CAPITOLO

9

Algoritmi, numeri macchina e calcolabilità

Un computer esegue miliardi di istruzioni al secondo, ma non può fare nulla che un algoritmo non sappia descrivere. (testo della redazione)

OBIETTIVI

1. Progettare algoritmi per approssimazioni successive e per selezione dei casi, con un criterio di arresto.
2. Eseguire e confrontare la ricerca sequenziale, la ricerca binaria e l'ordinamento per selezione, contando le operazioni.
3. Stimare un numero con una simulazione Monte Carlo e valutarne i limiti.
4. Spiegare come un computer rappresenta i numeri reali in virgola mobile e riconoscere errori di arrotondamento e traboccamento.
5. Descrivere la compressione dei dati multimediali, con e senza perdita.
6. Eseguire semplici macchine di Turing e discutere i concetti di funzione calcolabile e di problema non calcolabile.

INDICE DEL CAPITOLO

9.1 Strategie algoritmiche 2
9.2 Cercare e ordinare 6
9.3 Simulare il caso: il metodo Monte Carlo 11
9.4 I numeri reali nel computer 12
9.5 Dati multimediali e compressione 15
9.6 Che cosa si può calcolare 17

Matematica 2 · Parte VI · Elementi di informatica · versione 1.0
Capitolo 9: Algoritmi, numeri macchina e calcolabilità · pagine 221–246 del volume completo · estratto per lo studio

Quando una calcolatrice mostra la radice quadrata di 10, quando un motore di ricerca trova in un istante una parola tra miliardi di pagine, quando il telefono salva una foto, dietro c'è sempre un algoritmo: una sequenza finita di istruzioni precise che un esecutore, anche privo di intelligenza, può seguire senza dover decidere nulla. In *Matematica 1* hai imparato a descrivere un algoritmo e a controllarlo con una tabella di traccia. In questo capitolo usiamo quegli strumenti per rispondere a tre domande più profonde.

La prima: **come si progetta** un buon algoritmo? Vedremo alcune strategie ricorrenti, applicate a problemi che conosci da questo volume: le radici quadrate, un'equazione di terzo grado, i sistemi lineari, la ricerca in un elenco. La seconda: **con quali numeri** lavora davvero un computer? Scopriremo che non sono i numeri reali del capitolo 1, ma un insieme finito di «numeri macchina», e che per questo un computer può sbagliare un calcolo semplicissimo come 0,1 + 0,2. La terza: **che cosa si può calcolare**? Con la macchina di Turing daremo una definizione precisa di «algoritmo» e capiremo perché esistono problemi che nessun computer, per quanto potente, saprà mai risolvere.

9.1 Strategie algoritmiche

9.1.1 Che cosa sappiamo già

Ricordiamo in breve (vedi *Matematica 1*). Un algoritmo è una sequenza finita di istruzioni non ambigue che, a partire da dati di ingresso, produce dati di uscita in un numero finito di passi. Ogni algoritmo si costruisce con tre strutture: la **sequenza** (istruzioni eseguite una dopo l'altra), la **selezione** (si sceglie che cosa fare in base a una condizione) e l'**iterazione** (un gruppo di istruzioni si ripete finché una condizione è vera). Lo si descrive con un diagramma di flusso o con lo pseudocodice, e lo si controlla con la **tabella di traccia**, che registra il valore delle variabili dopo ogni passo.

DA RICORDARE

Nello pseudocodice di questo libro: **LEGGI** e **SCRIVI** per i dati di ingresso e di uscita; $x \leftarrow e$ per l'**assegnazione** (la variabile x prende il valore dell'espressione e); **SE … ALLORA … ALTRIMENTI … FINE SE** per la selezione; **MENTRE … ESEGUI … FINE MENTRE** e **PER … DA … A … FINE PER** per l'iterazione. Le righe rientrate appartengono al blocco che le precede.

9.1.2 Approssimazioni successive: il metodo di Erone

Molti problemi non si risolvono con una formula che dà subito il risultato esatto, ma con una strategia di **approssimazioni successive**: si parte da un valore approssimato e lo si migliora ripetendo sempre lo stesso calcolo, finché l'approssimazione è abbastanza buona. Nel capitolo 1 hai incontrato il metodo di Erone per $\sqrt{3}$. In generale, per calcolare $\sqrt{a}$ con $a > 0$: se x è un'approssimazione positiva di $\sqrt{a}$, il numero $\sqrt{a}$ è compreso tra x e $a : x$, e la loro media

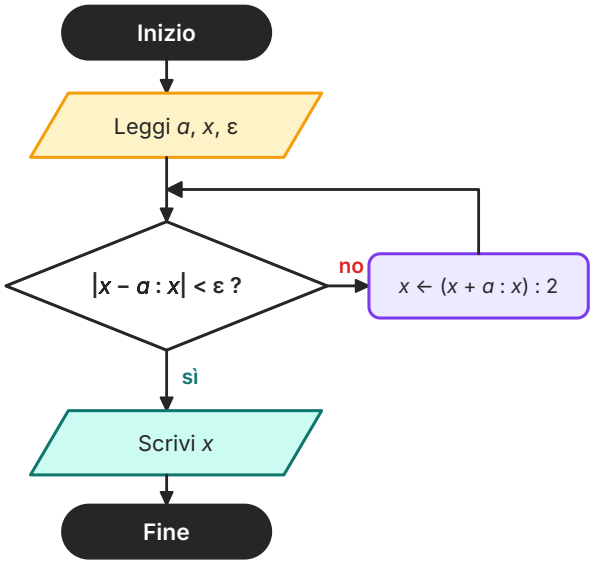
$$x \leftarrow \frac{x + a : x}{2}$$

è un'approssimazione migliore. Per trasformare quest'idea in un algoritmo manca un ingrediente: un **criterio di arresto**, cioè una condizione che dica quando fermarsi. Poiché $\sqrt{a}$ sta sempre tra x e $a : x$, l'errore che si commette scrivendo x al posto di $\sqrt{a}$ è minore della distanza tra i due numeri. Scelta allora una tolleranza $\varepsilon > 0$ (per esempio $\varepsilon = 10^{-6}$), ci si ferma appena $|x - a : x| < \varepsilon$: a quel punto l'errore è sicuramente minore di ε.

Pseudocodice Il metodo di Erone per la radice quadrata.

```
LEGGI a, x, ε   (a > 0; x > 0 approssimazione iniziale; ε > 0 tolleranza)
MENTRE |x − a : x| ≥ ε ESEGUI
    x ← (x + a : x) : 2
FINE MENTRE
SCRIVI x
```

La Figura 9.1 mostra lo stesso algoritmo come diagramma di flusso, con la traccia per $a = 10$, partendo da $x = 3$. L'ultima colonna è la cosa più interessante: la differenza tra x e $a : x$ passa da circa 0,3 a circa 0,009, poi a circa 0,000006, poi a meno di un miliardesimo. A ogni passo il numero di cifre esatte **circa raddoppia**: è questo che rende il metodo di Erone così rapido.



Traccia con $a = 10$, $x = 3$, $\varepsilon = 10^{-6}$

passo	x	$a : x$	$\lvert x - a : x\rvert$
0	3,0000000	3,3333333	0,3333333
1	3,1666667	3,1578947	0,0087719
2	3,1622807	3,1622746	$6{,}1 \cdot 10^{-6}$
3	3,1622777	3,1622777	$2{,}9 \cdot 10^{-12}$

Al passo 3 la differenza è minore di ε: l'algoritmo scrive $x \approx 3{,}1622777$.

Figura 9.1 Il metodo di Erone come diagramma di flusso e la sua tabella di traccia per √10. Dopo tre aggiornamenti la differenza $|x - a : x|$ scende sotto $\varepsilon = 10^{-6}$ e l'algoritmo si ferma; il valore esatto è $\sqrt{10} = 3{,}16227766\ldots$

ESEMPIO SVOLTO

Calcola $\sqrt{5}$ con il metodo di Erone, partendo da $x = 2$ con tolleranza $\varepsilon = 10^{-3}$, lavorando con le frazioni.

Passo 0. $x = 2$, $a : x = \dfrac{5}{2}$; la differenza è $\dfrac{1}{2} \geq 10^{-3}$: si continua.

Passo 1. $x = \dfrac{2 + 5/2}{2} = \dfrac{9}{4} = 2{,}25$; $a : x = \dfrac{20}{9} \approx 2{,}2222$. Differenza: $\dfrac{9}{4} - \dfrac{20}{9} = \dfrac{81 - 80}{36} = \dfrac{1}{36} \approx 0{,}028$: si continua.

Passo 2. $x = \dfrac{9/4 + 20/9}{2} = \dfrac{161}{72} \approx 2{,}23611$; $a : x = \dfrac{360}{161} \approx 2{,}23602$. Differenza: $\dfrac{161^2 - 360 \cdot 72}{72 \cdot 161} = \dfrac{25\,921 - 25\,920}{11\,592} = \dfrac{1}{11\,592} \approx 0{,}000086 < 10^{-3}$: ci si ferma.

L'algoritmo scrive $x = 161/72 \approx 2{,}2361$, con un errore minore di 0,0001. Infatti $\sqrt{5} = 2{,}236067\ldots$

9.1.3 La bisezione per un'equazione di terzo grado

Il metodo di Erone funziona solo per le radici. Per un'equazione qualsiasi si può usare il metodo di bisezione, che in *Matematica 1* hai applicato a √2: se un polinomio $f(x)$ assume valori di segno opposto agli estremi di un intervallo, l'equazione $f(x) = 0$ ha (almeno) una soluzione all'interno;

si calcola f nel punto medio e si tiene la metà dell'intervallo in cui il cambio di segno persiste. Le equazioni di terzo grado si risolvono in modo esatto solo in casi particolari, e per l'equazione $x^3 + x - 3 = 0$, per esempio, non conosci alcun metodo che dia la soluzione con una formula; si dimostra anzi che la soluzione non è un numero razionale. Poiché $f(1) = -1 < 0$ e $f(2) = 7 > 0$, una soluzione sta in $[1; 2]$, e la bisezione la stringa sempre di più (Tabella 9.1).

Tabella 9.1 Cinque passi di bisezione per $x^3 + x - 3 = 0$: $f(m)$ è il valore del polinomio nel punto medio m.

passo	a	b	m	$f(m)$	segno	nuovo intervallo
1	1	2	1,5	1,875	+	[1; 1,5]
2	1	1,5	1,25	0,203…	+	[1; 1,25]
3	1	1,25	1,125	−0,451…	−	[1,125; 1,25]
4	1,125	1,25	1,1875	−0,138…	−	[1,1875; 1,25]
5	1,1875	1,25	1,21875	0,029…	+	[1,1875; 1,21875]

Dopo cinque passi sappiamo che la soluzione sta in $[1{,}1875; 1{,}21875]$, un intervallo di ampiezza $\frac{1}{32} \approx 0{,}03$ (il valore è circa 1,2134). A ogni passo l'ampiezza si dimezza: dopo k passi è $1 : 2^k$ dell'ampiezza iniziale. Poiché $2^{10} = 1024$, servono circa 10 passi per guadagnare tre cifre decimali: la bisezione è sicura (funziona per qualunque equazione con un cambio di segno) ma molto più lenta di Erone, che raddoppia le cifre a ogni passo. Scegliere tra un metodo generale e uno più rapido ma specializzato è una decisione tipica di chi progetta algoritmi.

9.1.4 Selezione: decidere in base ai casi

Una seconda strategia è la **selezione** dei casi: l'algoritmo esamina i dati e sceglie la strada adatta. Il primo esempio lo conosci da *Matematica 1*: l'equazione di primo grado $ax = b$. Se $a \neq 0$ ha l'unica soluzione $b : a$; se $a = 0$ e $b \neq 0$ è impossibile; se $a = b = 0$ è indeterminata, perché ogni numero reale è soluzione. Trascritto in pseudocodice, il procedimento diventa un algoritmo che chiunque, anche una macchina, può eseguire.

Pseudocodice Le soluzioni dell'equazione di primo grado $ax = b$.

```
LEGGI a, b
SE a ≠ 0 ALLORA
    SCRIVI «una soluzione»: b : a
ALTRIMENTI SE b ≠ 0 ALLORA
    SCRIVI «impossibile»
ALTRIMENTI
    SCRIVI «indeterminata: ogni numero reale è soluzione»
FINE SE
```

Nota che il secondo SE viene esaminato solo quando $a = 0$: l'ordine delle domande fa parte dell'algoritmo. Allo stesso modo si scrive l'algoritmo per un sistema lineare di due equazioni in due incognite in forma normale (capitolo 6). Si calcola il determinante $D = ab' - ba'$: se $D \neq 0$ il sistema è determinato e la regola di Cramer dà $x = D_x : D$, $y = D_y : D$; se $D = 0$ l'algoritmo non può fermarsi, perché D da solo non distingue i due casi rimanenti, e deve guardare anche D_x e D_y: se sono entrambi nulli il sistema è indeterminato, altrimenti è impossibile (in ogni equazione almeno uno dei coefficienti delle incognite è diverso da zero, come nel capitolo 6). Per i sistemi più grandi l'eliminazione di Gauss del capitolo 6 è l'algoritmo che usano davvero i computer: una sequenza fissa di passi che riduce un'incognita alla volta, senza bisogno di intuizione.

9.2 Cercare e ordinare

9.2.1 La ricerca sequenziale

Cercare un dato in un elenco è uno dei compiti più frequenti di un computer: un nome in una rubrica, un prodotto in un magazzino, una parola in un dizionario. Il metodo più semplice è la **ricerca sequenziale**: si confronta il dato cercato con il primo elemento, poi con il secondo, e così via, finché lo si trova o l'elenco finisce. Funziona sempre, anche se l'elenco è in disordine; ma nel caso peggiore (il dato è l'ultimo, o non c'è) richiede tanti confronti quanti sono gli elementi: n confronti per un elenco di n elementi.

9.2.2 La ricerca binaria

Se l'elenco è **ordinato** si può fare molto meglio, come fa chiunque cerchi una parola nel vocabolario: lo si apre a metà e si decide se proseguire nella prima o nella seconda metà. È la **ricerca binaria** (o dicotomica). Si confronta il dato con l'elemento centrale dell'elenco: se è uguale, la

ricerca è finita; se il dato è maggiore, si scarta la prima metà; se è minore, la seconda. Poi si ripete sulla metà rimasta. È la stessa idea della bisezione: a ogni passo l'intervallo in cui cercare si dimezza.

Pseudocodice La ricerca binaria in un elenco ordinato.

```
LEGGI l'elenco ordinato E1, E2, ..., En e il dato d
i ← 1;   j ← n;   trovato ← falso
MENTRE i ≤ j e trovato = falso ESEGUI
    c ← parte intera di (i + j) : 2
    SE Ec = d ALLORA trovato ← vero
    ALTRIMENTI SE Ec < d ALLORA i ← c + 1
    ALTRIMENTI j ← c − 1
    FINE SE
FINE MENTRE
SE trovato = vero ALLORA SCRIVI c ALTRIMENTI SCRIVI «non presente» FINE SE
```

Nella Figura 9.2 l'algoritmo cerca 63 in un elenco ordinato di 16 numeri. Al primo passo l'elemento centrale (posto 8) è 34, minore di 63: la metà sinistra viene scartata tutta insieme, senza guardarla. Al secondo passo restano 8 elementi, al terzo 4, al quarto 1, ed è quello cercato. Bastano 4 confronti; una ricerca sequenziale ne avrebbe fatti 13.

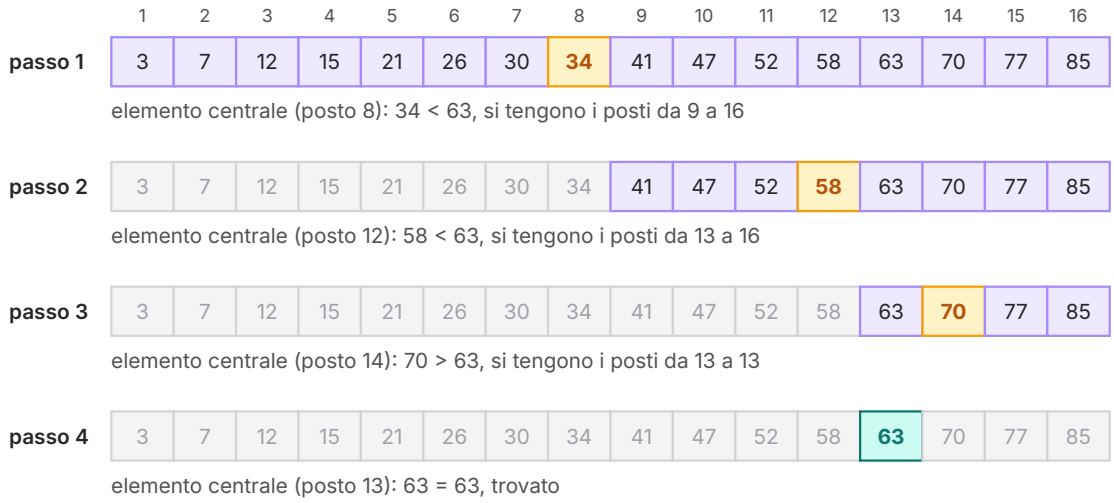


Figura 9.2 La ricerca binaria di 63 in un elenco ordinato di 16 numeri. In viola gli elementi ancora possibili, in grigio quelli scartati; in arancio l'elemento centrale confrontato con 63, in verde quello trovato.

Quanti confronti servono, al massimo? Ogni confronto con esito negativo lascia al più metà degli elementi: con $n = 16$ si passa a 8, 4, 2, 1, e l'ultimo elemento richiede un quinto confronto. In generale il numero massimo di confronti è il numero di volte in cui si può dimezzare n prima di arrivare a 1, più uno. Questo numero è legato al **logaritmo in base 2** di n, scritto $\log_2 n$: l'esponente a cui elevare 2 per ottenere n (per esempio $\log_2 16 = 4$ perché $2^4 = 16$, e $\log_2 1000 \approx 9{,}97$ perché $2^{10} = 1024$ supera di poco 1000). Lo studierai a fondo negli anni successivi; qui basta l'idea: la ricerca binaria richiede al più circa $\log_2 n$ confronti, per la precisione la parte intera di $\log_2 n$ più uno. Per un elenco di un milione di nomi sono 20 confronti invece di un milione.

9.2.3 Ordinare: l'ordinamento per selezione

La ricerca binaria richiede un elenco ordinato: ma come si ordina un elenco? Uno dei metodi più semplici è l'**ordinamento per selezione**. Si cerca il più piccolo di tutti gli elementi e lo si scambia con il primo; poi si cerca il più piccolo tra gli elementi dal secondo in poi e lo si scambia con il secondo; e così via, finché restano da sistemare due elementi soltanto.

Pseudocodice L'ordinamento per selezione.

```
LEGGI n e l'elenco E_1, E_2, ..., E_n
PER i DA 1 A n − 1
    k ← i    (posto del minimo trovato finora)
    PER j DA i + 1 A n
        SE E_j < E_k ALLORA k ← j FINE SE
    FINE PER
    scambia E_i ed E_k
FINE PER
SCRIVI l'elenco
```

ESEMPIO SVOLTO

Ordina l'elenco 8, 3, 9, 1, 5 con l'ordinamento per selezione e conta i confronti.

Passata 1 ($i = 1$): si confronta il minimo provvisorio con gli altri 4 elementi (4 confronti); il minimo è 1, al quarto posto. Scambio con il primo: **1**, 3, 9, 8, 5.

Passata 2 ($i = 2$): tra 3, 9, 8, 5 il minimo è 3, già al suo posto (3 confronti): 1, **3**, 9, 8, 5.

Passata 3 ($i = 3$): tra 9, 8, 5 il minimo è 5 (2 confronti); scambio con 9: 1, 3, **5**, 8, 9.

Passata 4 ($i = 4$): tra 8 e 9 il minimo è 8 (1 confronto): 1, 3, 5, **8**, 9. L'ultimo elemento è necessariamente il massimo.

Confronti: $4 + 3 + 2 + 1 = 10$, cioè $5 \cdot 4 : 2$.

L'esempio si generalizza: con n elementi la prima passata fa $n - 1$ confronti, la seconda $n - 2$, fino all'ultima che ne fa 1. Il totale è (con il trucco di Gauss di *Matematica 1*, che somma la sequenza con se stessa rovesciata)

$$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$$

e non dipende da come sono disposti i dati: anche un elenco già ordinato richiede tutti i confronti. Esistono algoritmi di ordinamento più rapidi, che studierai se approfondirai l'informatica; quello per selezione è il più facile da capire e da controllare.

9.2.4 Contare i passi: la complessità

Abbiamo misurato ciascun algoritmo con il numero di operazioni che compie in funzione della dimensione n dei dati. Questa funzione descrive la **complessità** dell'algoritmo, cioè come cre-

sce il suo costo quando i dati crescono. La ricerca sequenziale ha una crescita **lineare** (circa n confronti); l'ordinamento per selezione ha una crescita **quadratica** (circa $n^2 : 2$); la ricerca binaria ha una crescita **logaritmica** (circa $\log_2 n$). Per pochi dati la differenza non si nota; per molti dati è enorme (Tabella 9.2 e Figura 9.3).

Tabella 9.2 Numero di confronti (nel caso peggiore) di tre algoritmi, al crescere del numero n di elementi.

n	ricerca binaria	ricerca sequenziale	ordinamento per selezione
10	4	10	45
100	7	100	4 950
1 000	10	1 000	499 500
1 000 000	20	1 000 000	499 999 500 000

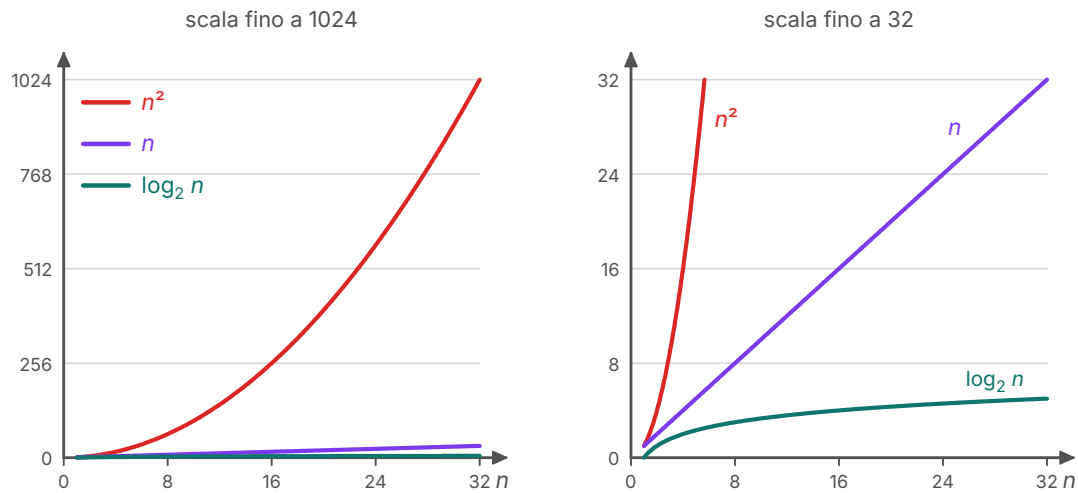


Figura 9.3 Confronto tra crescita quadratica (n^2), lineare (n) e logaritmica ($\log_2 n$) per n da 1 a 32. A sinistra, in scala fino a 1024, le curve di n e di $\log_2 n$ si confondono quasi con l'asse; a destra la stessa scala ingrandita fino a 32, dove n^2 esce presto dal grafico.

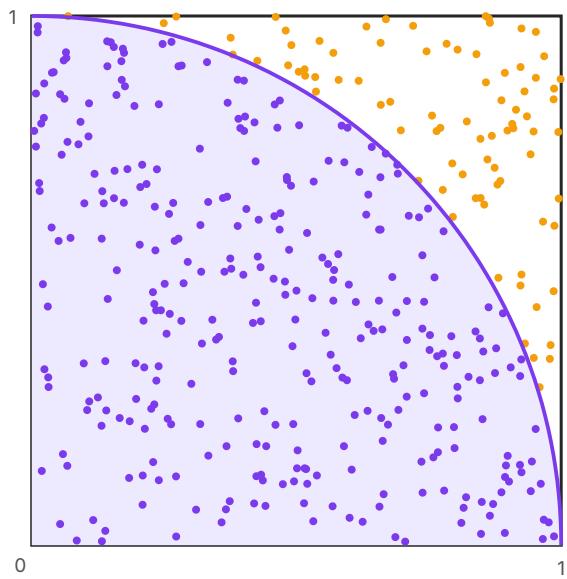
Immaginiamo un computer che esegua un miliardo di confronti al secondo. Ordinare per selezione un milione di dati richiede circa $5 \cdot 10^{11}$ confronti, cioè circa 500 secondi: più di otto minuti. Cercare un dato in quel milione con la ricerca binaria richiede 20 confronti, cioè 20 miliardesimi di secondo. Per questo, quando i dati sono molti, la scelta dell'algoritmo conta più della velocità della macchina: un computer mille volte più veloce non compensa un algoritmo quadratico al posto di uno logaritmico.

9.3 Simulare il caso: il metodo Monte Carlo

Gli algoritmi visti finora sono **deterministici**: con gli stessi dati danno sempre lo stesso risultato. Esiste però una famiglia di algoritmi che usa di proposito il caso. Consideriamo il quadrato di lato 1 con un vertice nell'origine e, al suo interno, il quarto di cerchio di centro l'origine e raggio 1 (Figura 9.4). Il quadrato ha area 1, il quarto di cerchio ha area π : 4. Se scegliamo un punto «a caso» nel quadrato, in modo che ogni zona abbia una probabilità proporzionale alla sua area, la probabilità che cada nel quarto di cerchio è il rapporto tra le aree: π : 4 ≈ 0,785.

Ora entra in gioco il legame tra probabilità e frequenza del capitolo 8: ripetendo l'esperimento molte volte, la frequenza relativa dei punti che cadono nel quarto di cerchio tende ad avvicinarsi alla probabilità. Il **metodo Monte Carlo** sfrutta proprio questo: si generano N punti casuali $(x; y)$ con $0 \le x \le 1$ e $0 \le y \le 1$, si contano i K punti per cui $x^2 + y^2 \le 1$ (cioè la cui distanza dall'origine non supera 1), e si stima

$$\pi \approx 4 \cdot \frac{K}{N}$$



Simulazione con 400 punti

- dentro il quarto di cerchio: 316
- fuori: 84

stima: π ≈ 4 · 316 : 400 = 3,16

Con più punti (stessa simulazione):

1 000 punti	π ≈ 3,112
10 000 punti	π ≈ 3,1432
100 000 punti	π ≈ 3,14304
1 000 000 punti	π ≈ 3,141152

Figura 9.4 Stima di π con il metodo Monte Carlo (simulazione al computer). A sinistra i primi 400 punti casuali: 316 cadono nel quarto di cerchio. A destra le stime ottenute proseguendo la stessa simulazione con più punti.

Nella simulazione della Figura 9.4, con 400 punti, 316 cadono dentro e la stima è 4 · 316 : 400 = 3,16. Proseguendo, con un milione di punti si ottiene 3,141152, che coincide con π = 3,14159... solo nelle prime tre cifre. Il metodo ha due limiti. Il primo: è **lento**. L'errore non

diminuisce in modo regolare e, in media, per guadagnare una cifra decimale servono circa cento volte più punti. Il secondo: il risultato è **casuale**. Ripetendo la simulazione con altri punti si ottiene un'altra stima, e nessuna stima è garantita: si può solo dire che è *molto probabile* che l'errore sia piccolo. Per calcolare π esistono algoritmi deterministici enormemente più efficienti; la forza del Monte Carlo è un'altra: funziona anche per problemi così complicati che nessuna formula li risolve, basta saperli simulare.

COLLEGAMENTI

Il metodo prese il nome dal celebre casinò negli anni Quaranta del Novecento, quando fu usato da Stanislaw Ulam, John von Neumann e altri scienziati per calcoli di fisica nucleare che non si sapevano risolvere con le formule; il primo articolo che lo descrive con questo nome è del 1949. Oggi le simulazioni Monte Carlo si usano in fisica (il moto di molte particelle), in ingegneria (l'affidabilità di una struttura soggetta a carichi casuali), in finanza e nella grafica al computer, dove simulano il percorso dei raggi di luce in una scena.

Con il computer

In un foglio di calcolo la funzione CASUALE() restituisce un numero casuale tra 0 e 1. Scrivi =CASUALE() nelle celle A2 e B2, e in C2 la formula =SE(A2^2+B2^2<=1;1;0), che vale 1 se il punto cade nel quarto di cerchio. Copia le tre formule fino alla riga 1001: avrai 1000 punti. In una cella a parte, =4*MEDIA(C2:C1001) è la stima di π. Ogni volta che il foglio viene ricalcolato (per esempio premendo F9) i punti cambiano e cambia la stima: osservare quanto oscilla è il modo migliore per capire i limiti del metodo.

9.4 I numeri reali nel computer

9.4.1 Numeri macchina

In *Matematica 1* hai visto che un computer memorizza i numeri interi in un numero fisso di bit: con n bit si rappresentano solo 2^n numeri diversi, e un risultato troppo grande produce un traboccamento (*overflow*). Con i numeri reali il problema è più serio. Un numero irrazionale come $\sqrt{2}$ ha infinite cifre non periodiche e non può essere scritto in un numero finito di bit; ma neppure tutti i razionali ci stanno, come vedremo tra poco. Un computer lavora quindi con un insieme **finito** di numeri, i **numeri macchina**, e ogni altro numero reale viene sostituito dal numero macchina più vicino. È l'approssimazione del capitolo 1, fatta automaticamente a ogni operazione.

9.4.2 La virgola mobile

Per rappresentare con lo stesso numero di bit sia numeri molto grandi (la distanza Terra-Sole in metri) sia numeri molto piccoli (la massa di un atomo in kilogrammi), i computer usano una versione binaria della notazione scientifica, la **virgola mobile**. Come in base 10 si scrive $149\,600\,000 = 1{,}496 \cdot 10^8$, in base 2 ogni numero diverso da zero si scrive nella forma

$$\pm\, 1{,}m_2 \cdot 2^e$$

dove la parte dopo la virgola, m, si chiama **mantissa** e l'intero e si chiama **esponente**. La virgola «si sposta» (da qui il nome) in modo che prima di essa ci sia sempre una sola cifra 1. Per esempio $6{,}5 = 110{,}1_2$ (perché $6 = 4 + 2$ e $0{,}5 = 2^{-1}$), e spostando la virgola di due posti $110{,}1_2 = 1{,}101_2 \cdot 2^2$.

Il formato più usato, fissato dallo standard internazionale IEEE 754, è la **doppia precisione**: 64 bit, di cui 1 per il segno, 11 per l'esponente e 52 per la mantissa (Figura 9.5). L'1 prima della virgola non si memorizza, perché c'è sempre: le cifre binarie significative sono quindi 53. L'esponente si memorizza aumentato di 1023, così non serve un bit per il suo segno.

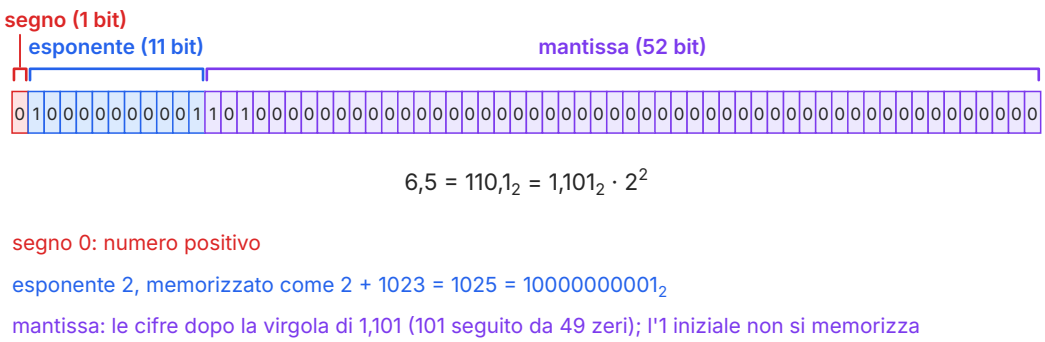


Figura 9.5 Il numero 6,5 in doppia precisione: 64 bit divisi in segno (1 bit), esponente (11 bit) e mantissa (52 bit).

IN SINTESI

Un numero in **doppia precisione** occupa 64 bit: segno (1 bit), esponente (11 bit), mantissa (52 bit), con 53 cifre binarie significative.

Precisione: l'errore relativo di arrotondamento è al più $2^{-53} \approx 1{,}1 \cdot 10^{-16}$; in pratica si conservano circa 16 cifre decimali significative.

Ampiezza: il più grande numero macchina è circa $1{,}8 \cdot 10^{308}$; un risultato più grande produce un traboccamento.

9.4.3 Il numero 0,1 non è un numero macchina

Per scrivere in base 2 un numero compreso tra 0 e 1 c'è un algoritmo semplice: lo si moltiplica per 2; la parte intera del risultato (0 o 1) è la cifra binaria successiva, e si ripete con la parte decimale. Se a un certo punto la parte decimale diventa 0 l'algoritmo termina; se invece ritorna un valore già incontrato, le cifre si ripetono periodicamente.

ESEMPIO SVOLTO

Scrivi in base 2 i numeri 0,375 e 0,1.

0,375. $0{,}375 \cdot 2 = \mathbf{0},75$; $0{,}75 \cdot 2 = \mathbf{1},5$; $0{,}5 \cdot 2 = \mathbf{1},0$. La parte decimale è 0: l'algoritmo termina e $0{,}375 = 0{,}011_2$. Verifica: $2^{-2} + 2^{-3} = 0{,}25 + 0{,}125 = 0{,}375$.

0,1. $0{,}1 \cdot 2 = \mathbf{0},2$; $0{,}2 \cdot 2 = \mathbf{0},4$; $0{,}4 \cdot 2 = \mathbf{0},8$; $0{,}8 \cdot 2 = \mathbf{1},6$; $0{,}6 \cdot 2 = \mathbf{1},2$; $0{,}2 \cdot 2 = \mathbf{0},4$ … La parte decimale 0,2 è già comparsa al secondo passo: da lì le cifre 0, 0, 1, 1 si ripetono all'infinito. Quindi $0{,}1 = 0{,}\overline{00011}_2$, un numero periodico in base 2.

Un numero macchina ha solo 53 cifre significative: 0,1 viene sostituito dal numero macchina più vicino, che differisce da 0,1 di circa $5{,}6 \cdot 10^{-18}$: vicinissimo, ma non è 0,1.

Il motivo generale è lo stesso che, in base 10, rende periodico 1 : 3: una frazione ridotta ai minimi termini ha uno sviluppo binario finito se e solo se il suo denominatore è una potenza di 2. I numeri 0,5, 0,25, $0{,}375 = \frac{3}{8}$ sono numeri macchina; $0{,}1 = \frac{1}{10}$, 0,2 e 0,3 no.

9.4.4 Errori di arrotondamento e traboccamento

Ogni operazione tra numeri macchina produce un risultato che viene a sua volta arrotondato al numero macchina più vicino: è l'**errore di arrotondamento**. È piccolissimo, ma può diventare visibile. In doppia precisione la somma 0,1 + 0,2 dà 0,30000000000000004: i due addendi erano già approssimati, e l'arrotondamento della somma non coincide con il numero macchina più vicino a 0,3. Allo stesso modo $2^{53} = 9\,007\,199\,254\,740\,992$ è un numero macchina, ma $2^{53} + 1$ non lo è (servirebbero 54 cifre binarie): un computer calcola $2^{53} + 1 = 2^{53}$. All'estremo opposto, il prodotto di numeri molto grandi produce un traboccamento: $170! \approx 7{,}3 \cdot 10^{306}$ è ancora rappresentabile, mentre 171! supera il massimo numero macchina e il computer lo segnala come «infinito». È per questo che, nel capitolo 1, il metodo di Erone nel foglio di calcolo a un certo punto smetteva di migliorare: aveva raggiunto il limite di precisione della macchina.

ERRORE FREQUENTE

Controllare se due numeri macchina sono uguali. L'algoritmo «$x \leftarrow 0$; **MENTRE** $x \neq 1$ **ESEGUI** $x \leftarrow x + 0{,}1$» dovrebbe fermarsi dopo dieci somme. Eseguito al computer non si ferma mai: dopo dieci somme x vale 0,9999999999999999, poi supera 1 senza mai coincidere con 1.

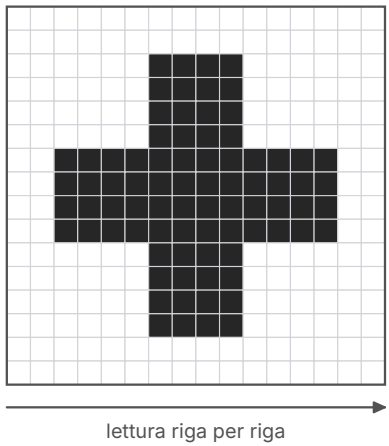
Tra numeri ottenuti con calcoli approssimati non si controlla l'uguaglianza, ma la vicinanza: $|x - 1| < \varepsilon$, con una tolleranza ε adatta (come nel criterio di arresto di Erone). Oppure si conta con una variabile intera: **PER** *i* **DA** 1 **A** 10.

9.5 Dati multimediali e compressione

Ricordiamo da *Matematica 1* che un'immagine digitale è una griglia di pixel: in un'immagine in bianco e nero basta 1 bit per pixel, in un'immagine a colori RGB servono 3 byte (24 bit) per pixel; un suono si memorizza misurando l'onda molte volte al secondo (in qualità CD, 44 100 campioni al secondo per canale, ciascuno di 16 bit, su due canali). Con questi numeri i file diventano presto enormi: una foto di 4000 × 3000 pixel occupa 12 000 000 · 3 = 36 000 000 byte, cioè 36 MB, e un minuto di musica in qualità CD più di 10 MB. Per conservarli e trasmetterli si usa la **compressione**: un algoritmo che scrive gli stessi dati con meno bit.

9.5.1 Compressione senza perdita: la codifica RLE

Una compressione è **senza perdita** se dal file compresso si ricostruisce **esattamente** il file originale. L'esempio più semplice è la **codifica RLE** (dall'inglese *run-length encoding*, codifica delle sequenze): invece di elencare i pixel uno per uno, si scrive quanti pixel consecutivi hanno lo stesso colore. In un'immagine in bianco e nero, letta riga per riga, i colori si alternano: basta allora la lista delle lunghezze delle sequenze, con la convenzione che la prima sequenza sia bianca (eventualmente di lunghezza 0, se l'immagine comincia con un pixel nero).



Codice RLE (B = bianco, N = nero):

38B **4N** 12B **4N** 12B **4N** 12B
4N 8B **12N** 4B **12N** 4B **12N**
4B **12N** 8B **4N** 12B **4N** 12B
4N 12B **4N** 38B

25 sequenze, somma 256 pixel

senza compressione: 256 pixel × 1 bit = 256 bit

RLE: 25 sequenze × 8 bit = 200 bit

Figura 9.6 Un'immagine in bianco e nero di 16 × 16 pixel e la sua codifica RLE: le lunghezze delle sequenze di pixel bianchi (B) e neri (N), lette riga per riga e proseguendo da una riga alla successiva.

ESEMPIO SVOLTO

Codifica con RLE l'immagine della Figura 9.6 e confronta lo spazio occupato con quello dell'immagine non compressa, usando 1 byte (8 bit) per ogni lunghezza.

Le prime due righe e i primi 6 pixel della terza sono bianchi: 16 + 16 + 6 = 38 pixel bianchi. Seguono 4 neri, poi 6 bianchi a fine riga e 6 all'inizio della riga successiva, cioè 12 bianchi, e così via. Le righe con il braccio orizzontale della croce danno 2 bianchi a fine riga più 2 all'inizio della successiva: 4 bianchi tra due sequenze di 12 neri. In tutto si ottengono 25 lunghezze, la cui somma è 256 come deve essere.

Non compressa, l'immagine occupa 256 pixel × 1 bit = 256 bit. Codificata, occupa 25 × 8 = 200 bit: circa il 78% dell'originale. Su immagini più grandi con ampie zone uniformi (un documento scansionato, un disegno tecnico) il guadagno è molto maggiore.

La codifica RLE non è sempre vantaggiosa. In un'immagine a scacchiera, dove il colore cambia a ogni pixel, le sequenze sono quasi tutte lunghe 1 e il file «compresso» diventa molto più grande dell'originale. È un fatto generale, non un difetto di questo algoritmo: nessuna compressione senza perdita può ridurre *tutti* i file, perché i file di n bit sono 2^n e quelli più corti sono meno: qualche file deve per forza restare uguale o allungarsi.

9.5.2 Compressione con perdita

Nelle foto e nella musica si accetta di perdere qualcosa, purché non si noti. Una compressione **con perdita** scarta le informazioni che l'occhio o l'orecchio percepiscono meno: piccole varia-

zioni di colore in una zona dell'immagine, suoni molto deboli coperti da altri più forti. Il file ricostruito non è identico all'originale, ma gli somiglia abbastanza. Sono di questo tipo i formati JPEG per le immagini e MP3 per l'audio. Un file MP3 a 128 kbit al secondo, per esempio, occupa per ogni secondo di musica 128 000 bit, contro i $44\,100 \cdot 16 \cdot 2 = 1\,411\,200$ bit del CD: circa undici volte meno.

9.6 Che cosa si può calcolare

9.6.1 La macchina di Turing

Finora abbiamo descritto gli algoritmi in modo informale. Per dimostrare che un problema *non* si può risolvere con un algoritmo serve invece una definizione precisa. Nel 1936 il matematico inglese Alan Turing propose un modello di esecutore così semplice da poter essere descritto completamente, e insieme così potente da eseguire qualunque algoritmo: la **macchina di Turing**.

IN SINTESI

Una **macchina di Turing** è formata da:

- un **nastro** illimitato diviso in caselle; ogni casella contiene un simbolo di un alfabeto finito (per esempio 0, 1) oppure è vuota (␣);
- una **testina** che a ogni passo legge la casella su cui si trova, vi scrive un simbolo e si sposta di una casella a destra (→), a sinistra (←) oppure resta ferma (–);
- un numero finito di **stati** q_0, q_1, …, tra cui uno stato iniziale q_0 e uno **stato di arresto** q_F;
- una **tabella delle transizioni**: per ogni coppia (stato, simbolo letto) indica il simbolo da scrivere, lo spostamento e il nuovo stato. Scriveremo (q, letto) → (scritto, spostamento, nuovo stato).

La macchina parte nello stato q_0 con la testina sul primo simbolo dei dati, applica le regole una alla volta e si ferma quando entra nello stato q_F. Il contenuto del nastro a quel punto è il risultato.

La Figura 9.7 mostra un passo di calcolo di una macchina molto semplice, che scambia gli 0 con gli 1 in una stringa di cifre binarie. Ha un solo stato di lavoro, q_0, e tre regole (Tabella 9.3): se legge 0 scrive 1, se legge 1 scrive 0, e in entrambi i casi si sposta a destra; quando trova la prima casella vuota, la stringa è finita e la macchina si ferma. Partendo da 0110, dopo quattro passi il nastro contiene 1001 e al quinto passo la macchina entra in q_F.

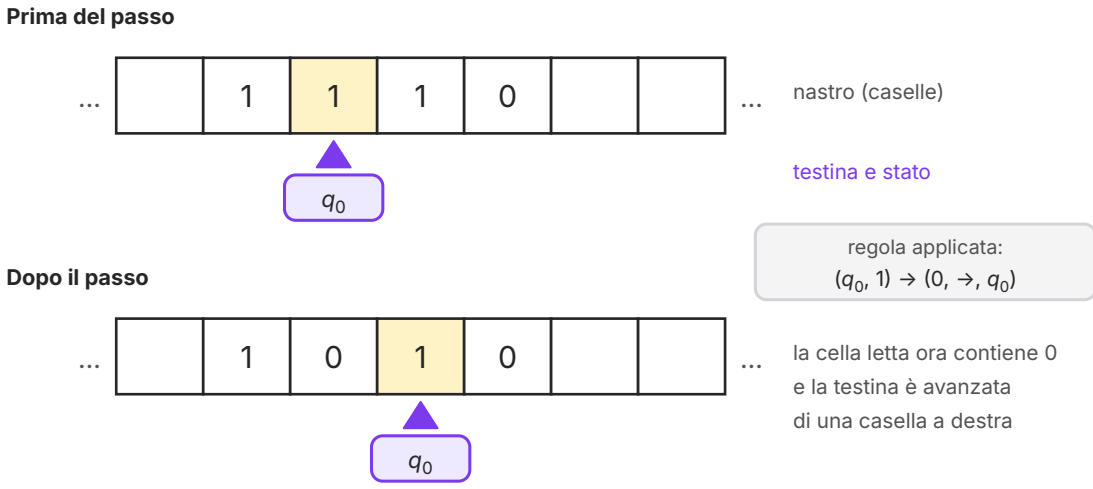


Figura 9.7 Un passo della macchina che scambia 0 e 1. La testina, nello stato q_0, legge 1 nella casella evidenziata: la regola $(q_0, 1) \to (0, \to, q_0)$ le fa scrivere 0 e spostarsi di una casella a destra, restando nello stato q_0. La stringa di partenza era 0110: al primo passo lo 0 iniziale è già diventato 1.

Tabella 9.3 La macchina che scambia 0 e 1.

stato	letto	scritto	spostamento	nuovo stato
q_0	0	1	→	q_0
q_0	1	0	→	q_0
q_0	□	□	–	q_F

9.6.2 Altri esempi

Una macchina di Turing può anche calcolare funzioni numeriche. Il modo più semplice di scrivere un numero naturale n sul nastro è la notazione **unaria**: n cifre 1 consecutive (3 si scrive 111). La funzione *successore*, $n \mapsto n + 1$, è calcolata dalla macchina della Tabella 9.4: la testina scorre verso destra gli 1 senza modificarli e, alla prima casella vuota, scrive un 1 e si ferma. Partendo da 111 la macchina fa tre passi di scorrimento e un passo di scrittura: il nastro contiene 1111, cioè 4.

Tabella 9.4 La macchina che calcola il successore in notazione unaria.

stato	letto	scritto	spostamento	nuovo stato
q_0	1	1	→	q_0
q_0	□	1	–	q_F

Una macchina di Turing può anche **decidere** se un dato ha una certa proprietà, scrivendo sul nastro la risposta. La macchina della Tabella 9.5 stabilisce se una stringa di 1 ha lunghezza pari: usa due stati, q_P («finora ho letto un numero pari di 1») e q_D («finora dispari»), e alla fine della stringa scrive P oppure D. Lo stato fa da memoria: la macchina non conta gli 1, ricorda solo la parità.

Tabella 9.5 La macchina che decide se una stringa di 1 ha lunghezza pari: lo stato iniziale è qui la memoria «pari», ed è chiamato per chiarezza q_P invece di q_0.

stato	letto	scritto	spostamento	nuovo stato
q_P	1	1	→	q_D
q_D	1	1	→	q_P
q_P	□	P	–	q_F
q_D	□	D	–	q_F

ESEMPIO SVOLTO

Esegui la macchina della Tabella 9.5 sulla stringa 111, partendo dallo stato q_P con la testina sul primo 1.

Passo 1. Stato q_P, legge 1 (casella 1): scrive 1, va a destra, stato q_D.

Passo 2. Stato q_D, legge 1 (casella 2): scrive 1, va a destra, stato q_P.

Passo 3. Stato q_P, legge 1 (casella 3): scrive 1, va a destra, stato q_D.

Passo 4. Stato q_D, legge ▫ (casella 4): scrive D, resta ferma, stato q_F: la macchina si ferma.

Il nastro contiene 111D: la stringa ha lunghezza dispari. Con 1111 la macchina avrebbe fatto un passo in più e scritto P; con la stringa vuota scrive subito P, perché 0 è pari.

9.6.3 Funzioni calcolabili e tesi di Church-Turing

Una funzione si dice **calcolabile secondo Turing** se esiste una macchina di Turing che, ricevuto sul nastro un qualunque dato del dominio, si ferma dopo un numero finito di passi lasciando sul nastro il valore della funzione. Il successore è calcolabile; con macchine più lunghe, che se-

guono le stesse regole, si calcolano la somma, il prodotto, il MCD con l'algoritmo di Euclide, e tutte le funzioni che sai calcolare con carta e penna.

Turing sostenne che la sua definizione cattura esattamente l'idea intuitiva di algoritmo: **ogni funzione che si può calcolare con un algoritmo si può calcolare con una macchina di Turing**. Nello stesso 1936 il logico americano Alonzo Church arrivò alla stessa conclusione con un formalismo diverso, e i due modelli si rivelarono equivalenti. L'affermazione è nota come **tesi di Church-Turing**. È una *tesi* e non un teorema: non si può dimostrare, perché collega un concetto preciso (la macchina di Turing) a uno intuitivo (l'algoritmo). Ma nessuno ha mai trovato un procedimento di calcolo, compresi quelli dei computer moderni, che una macchina di Turing non sappia imitare.

9.6.4 Esistono funzioni non calcolabili

Il risultato più sorprendente di Turing è negativo. Come hai letto in *Matematica 1*, non esiste un algoritmo che, ricevuti un qualunque programma e i suoi dati, decida sempre correttamente se quel programma, eseguito su quei dati, prima o poi si ferma: è il **problema della fermata**. La funzione che a ogni coppia (programma, dati) associa «si ferma» o «non si ferma» è ben definita, ma **non calcolabile**. Non è una questione di velocità o di memoria: nessun computer, per quanto potente, potrà mai calcolarla. Esistono dunque problemi formulati con precisione e del tutto privi di soluzione algoritmica: un limite della matematica stessa, scoperto con gli strumenti della matematica.

APPROFONDIMENTO · L'ARGOMENTO DIAGONALE

L'idea della dimostrazione risale a Georg Cantor (1891), che la usò per provare che le successioni infinite di due simboli (e quindi, in sostanza, i numeri reali) non si possono mettere in fila come i naturali. In forma semplificata funziona così. Ogni programma è una stringa finita di simboli, quindi i programmi si possono elencare: P_1, P_2, P_3, … Supponiamo, per assurdo, che esista un programma H che risolve il problema della fermata: dati P_i e un dato, risponde sempre (e in un tempo finito) «si ferma» o «non si ferma».

Con H costruiamo un nuovo programma Q: ricevuto un numero i, Q chiede a H se P_i, eseguito con il dato i, si ferma; se la risposta è «si ferma», Q entra in un ciclo infinito; se è «non si ferma», Q si ferma subito. Così Q si comporta, su ogni dato i, in modo **opposto** a P_i: si ferma esattamente quando P_i non si ferma.

Ma Q è un programma, quindi è nell'elenco: $Q = P_k$ per qualche k. Eseguiamo Q con il dato k: per costruzione si ferma se e solo se P_k, cioè Q stesso, non si ferma. È una contraddizione, e l'unica ipotesi fatta era l'esistenza di H: quindi H non esiste. Il nome «diagonale» viene dal fatto che Q si confronta con ogni P_i proprio sul dato i: in una tabella con i programmi sulle righe e i dati sulle colonne, si guardano solo le caselle della diagonale.

SINTESI

- Un algoritmo per **approssimazioni successive** migliora un valore ripetendo lo stesso calcolo e ha bisogno di un **criterio di arresto**. Il metodo di Erone ($x \leftarrow (x + a : x) : 2$) si ferma quando $|x - a : x| < \varepsilon$, e allora l'errore è minore di ε; le cifre esatte circa raddoppiano a ogni passo.
- La bisezione dimezza a ogni passo l'intervallo che contiene una soluzione: è generale, ma lenta (circa 10 passi per tre cifre decimali).
- Gli algoritmi per l'equazione di primo grado $ax = b$ e per i sistemi 2×2 sono esempi di selezione: i valori di a e b, oppure di D, D_x, D_y, decidono il caso.
- La **ricerca sequenziale** richiede fino a n confronti; la **ricerca binaria**, su un elenco ordinato, al più la parte intera di $\log_2 n$ più uno (20 per un milione di elementi).
- L'**ordinamento per selezione** di n elementi richiede sempre $n(n - 1) : 2$ confronti.
- La **complessità** descrive come cresce il costo di un algoritmo con la dimensione dei dati: logaritmica, lineare, quadratica. Con molti dati la scelta dell'algoritmo conta più della velocità della macchina.
- Il **metodo Monte Carlo** stima $\pi \approx 4K : N$ contando i punti casuali che cadono in un quarto di cerchio; è semplice e versatile, ma lento e con un risultato soltanto probabilmente vicino al vero.
- Un computer usa un insieme finito di **numeri macchina** in **virgola mobile** (segno, esponente, mantissa). La doppia precisione ha 64 bit (1 + 11 + 52), 53 cifre binarie significative, circa 16 cifre decimali.
- Il numero 0,1 è periodico in base 2 ($0{,}0\overline{0011}_2$): per gli errori di arrotondamento $0{,}1 + 0{,}2 \neq 0{,}3$ in doppia precisione. Tra numeri macchina si controlla la vicinanza, non l'uguaglianza.
- La compressione **senza perdita** (come la codifica RLE) ricostruisce esattamente i dati, ma non può ridurre tutti i file; quella **con perdita** (JPEG, MP3) scarta ciò che si percepisce meno.
- Una **macchina di Turing** (nastro, testina, stati, tabella delle transizioni) è un modello preciso di algoritmo. Secondo la **tesi di Church-Turing** ogni funzione calcolabile con un algoritmo è calcolabile da una macchina di Turing.
- Esistono funzioni ben definite ma **non calcolabili**: il problema della fermata non ha soluzione algoritmica.

ESERCIZI

Comprendere

1. ● In un elenco ordinato di 1000 elementi, quanti confronti richiede al massimo la ricerca binaria?
   - **A.** 500
   - **B.** 1000
   - **C.** 10
   - **D.** 100

2. ● Quanti confronti esegue l'ordinamento per selezione su un elenco di 20 elementi?
   - **A.** 20
   - **B.** 190
   - **C.** 400
   - **D.** 380

3. ● In doppia precisione, quante cifre binarie significative ha un numero macchina (contando l'1 che precede la virgola)?
   - **A.** 52
   - **B.** 11
   - **C.** 64
   - **D.** 53

4. ● Quale di questi numeri ha uno sviluppo binario finito?
   - **A.** 0,1
   - **B.** 0,2
   - **C.** 0,375
   - **D.** $\frac{1}{3}$

5. ● In una simulazione Monte Carlo, 790 punti casuali su 1000 cadono nel quarto di cerchio. La stima di π è:
   - **A.** 3,16
   - **B.** 0,79
   - **C.** 3,14
   - **D.** 2,37

6. ●● La tesi di Church-Turing afferma che:
   - **A.** ogni funzione è calcolabile da una macchina di Turing
   - **B.** esiste un algoritmo che risolve il problema della fermata
   - **C.** un computer moderno calcola funzioni che nessuna macchina di Turing può calcolare
   - **D.** ogni funzione calcolabile con un algoritmo è calcolabile da una macchina di Turing

7. ● Vero o falso? Correggi le affermazioni false.
   a) La ricerca binaria si può usare anche su un elenco non ordinato.
   b) In doppia precisione la somma 0,1 + 0,2 dà esattamente 0,3.
   c) Dal file compresso con la codifica RLE si ricostruisce esattamente l'immagine originale.
   d) La tesi di Church-Turing è un teorema dimostrato da Turing nel 1936.
   e) Ripetendo una simulazione Monte Carlo con punti diversi si ottiene sempre la stessa stima.
   f) In un elenco ordinato di 64 elementi la ricerca binaria richiede al massimo 7 confronti.

8. ● Completa.
   a) In virgola mobile un numero si scrive con un segno, una _____ e un _____ .
   b) Una macchina di Turing è formata da un nastro, una _____ , un insieme finito di _____ e una tabella delle _____ .
   c) Una compressione si dice _____ quando dal file compresso si ricostruisce esattamente l'originale.
   d) Il numero di confronti dell'ordinamento per selezione di n elementi è _____ .
   e) Una condizione che stabilisce quando un algoritmo per approssimazioni successive deve fermarsi si chiama _____ .

Calcolare

9. ● Applica il metodo di Erone per calcolare $\sqrt{7}$, partendo da $x = 3$ con tolleranza $\varepsilon = 0{,}01$. Compila la tabella di traccia (x, $7 : x$, differenza), scrivendo i valori di x anche come frazioni. Quale valore scrive l'algoritmo?

10. ●● L'equazione $x^3 - 2x - 2 = 0$ ha una soluzione nell'intervallo $[1; 2]$. a) Verifica che $f(x) = x^3 - 2x - 2$ cambia segno agli estremi. b) Esegui quattro passi di bisezione, riportando a ogni passo il punto medio, il segno di f e il nuovo intervallo. c) Quanto è ampio l'intervallo finale?

11. ● Esegui l'algoritmo per l'equazione $ax = b$ (pseudocodice del paragrafo sulla selezione) con i dati: a) $a = 3$, $b = -12$; b) $a = 0$, $b = 5$; c) $a = 0$, $b = 0$. Per ciascun caso indica quali condizioni l'algoritmo esamina e che cosa scrive.

12. ●● Per ciascun sistema calcola D, D_x, D_y e segui l'algoritmo descritto nel testo: a) $2x + 3y = 7$, $x - y = 1$; b) $x + 2y = 3$, $2x + 4y = 5$; c) $x + 2y = 3$, $2x + 4y = 6$.

13. ● Nell'elenco ordinato della Figura 9.2 (3, 7, 12, 15, 21, 26, 30, 34, 41, 47, 52, 58, 63, 70, 77, 85) esegui la ricerca binaria di 21 e poi di 50. Per ciascuna scrivi gli elementi confrontati e il numero di confronti.

14. ●● Ordina l'elenco 6, 2, 7, 4, 1, 5 con l'ordinamento per selezione. Scrivi l'elenco dopo ogni passata e conta i confronti e gli scambi effettivamente eseguiti (non contare lo «scambio» di un elemento con se stesso).

15. ● Scrivi in base 2, con l'algoritmo delle moltiplicazioni per 2: a) 0,75; b) 0,625; c) 0,2. Quali sono numeri macchina?

16. ●● Scrivi i tre campi (segno, esponente memorizzato, prime cifre della mantissa) della rappresentazione in doppia precisione di: a) 13,25; b) −0,75.

17. ● a) Un'immagine in bianco e nero di 8 × 4 pixel ha la codifica RLE 3, 2, 6, 2, 6, 2, 6, 2, 3 (la prima sequenza è bianca). Disegnala. b) Con 8 bit per ogni lunghezza, quanti bit occupa la codifica? Conviene rispetto all'immagine non compressa? c) Scrivi la codifica RLE dell'immagine di 8 × 2 pixel le cui righe sono entrambe NNNNBBBB (N = nero, B = bianco).

18. ● a) Esegui la macchina che scambia 0 e 1 (Tabella 9.3) sulla stringa 1011: quanti passi fa e che cosa resta sul nastro? b) Esegui la macchina del successore (Tabella 9.4) sul numero 2 in notazione unaria.

19. ●● Esegui passo per passo la macchina seguente sulla stringa 11, partendo da q_0 sul primo 1, e descrivi che cosa fa in generale.
$(q_0, 1) \to (1, \to, q_0)$; $(q_0, \square) \to (0, \leftarrow, q_1)$; $(q_1, 1) \to (1, \leftarrow, q_1)$; $(q_1, \square) \to (\square, \to, q_F)$.

Dimostrare e argomentare

20. ●● a) Dimostra che, se $a > 0$ e $x > \sqrt{a}$, allora $a : x < \sqrt{a}$. b) Deduci che, se $x \neq \sqrt{a}$, il numero $\sqrt{a}$ è compreso tra x e $a : x$, e quindi che $|x - \sqrt{a}| < |x - a : x|$. c) Spiega perché il criterio di arresto del metodo di Erone garantisce un errore minore di ε.

21. ●● Spiega perché, in un elenco ordinato di 2^k elementi, la ricerca binaria richiede al massimo $k + 1$ confronti. (Suggerimento: quanti elementi restano, al più, dopo un confronto con esito negativo?)

22. ● Dimostra che l'ordinamento per selezione di n elementi esegue $n(n - 1) : 2$ confronti, qualunque sia l'ordine iniziale degli elementi.

23. ● Trova l'errore. Luca vuole scrivere i numeri 0; 0,1; 0,2; ...; 1 e propone: «$x \leftarrow 0$; **MENTRE** $x \neq 1$ **ESEGUI SCRIVI** x; $x \leftarrow x + 0{,}1$ **FINE MENTRE**; **SCRIVI** x». Eseguito al computer, il programma non si ferma. Spiega perché e correggilo in due modi diversi.

24. ●●● Dimostra che una frazione $p : q$, ridotta ai minimi termini e compresa tra 0 e 1, ha uno sviluppo binario finito se e solo se q è una potenza di 2. (Suggerimento: uno sviluppo con k cifre dopo la virgola rappresenta un numero della forma $N : 2^k$.) Deduci che 0,1 non è un numero macchina.

Problemi

25. ● Un elenco ordinato contiene 1 000 000 di nomi e un confronto richiede un milionesimo di secondo. Quanto tempo richiede, nel caso peggiore, la ricerca di un nome con la ricerca sequenziale? E con la ricerca binaria?

26. ●● Un computer esegue un miliardo di confronti al secondo. Quanto tempo impiega a ordinare per selezione a) 100 000 elementi; b) 10 000 000 di elementi? Esprimi il secondo risultato in ore.

27. ● Una fotocamera scatta foto di 4000 × 3000 pixel a colori (3 byte per pixel). a) Quanto occupa una foto non compressa? b) Con la compressione JPEG il file occupa 3,6 MB: qual è il rapporto di compressione? c) Quante foto compresse entrano in una scheda di memoria da 64 GB (1 GB = 10^9 byte)?

28. ●● Una canzone dura 4 minuti. a) Quanti byte occupa in qualità CD (44 100 campioni al secondo, 16 bit per campione, 2 canali)? b) Quanti in formato MP3 a 128 kbit al secondo (1 kbit = 1000 bit)? c) Qual è il rapporto tra le due dimensioni?

29. ●● In una simulazione Monte Carlo con 10 000 punti, 7 858 cadono nel quarto di cerchio. a) Calcola la stima di π, il suo errore assoluto e il suo errore relativo (usa π ≈ 3,14159). b) Quanti punti avrebbero dovuto cadere dentro per ottenere la stima 3,1416? c) Perché non ha senso chiedersi quanti punti servono per essere *sicuri* che la stima abbia due decimali esatti?

30. ●● Il più grande numero in doppia precisione è circa $1{,}8 \cdot 10^{308}$, e $170! \approx 7{,}26 \cdot 10^{306}$. Stima 171! e spiega perché un programma che calcola i fattoriali in doppia precisione si ferma con un traboccamento proprio a 171.

Con il computer

31. ●● In un foglio di calcolo realizza il metodo di Erone per √2: in A2 il valore iniziale 1, in B2 la formula =2/A2, in C2 =ASS(A2-B2), in A3 =(A2+B2)/2; copia le formule verso il basso. a) A quale passo la colonna C scende sotto 10^{-10}? b) Che cosa succede alla colonna C nei passi successivi, e perché?

32. ●● Realizza nel foglio di calcolo la stima Monte Carlo di π descritta nel paragrafo «Con il computer», con 1000 punti. a) Ricalcola il foglio dieci volte e annota le stime: qual è la più piccola e quale la più grande? b) Ripeti con 10 000 punti. Che cosa osservi?

Verso la verifica

33. ●● Progetta una macchina di Turing che stabilisca se una stringa di 1 ha una lunghezza multipla di 3, scrivendo alla fine S (sì) oppure N (no). Scrivi la tabella delle transizioni ed eseguila sulla stringa 1111.

34. ●● a) Con la bisezione, partendo da un intervallo di ampiezza 1, quanti passi servono per ottenere un intervallo di ampiezza minore di 10^{-6}? b) Confronta con il metodo di Erone per √10 nella Figura 9.1: quale dei due conviene, e perché la bisezione resta comunque utile?

35. ●●● *Sfida.* Un numero naturale si scrive in notazione unaria con tanti 1 quanto vale. Progetta una macchina di Turing che calcoli la somma di due numeri scritti in notazione unaria e separati da uno 0 (per esempio 110111 rappresenta 2 + 3) lasciando sul nastro il risultato in notazione unaria. Scrivi la tabella delle transizioni, eseguila su 110111 contando i passi, e controlla che funzioni anche quando uno dei due addendi è 0.

PER APPROFONDIRE

- [HAR] Harel, *Algorithmics: The Spirit of Computing*: le strategie algoritmiche, la complessità, la calcolabilità e i problemi non risolubili, spiegati senza linguaggi di programmazione.
- [CR] Courant, Robbins, *Che cos'è la matematica?*: i sistemi di numerazione, i numeri reali e le loro approssimazioni.
- [BOY] Boyer, *Storia della matematica*: dagli algoritmi numerici antichi, come quello di Erone, alla logica matematica del Novecento.

Glossario del capitolo

Approssimazioni successive
Strategia algoritmica che parte da un valore approssimato e lo migliora ripetendo sempre lo stesso calcolo.

Codifica RLE
Compressione senza perdita che sostituisce ogni sequenza di simboli uguali consecutivi con la sua lunghezza.

Complessità
Il modo in cui il numero di operazioni di un algoritmo cresce al crescere della dimensione dei dati (logaritmica, lineare, quadratica...).

Compressione con perdita
Codifica che riduce lo spazio occupato scartando informazioni poco percepibili; il file ricostruito non coincide con l'originale (JPEG, MP3).

Compressione senza perdita
Codifica che riduce lo spazio occupato da un file e permette di ricostruirlo esattamente (per esempio la codifica RLE).

Criterio di arresto
Condizione che stabilisce quando un algoritmo iterativo deve fermarsi, per esempio quando la differenza tra due approssimazioni è minore di una tolleranza ε.

Doppia precisione
Formato in virgola mobile a 64 bit (1 di segno, 11 di esponente, 52 di mantissa), con circa 16 cifre decimali significative.

Errore di arrotondamento
Differenza tra il risultato esatto di un'operazione e il numero macchina con cui viene rappresentato.

Logaritmo in base 2
Di un numero positivo n: l'esponente a cui elevare 2 per ottenere n ($\log_2 8 = 3$).

Macchina di Turing
Modello di calcolo formato da un nastro illimitato di caselle, una testina che legge e scrive, un insieme finito di stati e una tabella delle transizioni.

Mantissa
Nella virgola mobile, la sequenza delle cifre binarie che seguono la virgola nella forma 1,m.

Metodo Monte Carlo
Metodo di calcolo che usa numeri casuali e stima un valore con la frequenza di un evento simulato, per esempio π con i punti che cadono in un quarto di cerchio.

Numero macchina
Numero rappresentabile esattamente con i bit a disposizione di un computer; gli altri numeri reali vengono sostituiti dal numero macchina più vicino.

Ordinamento per selezione
Algoritmo di ordinamento che a ogni passata cerca il minimo degli elementi non ancora sistemati e lo porta al primo posto libero; esegue $n(n-1)/2$ confronti.

Problema della fermata
Il problema di decidere, per un qualunque programma e i suoi dati, se l'esecuzione termina; Turing dimostrò che nessun algoritmo lo risolve.

Ricerca binaria
Ricerca in un elenco ordinato che confronta il dato con l'elemento centrale e dimezza a ogni passo la parte in cui cercare; richiede circa $\log_2 n$ confronti.

Ricerca sequenziale
Ricerca di un dato confrontandolo uno dopo l'altro con tutti gli elementi di un elenco; nel caso peggiore richiede n confronti.

Tabella delle transizioni
Elenco delle regole di una macchina di Turing: per ogni stato e simbolo letto, il simbolo da scrivere, lo spostamento e il nuovo stato.

Tesi di Church-Turing
Affermazione, non dimostrabile, secondo cui ogni funzione calcolabile mediante un algoritmo è calcolabile da una macchina di Turing.

Virgola mobile
Rappresentazione di un numero in notazione scientifica binaria, $\pm 1{,}m \cdot 2^e$, con un segno, una mantissa e un esponente.

Soluzioni

1. C. La parte intera di $\log_2 1000 \approx 9{,}97$ è 9; più uno: 10 (infatti $2^9 = 512 < 1000 < 1024 = 2^{10}$).

2. B. $20 \cdot 19 : 2 = 190$.

3. D. 52 cifre memorizzate più l'1 iniziale, che non si memorizza: 53.

4. C. 0,375 = 3/8 ha per denominatore una potenza di 2: $0{,}375 = 0{,}011_2$. Gli altri sono periodici in base 2.

5. A. $4 \cdot 790 : 1000 = 3{,}16$.

6. D. La tesi collega l'idea intuitiva di algoritmo alla macchina di Turing; il problema della fermata non è risolubile e i computer moderni non superano le macchine di Turing.

7. a) F: serve un elenco ordinato (altrimenti scartare una metà non è giustificato). b) F: dà 0,30000000000000004. c) V. d) F: è una tesi, non dimostrabile, perché collega un concetto preciso a uno intuitivo. e) F: la stima cambia con i punti. f) V: $\log_2 64 = 6$, più uno fa 7.

8. a) mantissa, esponente. b) testina, stati, transizioni. c) senza perdita. d) $n(n-1) : 2$. e) criterio di arresto.

9. Passo 0: $x = 3$, $7 : x \approx 2{,}3333$, differenza $\approx 0{,}667$. Passo 1: $x = 8/3 \approx 2{,}6667$, $7 : x = 21/8 = 2{,}625$, differenza $\approx 0{,}0417 \geq 0{,}01$. Passo 2: $x = 127/48 \approx 2{,}64583$, $7 : x = 336/127 \approx 2{,}64567$, differenza $\approx 0{,}00016 < 0{,}01$: l'algoritmo scrive $127/48 \approx 2{,}6458$ ($\sqrt{7} = 2{,}64575\ldots$).

10. a) $f(1) = -3 < 0$, $f(2) = 2 > 0$. b) $m = 1{,}5$: $f = -1{,}625$ (−), [1,5; 2]; $m = 1{,}75$: $f \approx -0{,}141$ (−), [1,75; 2]; $m = 1{,}875$: $f \approx 0{,}842$ (+), [1,75; 1,875]; $m = 1{,}8125$: $f \approx 0{,}329$ (+), [1,75; 1,8125]. c) 1/16 = 0,0625 (la soluzione è circa 1,7693).

11. a) $a \neq 0$: scrive «una soluzione»: $-12 : 3 = -4$ (il secondo SE non viene esaminato). b) $a = 0$ e $b \neq 0$: «impossibile». c) $a = 0$ e $b = 0$: «indeterminata: ogni numero reale è soluzione».

12. a) $D = 2 \cdot (-1) - 3 \cdot 1 = -5$, $D_x = 7 \cdot (-1) - 3 \cdot 1 = -10$, $D_y = 2 \cdot 1 - 7 \cdot 1 = -5$: determinato, $x = 2$, $y = 1$. b) $D = 4 - 4 = 0$, $D_x = 3 \cdot 4 - 2 \cdot 5 = 2 \neq 0$: impossibile. c) $D = 0$, $D_x = 12 - 12 = 0$, $D_y = 6 - 6 = 0$: indeterminato (la seconda equazione è il doppio della prima).

13. 21: si confrontano 34 (posto 8, 21 < 34), 15 (posto 4, 21 > 15), 26 (posto 6, 21 < 26), 21 (posto 5): trovato, 4 confronti. 50: 34 (posto 8), 58 (posto 12), 47 (posto 10), 52 (posto 11); a quel punto $i = 11 > j = 10$: non presente, 4 confronti.

14. Dopo le passate: 1, 2, 7, 4, 6, 5 (scambio 6 ↔ 1); 1, 2, 7, 4, 6, 5 (il 2 è già al suo posto); 1, 2, 4, 7, 6, 5 (scambio 7 ↔ 4); 1, 2, 4, 5, 6, 7 (scambio 7 ↔ 5); 1, 2, 4, 5, 6, 7 (il 6 è già al suo posto). Confronti: 5 + 4 + 3 + 2 + 1 = 15; scambi: 3.

15. a) $0{,}75 \cdot 2 = 1{,}5 \to 1$; $0{,}5 \cdot 2 = 1{,}0 \to 1$: $0{,}11_2$. b) $0{,}625 \cdot 2 = 1{,}25 \to 1$; $0{,}25 \cdot 2 = 0{,}5 \to 0$; $0{,}5 \cdot 2 = 1 \to 1$: $0{,}101_2$. c) $\underline{0{,}2} \to 0{,}4\ (0) \to 0{,}8\ (0) \to 1{,}6\ (1) \to 1{,}2\ (1) \to 0{,}4\ (0) \ldots$: $0{,}0011_2$. Sono numeri macchina 0,75 e 0,625; 0,2 no.

16. a) $13{,}25 = 1101{,}01_2 = 1{,}10101_2 \cdot 2^3$: segno 0; esponente $3 + 1023 = 1026 = 10000000010_2$; mantissa 10101 seguito da 47 zeri. b) $-0{,}75 = -0{,}11_2 = -1{,}1_2 \cdot 2^{-1}$: segno 1; esponente $-1 + 1023 = 1022 = 01111111110_2$; mantissa 1 seguito da 51 zeri.

17. a) Ogni riga ha 3 pixel bianchi, 2 neri, 3 bianchi: una barra verticale nera di 2 pixel al centro, alta 4 righe (i 6 bianchi uniscono la fine di una riga all'inizio della successiva; 3 + 2 + 6 + 2 + 6 + 2 + 6 + 2 + 3 = 32). b) $9 \cdot 8 = 72$ bit, contro 32 bit dell'immagine non compressa: non conviene, le sequenze sono troppo corte. c) 0, 4, 4, 4, 4 (la prima sequenza bianca ha lunghezza 0; i 4 bianchi di fine riga e i 4 neri della riga dopo restano separati).

18. a) Quattro passi di scrittura (1 → 0, 0 → 1, 1 → 0, 1 → 0) più il passo sulla casella vuota che porta in q_F: 5 passi; sul nastro resta 0100. b) Si parte da 11: due

passi di scorrimento, poi sulla casella vuota scrive 1 e si ferma (3 passi): 111, cioè 3.

19. Passi: $(q_0, 1) \to$ a destra; $(q_0, 1) \to$ a destra; $(q_0, \square) \to$ scrive 0, va a sinistra, q_1; $(q_1, 1) \to$ a sinistra; $(q_1, 1) \to$ a sinistra; $(q_1, \square) \to$ a destra, q_F. Il nastro contiene 110 e la testina è sul primo 1 (6 passi). In generale la macchina aggiunge uno 0 in fondo a una stringa di 1 e torna all'inizio della stringa.

20. a) Se $x > \sqrt{a} > 0$, dividendo a per x si ottiene $a : x < a : \sqrt{a} = \sqrt{a}$. b) Analogamente se $0 < x < \sqrt{a}$ allora $a : x > \sqrt{a}$; in entrambi i casi $\sqrt{a}$ sta tra x e $a : x$, quindi dista da x meno di quanto dista $a : x$: $|x - \sqrt{a}| < |x - a : x|$. c) L'algoritmo si ferma quando $|x - a : x| < \varepsilon$, e allora $|x - \sqrt{a}| < \varepsilon$ (se $x = \sqrt{a}$ l'errore è zero).

21. Traccia: se restano s elementi e il centrale è nel posto c, a sinistra ne restano $c - i$ e a destra $j - c$; poiché c è la parte intera di $(i + j) : 2$, entrambe le parti hanno al più $s : 2$ elementi. Partendo da 2^k, dopo ogni confronto negativo restano al più 2^{k-1}, 2^{k-2}, ..., e dopo k confronti al più $2^0 = 1$ elemento, che richiede un ultimo confronto: in tutto $k + 1$.

22. Alla passata i il minimo si cerca tra gli elementi dei posti da i a n, confrontando il minimo provvisorio con i $n - i$ elementi successivi, qualunque essi siano. Totale $(n - 1) + (n - 2) + \ldots + 1$; scrivendo la somma due volte, una al rovescio, e sommando termine a termine si ottengono $n - 1$ coppie di somma n: il doppio della somma è $n(n - 1)$, quindi la somma è $n(n - 1) : 2$.

23. 0,1 non è un numero macchina e gli arrotondamenti si accumulano: dopo dieci somme x vale 0,9999999999999999, diverso da 1; poi x supera 1 e non torna più a 1, quindi la condizione $x \neq 1$ resta sempre vera. Correzioni: **MENTRE** $x < 1 - 0{,}05$ (confronto con una tolleranza); oppure un contatore intero, **PER** i **DA** 0 **A** 10 **ESEGUI SCRIVI** $i : 10$.

24. Se lo sviluppo ha k cifre, $p : q = N : 2^k$ con N intero, cioè $p \cdot 2^k = N \cdot q$: q divide $p \cdot 2^k$ ed è primo con p, quindi divide 2^k ed è una potenza di 2. Viceversa, se $q = 2^k$ allora $p : q = p \cdot 2^{-k}$, e scrivendo p in base 2 basta spostare la virgola di k posti: lo sviluppo è finito. 0,1 = 1/10 è ridotta e 10 non è una potenza di 2: lo sviluppo è infinito (periodico), e un numero macchina ha un numero finito di cifre.

25. Sequenziale: 1 000 000 di confronti, cioè 1 secondo. Binaria: 20 confronti, cioè 20 milionesimi di secondo.

26. a) $100\,000 \cdot 99\,999 : 2 = 4\,999\,950\,000$ confronti: circa 5 secondi. b) $10^7 \cdot (10^7 - 1) : 2 \approx 5 \cdot 10^{13}$ confronti: circa 50 000 secondi, cioè circa 13,9 ore.

27. a) $4000 \cdot 3000 \cdot 3 = 36\,000\,000$ byte = 36 MB. b) 36 : 3,6 = 10: 10 a 1. c) $64 \cdot 10^9 : (3{,}6 \cdot 10^6) \approx 17\,777{,}8$: 17 777 foto (non compresse sarebbero 1777).

28. a) $44\,100 \cdot 16 \cdot 2 = 1\,411\,200$ bit al secondo; in 240 s: 338 688 000 bit = 42 336 000 byte ≈ 42,3 MB. b) $128\,000 \cdot 240 = 30\,720\,000$ bit = 3 840 000 byte = 3,84 MB. c) 1 411 200 : 128 000 ≈ 11,0: il file MP3 è circa 11 volte più piccolo.

29. a) $4 \cdot 7858 : 10\,000 = 3{,}1432$; errore assoluto ≈ $|3{,}1432 - 3{,}14159| \approx 0{,}0016$; errore relativo ≈ 0,0016 : 3,14159 ≈ 0,05%. b) $3{,}1416 \cdot 10\,000 : 4 = 7854$ punti. c) Perché il risultato è casuale: anche con moltissimi punti c'è una probabilità (piccolissima, ma non nulla) che la stima sia lontana da π; si può solo rendere molto probabile un errore piccolo.

30. $171! = 171 \cdot 170! \approx 171 \cdot 7{,}26 \cdot 10^{306} \approx 1{,}24 \cdot 10^{309}$, maggiore di $1{,}8 \cdot 10^{308}$: non è rappresentabile, mentre 170! sì. Poiché i fattoriali crescono, tutti quelli da 171 in poi producono traboccamento.

31. a) Le differenze sono circa 1; 0,167; 0,0049; $4{,}2 \cdot 10^{-6}$; $3{,}2 \cdot 10^{-12}$: scende sotto 10^{-10} al passo 4 ($x \approx 1{,}4142135624$). b) Al passo successivo la differenza diventa circa $2 \cdot 10^{-16}$ o 0 e non diminuisce più: si è raggiunto il limite di precisione dei numeri macchina (circa 16 cifre significative).

32. Traccia: con 1000 punti le stime oscillano tipicamente tra circa 3,04 e 3,24 (dipende dai numeri casuali); con 10 000 punti l'oscillazione si riduce, all'incirca a un terzo: per dimezzare l'oscillazione servono circa quattro volte più punti.

33. Stati q_0 (letti 0 modulo 3), q_1, q_2: $(q_0, 1) \to (1, \to, q_1)$; $(q_1, 1) \to (1, \to, q_2)$; $(q_2, 1) \to (1, \to, q_0)$; $(q_0, \square) \to$ (S, –, q_F); $(q_1, \square) \to$ (N, –, q_F); $(q_2, \square) \to$ (N, –, q_F). Su 1111: $q_0 \to q_1 \to q_2 \to q_0 \to q_1$, poi legge □ in q_1 e scrive N: 1111N (4 non è multiplo di 3).

34. a) Dopo k passi l'ampiezza è $1 : 2^k$; serve $2^k > 10^6$: $2^{19} = 524\,288$ non basta, $2^{20} = 1\,048\,576$ sì: 20 passi. b) Erone raggiunge una precisione migliore di 10^{-6} in 3 passi, ma vale solo per le radici quadrate; la bisezione funziona per qualunque equazione $f(x) = 0$ con un cambio di segno, per esempio di terzo grado.

35. $(q_0, 1) \to (1, \to, q_0)$; $(q_0, 0) \to (1, \to, q_1)$; $(q_1, 1) \to (1, \to, q_1)$; $(q_1, \square) \to (\square, \leftarrow, q_2)$; $(q_2, 1) \to (\square, -, q_F)$. L'idea: lo 0 separatore diventa 1 (gli 1 sono ora uno in più della somma) e l'ultimo 1 viene cancellato. Su 110111: 3 passi in q_0 (l'ultimo trasforma lo 0 in 1), 3 passi in q_1 sugli 1 del secondo addendo, 1 passo sulla casella vuota, 1 passo che cancella l'ultimo 1: 8 passi, nastro 11111 (cioè 5). Con 0111 (0 + 3) il primo passo trasforma lo 0 in 1 e il risultato è 111; con 110 (2 + 0) lo 0 diventa 1, la casella vuota segue subito e l'ultimo 1 viene cancellato: 11. Anche con il solo 0 (0 + 0) il nastro resta vuoto, cioè 0.