprimerci e far di conto. Tuttavia, sebbene per ragionare con i numeri abbiamo bisogno di rappresentarli in qualche base, il concetto di numero è indipendente dalla base di rappresentazione. Inoltre, quando scriviamo un numero, non posizioniamo le cifre a caso, ma diamo un peso a ogni cifra a seconda della sua posizione. Questo tipo di rappresentazione prende il nome di *notazione posizionale* (e anche questa non dipende dalla base adottata). Nella notazione posizionale la posizione di una cifra in un numero indica il suo peso in potenze della base. Ad esempio, 123 in base 10 indica un numero composto da

- 3 unità, ovvero 3×10^0,
- 2 decine, ovvero 2×10^1,
- 1 centinaia, ovvero 1×10^2.

Quindi, 123 in base 10 indica il numero $3 \times 10^0 + 2 \times 10^1 + 1 \times 10^2 + \ = 123$. Similmente, 110010 in base 2, indicato anche come $[110010]_2$, è una rappresentazione del numero

$$[110010]_2 = 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 = 0 + 2 + 0 + 0 + 16 + 32 = 50 .$$

L'ultimo esempio dovrebbe chiarire come convertire un numero da base 2 a base 10: si moltiplica ogni cifra per il suo peso e quindi si somma il tutto. In modo simile, è possibile convertire in base 10 un numero espresso in una base qualsiasi B. Il peso della cifre si ottiene nel modo seguente:

- La cifra meno significativa ha peso B^0 (e quindi sarà 1 per tutte le basi).
- La prima cifra a sinistra della cifra meno significativa ha peso B^1.
- La seconda cifra a sinistra della cifra meno significativa ha peso B^2.
- …
- La n-esima prima cifra a sinistra della cifra meno significativa ha peso B^n .
- …

La conversione inversa, ovvero da base 10 a una base qualsiasi B, si ottiene dividendo ripetutamente il numero per B. I resti rappresentano cifre via via più significative. Chiaramente, il processo può essere terminato quando il quoziente della divisone è 0, in quanto continuare significherebbe aggiungere zeri a sinistra del numero (ottenendo quindi lo stesso numero). Ad esempio, le divisioni da eseguire per convertire 123 in base 2 sono le seguenti:

- 123 / 2 = 61 con resto 1
- 61 / 2 = 30 con resto 1
- 30 / 2 = 15 con resto 0
- 15 / 2 = 7 con resto 1
- 7 / 2 = 3 con resto 1
- 3 / 2 = 1 con resto 1
- 1 / 2 = 0 con resto 1

Come detto, continuare a dividere il quoziente produrrebbe un ulteriore 0 con resto 0, e pertanto ci fermiamo. La rappresentazione in base due di 123 è quindi $[1111011]_2$. Per verificare la correttezza della conversione possiamo convertire $[1111011]_2$ in base 10:

$$[1111011]_2 = 1\times 2^0 + 1\times 2^1 + 0\times 2^2 + 1\times 2^3 + 1\times 2^4 + 1\times 2^5 + 1\times 2^6 = 1+2+0+8+16+32+64 = 123$$

.

Avendo ottenuto 123, che è il numero da cui siamo partiti, abbiamo una buona confidenza che la conversione sia stata eseguita in modo corretto (è improbabile che abbiamo sbagliato per due volte e in modo tale che i risultati coincidano).
I due algoritmi, combinati, consentono di passare da una base qualsiasi a un'altra base qualsiasi. Infatti, volendo convertire un numero da base B a base B' , possiamo convertire il numero da base B a base 10, e quindi da base 10 a base B'.

Oltre alla base 2, basi degne di nota in informatica sono le basi 8 e 16. Per la base 16 le cifre decimali non sono sufficienti, e quindi generalmente si adottano le prime lettere dell'alfabeto (maiuscole o minuscole) per le cifre 10 (A), 11 (B), 12 (C), 13 (D), 14 (E) e 15 (F). A titolo esemplificativo, per convertire il numero $[123]_{10}$ in base 8 applichiamo l'algoritmo delle divisioni ripetute (si noti che in questo caso scrivere 123 è ambiguo, in quanto non si potrebbe distinguere fra $[123]_{10}$ e $[123]_8$):

- 123 / 8 = 15 con resto 3
- 15 / 8 = 1 con resto 7
- 1 / 8 = 0 con resto 1

Otteniamo quindi $[173]_8$. Verifichiamo la correttezza di quanto fatto convertendo $[173]_8$ in base 10 con l'algoritmo delle moltiplicazioni:

$$[173]_8 = 3\times 8^0 + 7\times 8^1 + 1\times 8^2 = 3+56+64 = [123]_{10} .$$

Rappresentiamo ora $[123]_{10}$ in base 16:

- 123 / 16 = 7 con resto 11 , ovvero B
- 7 / 16 = 0 con resto 7

Quindi, $[123]_{10} = [7B]_{16}$. Verifichiamo riportando il numero in base 10:

$$[7B]_{16} = 11\times 16^0 + 7\times 16^1 = 11+112 = [123]_{10} .$$

Per basi esprimibili come potenza l'una dell'altra è possibile passare dall'una all'altra base applicando algoritmi più semplici che descriviamo informalmente per mezzo di qualche esempio. Per convertire un numero da base 8 a base 2 convertiamo ogni cifra in base 2 utilizzando 3 cifre (perché $8 = 2^3$). Ad esempio, possiamo passare da $[173]_8$ a $[1111011]_2$ senza passare per $[123]_{10}$ nel modo seguente:

$$[173]_8 = 001.111.011 = [1111011]_2 .$$

Si noti che l'applicazione dell'algoritmo ha prodotto 001 per la cifra più significativa di $[173]_8$, ovvero 1, 111 per 7, e 011 per 3. La rappresentazione di $[173]_8$ in base 2 è quindi la concatenazione di queste 3 triple (dove gli zeri più a sinistra sono stati omessi per comodità).
Allo stesso modo, per passare da un numero in base 16 a un numero in base 2 convertiamo ogni cifra in base 2 utilizzando 4 cifre (perché $16 = 2^4$). Ad esempio, $[7B]_{16}$ sarà la concatenazione di 0111, ottenuto da 7, e 1011, ottenuto da B:

$$[7B]_{16} = 0111.1011 = [1111011]_2 .$$

La conversione inversa, ad esempio da base 2 a base 8, si esegue raggruppando le cifre della rappresentazione in base 2, da destra verso sinistra, in gruppi di 3 cifre (perché $8 = 2^3$). Quindi si converte ogni gruppetto in base 8. Ad esempio, per $[1111011]_2$ avremo i gruppetti 011, da cui otteniamo 3, 111, ovvero 7, e 1:

$$[1111011]_2 = 1.111.011 = [173]_8 .$$

Similmente, da base 2 a base 16 raggruppiamo in gruppi di 4 cifre (perché $16 = 2^4$).

$$[1111011]_2 = 111.1011 = [7B]_{16} .$$

Chiaramente non possiamo convertire in questo modo un numero da base 8 a base 16, in quanto 16 non è esprimibile come potenza di 8. Però, possiamo evitare di passare dalla base 10, preferendo la più comoda base 2.

Codifica dei numeri interi

I numeri interi positivi possono essere rappresentati con un certo numero di bit semplicemente convertendoli in binario. Ne consegue che se il numero di bit scelto per rappresentare i numeri interi è N, il massimo intero rappresentabile sarà $2^N - 1$ (in quanto il più piccolo è 0).
Per i numeri con segno dobbiamo distinguere fra positivi e negativi. Un approccio naive potrebbe essere convertire il numero nella sua rappresentazione binaria e riservare il primo bit alla rappresentazione del segno (visto che il segno può assumere due valori, positivo o negativo). In questo modo, con N bit potremo rappresentare i numeri da $-2^{N-1} - 1$ a $+2^{N-1} - 1$, per un totale di $2^N - 1$ numeri. La combinazione persa è quella per rappresentare lo zero, che in questo schema ha due codifiche, una positiva e una negativa. Questo è il principale inconveniente della rappresentazione naive, che quindi risulta essere poco utile.

Una rappresentazione più adeguata per i numeri negativi è quella in complemento a 2. Questa rappresentazione prevede di codificare i numeri positivi nel modo tradizionale, semplicemente convertendoli in base 2, mentre i numeri negativi vengono convertiti in binario dopo avergli sommato 2^N, dove N è il numero di bit usati nella rappresentazione. Ne consegue che il primo bit sarà sempre 0 per i numeri positivi e 1 per quelli negativi. Volendo codificare +123 in complemento a 2, utilizzando 8 bit, avremo la semplice conversione in base 2 (vedi sezione precedente), e quindi $\overline{[01111011]}_2$ (si noti che, per distinguere fra rappresentazione in base 2 e rappresentazione in complemento a 2, il pedice di quest'ultimo è stato arricchito con un trattino in alto). Diversamente, per –123, dovremo prima sommare 2^8 = 256 al numero negativo, ottenendo quindi 256 – 123 = 133. Quindi, convertire 133 in base 2 con l'algoritmo delle divisioni ripetute:

- 133 / 2 = 66 con resto 1
- 66 / 2 = 33 con resto 0
- 33 / 2 = 16 con resto 1
- 16 / 2 = 8 con resto 0
- 8 / 2 = 4 con resto 0
- 4 / 2 = 2 con resto 0
- 2 / 2 = 1 con resto 0
- 1 / 2 = 0 con resto 1

Otteniamo quindi $\overline{[10000101]}_2$. In modo equivalente, possiamo ottenere la rappresentazione in complemento a 2 di un numero negativo convertendo il suo opposto (il numero positivo con le stesse cifre) in binario, quindi invertendo tutte le sue cifre (0 diventa 1 e 1 diventa 0), e infine sommando 1 al numero ottenuto. Nel nostro caso, l'opposto di –123 è +123, che in base 2 si rappresenta come $[01111011]_2$. Invertendo ogni cifra otteniamo 10000100, a cui dobbiamo sommare 1. Otteniamo quindi $\overline{[10000101]}_2$, che è ciò che ci aspettavamo.

Codifica dei numeri non interi

I numeri non interi sono caratterizzati da una parte intera e una frazionaria, che è detta decimale se il numero è espresso in base 10. Come per la parte intera, anche le cifre della parte decimale hanno un peso determinato dalla posizione in cui compaiono. In particolare, per una base B, la cifra che segue la virgola avrà peso B^{-1}, la successiva B^{-2}, e così via la n-esima cifra avrà peso B^{-n}. In modo equivalente, possiamo dire che la cifra che segue la virgola ha peso $\frac{1}{B^1}$, la successiva $\frac{1}{B^2}$, e così via. Ad esempio, $[0.123]_{10}$ rappresenta il numero formato da

- 0 unità, ovvero 0×10^0,
- 1 decimo, ovvero 1×10^{-1},

- 2 centesimi, ovvero 2×10^{-2},
- 3 millesimi, ovvero 3×10^{-3}.

L'esempio suggerisce un semplice algoritmo per la conversione in base 10 di un numero non intero rappresentato in una base qualsiasi B: moltiplicare ogni cifra per il suo peso, ottenuto come potenza di B con esponente 0 per la cifra a sinistra della virgola, crescente a sinistra e decrescente a destra, e quindi sommare il tutto. Ad esempio, il numero $[0.1101]_2$ rappresenta

$$[0.1101]_2 = 0\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4} = \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16} = [0.8125]_{10} .$$

La parte decimale di un numero frazionario (espresso in base 10) può essere convertita in base B applicando moltiplicazioni ripetute alla parte decimale e quindi prendendo di volta in volta la parte intera del prodotto come cifra in base B. Ogni moltiplicazione aggiunge una cifra di precisione alla conversione. Se si ottiene un prodotto con parte decimale nulla il processo può essere interrotto in quanto procedendo aggiungeremmo ulteriori zeri a destra dell'ultima cifra significativa (e quindi otterremmo lo stesso numero). Ad esempio, convertendo $[0.8125]_{10}$ in base 2 otteniamo:

- 0.8125 × 2 = 1.625 con parte intera 1
- 0.625 × 2 = 1.25 con parte intera 1
- 0.25 × 2 = 0.5 con parte intera 0
- 0.5 × 2 = 1.0 con parte intera 1

L'ultimo prodotto ha parte decimale nulla e quindi l'algoritmo termina. Per convertire $[123.8125]_{10}$ in base 2 convertiamo separatamente la parte intera e la parte decimale, ottenendo $[1111011]_2$ e $[0.1101]_2$, che sommati danno $[1111011.1101]_2$.
Come per i numeri interi, anche per i numeri con la virgola possiamo applicare l'algoritmo veloce per basi che sono potenze l'una dell'altra. Ad esempio, per convertire $[1111011.1101]_2$ in base 16 raggruppiamo le cifre a gruppi di 4, sia a sinistra che a destra della virgola, eventualmente aggiungendo zeri per completare i gruppetti. Quindi abbiamo $[1111011.1101]_2 = [7B.D]_{16}$, dove D è stato ottenuto dal gruppetto 1101 che segue la virgola. In modo simile, convertendo in base 8 dovremo raggruppare le cifre 3 alla volta. Avremo quindi $[173.64]_8$, dove il 6 è ottenuto dal gruppetto 110, e il 4 da 100 (all'ultimo 1 sono stati aggiunti due zeri per formare il gruppetto).

Per la rappresentazione dei numeri decimali in un computer possiamo usare la rappresentazione a virgola fissa o a virgola mobile. Con la notazione a virgola fissa viene destinato un certo numero di bit a rappresentare la parte intera del numero e un altro per la parte decimale. Ad esempio, destinando 8 bit alla parte intera e 8 alla parte frazionaria, per $[123.8125]_{10}$ avremo $[01111011.11010000]_2$.

I numeri in virgola mobile, invece, sono costituiti da mantissa ed esponente. Il numero rappresentato in notazione a virgola mobile si ottiene moltiplicando la mantissa per la base elevata l'esponente. In genere la mantissa è un numero con parte intera nulla. Ad esempio, $[0.8125; +2]_{10}$, dove 0.8125 è la mantissa e +2 l'esponente, rappresenta il numero

$$[0.8125; +2]_{10} = 0.8125 \times 10^2 = [81.25]_{10} .$$

Si noti che poiché l'esponente è applicato alla base, la moltiplicazione ha l'effetto di spostare la virgola a sinistra o a destra, a seconda del segno, di tante posizioni quante indicate dall'esponente.
In modo simile faremo per un numero rappresentato in virgola mobile in base 2. Ad esempio, $[0.1101; +10]_2$ rappresenta

$$[0.1101; +10]_2 = [0.1101]_2 \times 2^{[10]} = [0.1101]_2 \times 2^2 = [11.01]_2 =$$
$$= 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 2 + 1 + \frac{1}{4} = [3.25]_{10} .$$

La conversione in virgola mobile e base B di un numero non intero espresso in base 10 può essere ottenuta nel modo seguente:

- Se il numero in base 10 è espresso in virgola mobile, allora determiniamo il numero rappresentato (spostando la virgola come indicato dall'esponente).
- Convertiamo il numero in base B usando l'algoritmo delle divisioni successive.
- Spostiamo la virgola a sinistra o a destra in modo da avere la mantissa nella forma desiderata.
- L'esponente è il numero di posizioni che la virgola è stata spostata (in positivo o in negativo).

Ad esempio, volendo convertire $[13.125]_{10}$ in notazione a virgola mobile e base 2, determiniamo prima la sua rappresentazione in base 2:

$[13]_{10} = [1101]_2$, a cui dobbiamo sommare $[0.125]_{10} = [0.001]_2$, ottenendo $[1101.001]_2$.

A questo punto convertiamo $[1101.001]_2$ in notazione a virgola mobile. Dobbiamo spostare la virgola di 4 posizioni verso sinistra, quindi l'esponente sarà +4 (da convertire anch'esso in base 2):

$[1101.001]_2 = [0.1101001; +100]_2$.

Anche per i numeri a virgola mobile bisogna stabilire quanti bit riservare alla mantissa e quanti all'esponente. Assegnando 8 bit alla mantissa e 8 all'esponente (per l'esponente dobbiamo considerare anche il segno), il numero più grande che possiamo rappresentare è

$$\left[0.11111111;+1111111\right]_2 = 0.99609375 \times 2^{127} \quad 1.69 \times 10^{38}$$

Naturalmente non possiamo rappresentare tutti i numeri non interi fra 0 e 1.69×10^{38}, che sono in numero infinito. Quello che consente di raggiungere numeri così elevati è la perdita di precisione.

Codifica dei caratteri

I caratteri, come ogni altra informazione, possono essere codificati da un gruppo di bit. Ad esempio, volendo distinguere fra cifre (0, …, 9), lettere minuscole e maiuscole dell'alfabeto inglese (a, …, z, A, …, Z) e alcuni simboli di punteggiatura (:, ;, ., nuova linea, e pochi altri), abbiamo bisogno di rappresentare circa 120 simboli. Lo standard ASCII adotta quindi una codifica a 7 bit (128 combinazioni). Ad ogni combinazione dei 7 bit corrisponde un carattere diverso. Ad esempio:

- 10000001 codifica il carattere A (maiuscolo);
- 10000010 codifica il carattere B (maiuscolo);
- 11000001 codifica il carattere a (minuscolo);
- 11000010 codifica il carattere b (minuscolo).

Come detto in precedenza, in informatica si usano principalmente multipli di byte, quindi un carattere ASCII generalmente viene memorizzato in un intero byte, fissando il bit più significativo a 0. Ad esempio, A sarà memorizzato come il byte 01000001 e B come 01000010. Sembra naturale, per tanto, che sia stato definito un nuovo standard, detto ASCII esteso, nel quale il bit più significativo non è fissato a 0. Lo standard ASCII esteso consente, quindi, di rappresentare simboli aggiuntivi. Di seguito è riportata la tabella di corrispondenza completa dello standard ASCII (non esteso).

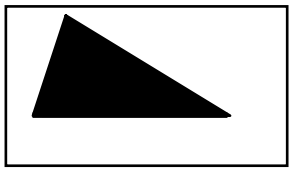
00000000	Null	00100000	Spc	01000000	@	01100000	`
00000001	Start of heading	00100001	!	01000001	A	01100001	a
00000010	Start of text	00100010	"	01000010	B	01100010	b
00000011	End of text	00100011	#	01000011	C	01100011	c
00000100	End of transmit	00100100	$	01000100	D	01100100	d
00000101	Enquiry	00100101	%	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

Recentemente è stato introdotto un nuovo standard per consentire di rappresentare i caratteri di altri linguaggi (arabo, cirillico, cinese, …). Lo standard prende il nome di UNICODE e usa 16 bit.

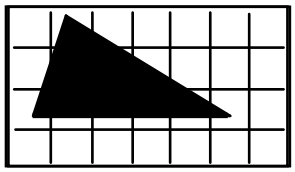
Codifica di dati multimediali

Un'immagine digitale è costituita da una griglia di punti detti pixel. Tanti più pixel compongono l'immagine, tanto più quest'ultima sarà fedele all'originale. Rappresentando in modo opportuno ogni pixel è possibile codificare un'immagine per essere elaborata da un calcolatore elettronico. Ad esempio, per un'immagine in bianco e nero, ogni pixel può essere rappresentato da un bit (associando 0 al nero e 1 al bianco).

Consideriamo la seguente immagine in bianco e nero.



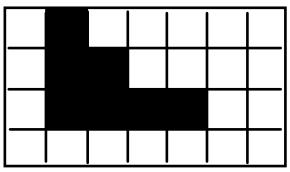
Per digitalizzare quest'immagine dobbiamo stabilire la dimensione della griglia in pixel, ad esempio 7 in orizzontale e 4 in verticale.



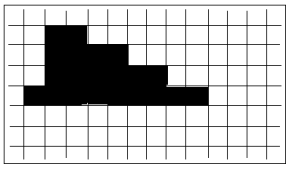
Possiamo quindi determinare il valore di ogni pixel: sarà 0 se *prevalentemente* nero, 1 altrimenti. Per il nostro esempio otteniamo:

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

La griglia rappresenta l'immagine digitale



Si noti che è stata persa dell'informazione per via della discretizzazione eseguita. Come già detto, possiamo attenuare la perdita d'informazione aumentando il numero di pixel della griglia. Ad esempio, con una griglia 14×18 otteniamo



All'aumentare del numero di pixel aumenterà lo spazio necessario per la memorizzazione dell'immagine. Ad esempio, per memorizzare un'immagine in bianco e nero di dimensione 25×10 pixel, avremo bisogno di

$$25 \times 10 \times 1\,bit = 250\,bit = 250/8\,byte = 32\,byte$$.

Si noti che 250/8 = 31.25 e pertanto dobbiamo arrotondare per eccesso a 32 byte.
In modo simile, un'immagine a 256 toni di grigio avrà bisogno di 1 byte (8 bit) per ogni pixel. Quindi, per un'immagine a 256 toni di grigio di dimensione 25×10 pixel, avremo bisogno di

$$25 \times 10 \times 1\,byte = 250\,byte$$.

I colori vengono in genere ottenuti come una combinazione di rosso, verde e blu (secondo la codifica RGB – red, green, blue). Se assegniamo un byte a ogni colore fondamentale, per ogni pixel avremo bisogno di 3 byte. Questo tipo di immagini viene comunemente detta a 24 bit. Quindi, per un'immagine a colori a 24 bit di dimensione 25×10 pixel, avremo bisogno di

$$25 \times 10 \times 3\,byte = 750\,byte$$.

I video possono essere codificati come sequenze di immagini (sebbene nella pratica vengano usate codifiche più efficienti sia per le immagini che per i video).

I suoni vengono in genere campionati, ovvero la loro onda sonora è misurata a intervalli di tempo regolari. Questo implica che, come per le immagini, la codifica digitale di un suono comporta una perdita di informazione. La perdita è tanto maggiore quanto maggiore è l'intervallo di campionamento. Lo standard adottato dai CD musicali prevede 44000 campionamenti al secondo e 16 bit per campione. Altri formati comuni sono WAV, MP3 e WMA.
Il formato MIDI, invece, codifica uno spartito musicale che poi viene interpretato dal computer. Un MIDI richiede generalmente pochi KB per essere memorizzato (contro i MB del formato MP3), ma può codificare solo un numero ristretto di strumenti musicali.