CADEC

Spediz. in abbonamento postale GR. II/70 L. 2.000

(...)

27 CORSO PRATICO COL COMPUTER

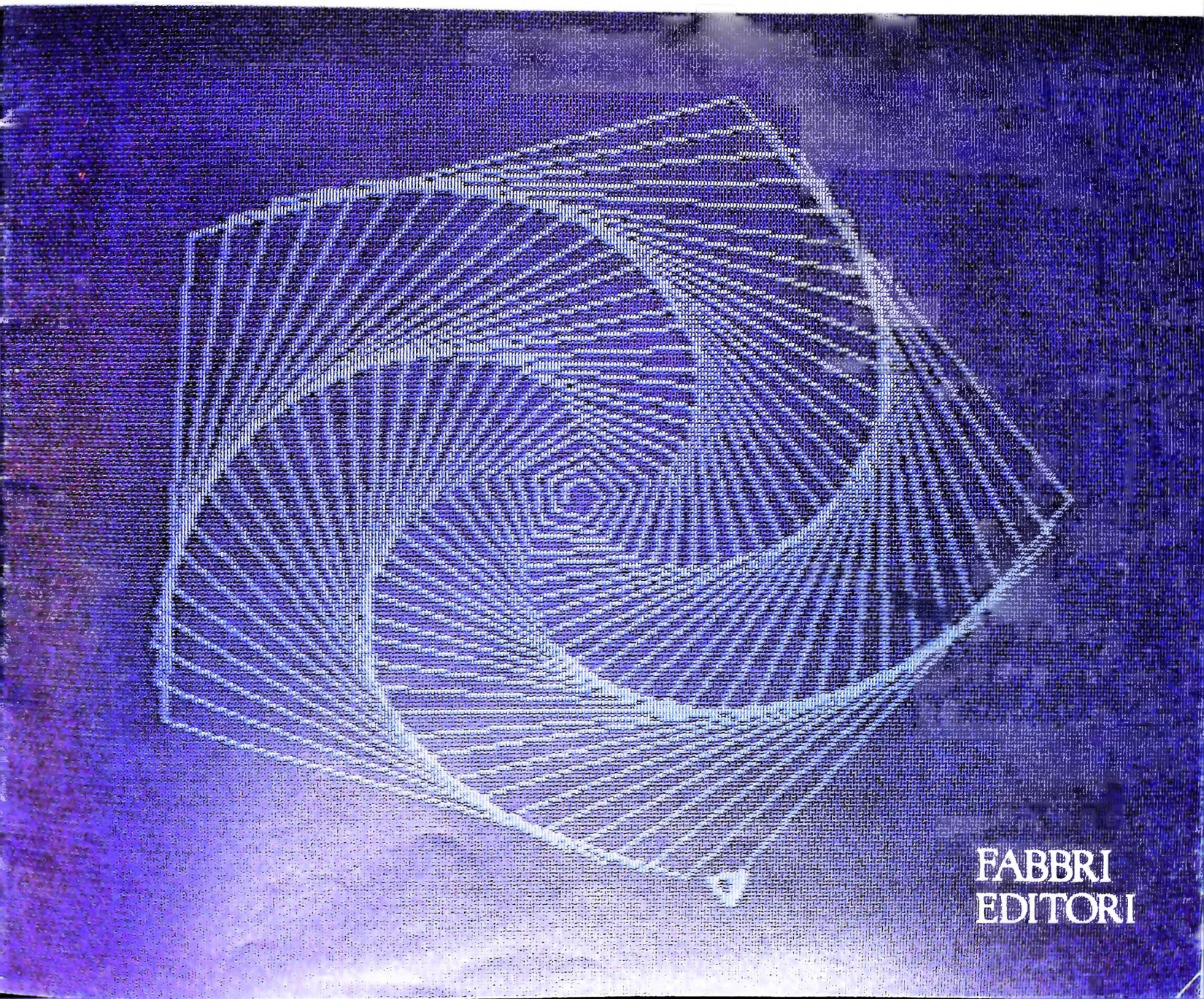
421792

è una iniziativa
FABBRI EDITORI

in collaborazione con
BANCO DI ROMA

e **OLIVETTI**

diretto da GIANNI DEGLI ANTONI



FABBRI EDITORI

Direttore dell'opera
GIANNI DEGLI ANTONI

Comitato Scientifico
GIANNI DEGLI ANTONI
Docente di Teoria dell'informazione, Direttore dell'Istituto di Cibernetica dell'Università degli Studi di Milano

UMBERTO ECO
Ordinario di Semiotica presso l'Università di Bologna

MARIO ITALIANI
Ordinario di Teoria e Applicazione delle Macchine Calcolatrici presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

MARCO MAIOCCHI
Professore incaricato di Teoria e Applicazione delle Macchine Calcolatrici presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

DANIELE MARINI
Ricercatore universitario presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

Curatori di rubriche
ADRIANO DE LUCA (Professore di Architettura del Calcolatori all'Università Autonoma Metropolitana di Città del Messico), GOFFREDO HAUS, MARCO MAIOCCHI, DANIELE MARINI, GIANCARLO MAURI, CLAUDIO PARMELLI, ENNIO PROVERA

Testi
GIANCARLO MAURI, Eidos (TIZIANO BRUGNETTI);
MARIA ALBERTA ALBERTI - GIANCARLO MAURI,
Etnoteam (ADRIANA BICEGO)

Tavole
Logical Studio Communication
Il Corso di Programmazione e BASIC è stato realizzato da Etnoteam S.p.A., Milano
Computergrafica è stato realizzato da Eidos, S.c.r l., Milano
Usare il Computer è stato realizzato in collaborazione con PARSEC S.N.C. - Milano

Direttore Editoriale
ORSOLA FENGHI

Redazione
CARLA VERGANI
LOGICAL STUDIO COMMUNICATION

Art Director
CESARE BARONI

Impaginazione
BRUNO DE CHECCHI
PAOLA ROZZA

Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÈ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

IMPARARE DAI PROGRAMMI O IMPARARE PROGRAMMANDO?

Il linguaggio LOGO, ambiente per "imparare a imparare".

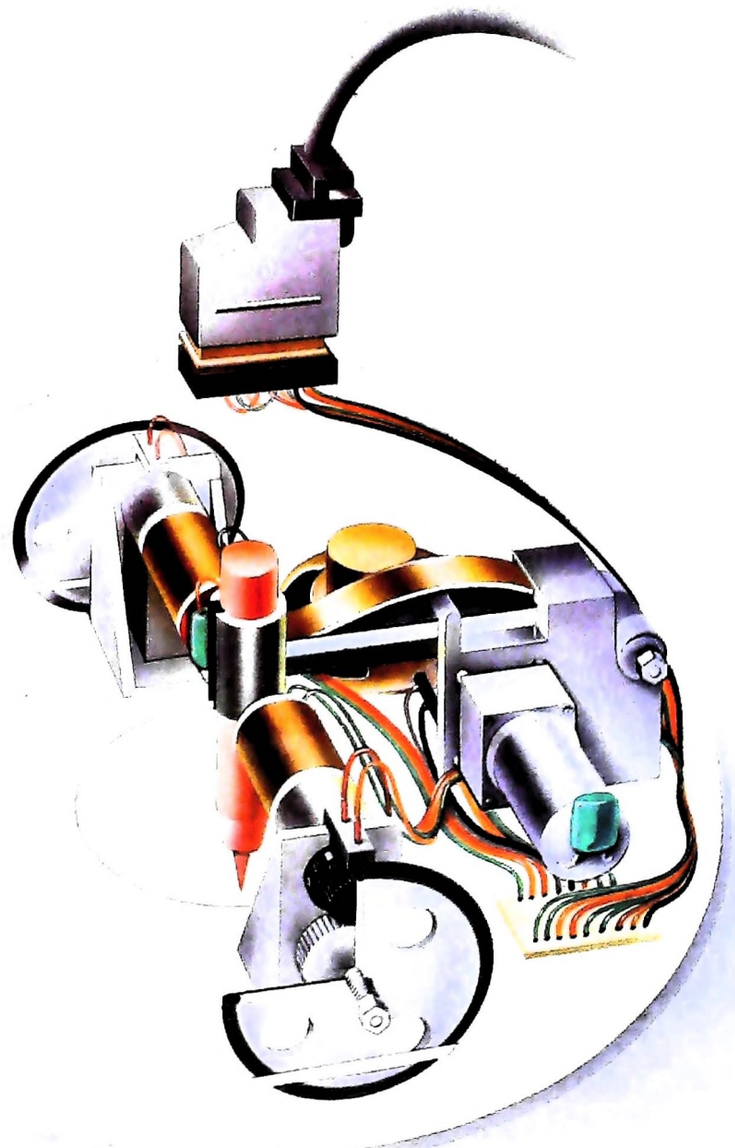
Parlando oggi di scuola ed elaboratore, non si può non parlare del linguaggio LOGO, che dopo quindici anni di esistenza "sotterranea" all'interno di una ristretta cerchia di ricercatori, sta oggi incontrando un crescente favore come linguaggio "pedagogico", soprattutto nella scuola di base.

Che cos'è LOGO?

LOGO nasce alla fine degli anni Sessanta presso il Laboratorio di Intelligenza Artificiale del Massachusetts Institute of Technology da una felice intuizione di Seymour Papert, che si propone di indicare possibilità di utilizzo dell'elaboratore alternative a quelle, da lui ritenute troppo rigide e demotivanti, basate su programmi per l'istruzione assistita da elaboratore.

Secondo Seymour Papert, è necessario spingersi oltre gli aspetti di uso puramente strumentale della macchina, ed esaminare invece a fondo "quello che l'elaboratore può apportare ai processi mentali, ... , esercitando la sua influenza sui nostri modi di pensare, anche quando siamo fisicamente allontanati da esso".

La "tartaruga" rappresenta un'interessante applicazione dell'elettronica nel campo della scuola. È un piccolo robot collegato con il microcomputer e azionato per mezzo di un linguaggio facilmente apprendibile il LOGO. Da semplici programmazioni, si potrà passare, in breve, alla programmazione di figure più complesse e articolate.



Procedure ricorsive

Si consideri la procedura seguente:

```
TO POLIGONI :LATO :ANGOLO
   FORWARD :LATO              ; avanza di lato
   RIGHT :ANGOLO              ; ruota di angolo
   POLIGONI :LATO :ANGOLO     ; ripeti il processo
```

(dove :LATO e :ANGOLO sono i parametri della procedura. Questo implica che LATO o ANGOLO sono i nomi formali di variabili di cui bisogna considerare i valori correnti, passati alla procedura all'atto dell'esecuzione).
Questo è un semplicissimo esempio di procedura ricorsiva; il comando:

```
POLIGONI 60 60
```

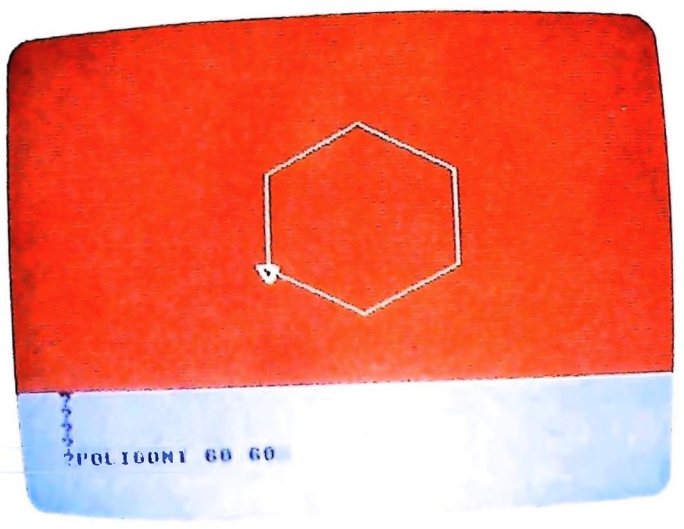
Per effetto del comando la tartaruga traccia un segmento di lunghezza 60, ruota di un angolo di 60 gradi, quindi ricomincia a eseguire lo stesso comando. Dopo 6 cicli, la tartaruga avrà disegnato un esagono regolare, riportandosi nella posizione di partenza (figura 1), ma continuerà a ripercorrere all'infinito (salvo un comando di interruzione dall'esterno) la figura già tracciata. Variando il valore dell'angolo si ottengono figure notevolmente diverse tra di loro (come quelle riprodotte nelle figure 2 e 3).

Più interessante è il programma, che chiameremo SPIRALI, sostanzialmente simile al programma POLIGONI, e che effettua la chiamata ricorsiva non con gli stessi input, ma incrementando ogni volta il lato di un valore prefissato INCR; questo ci consente anche di introdurre una semplice condizione di terminazione:

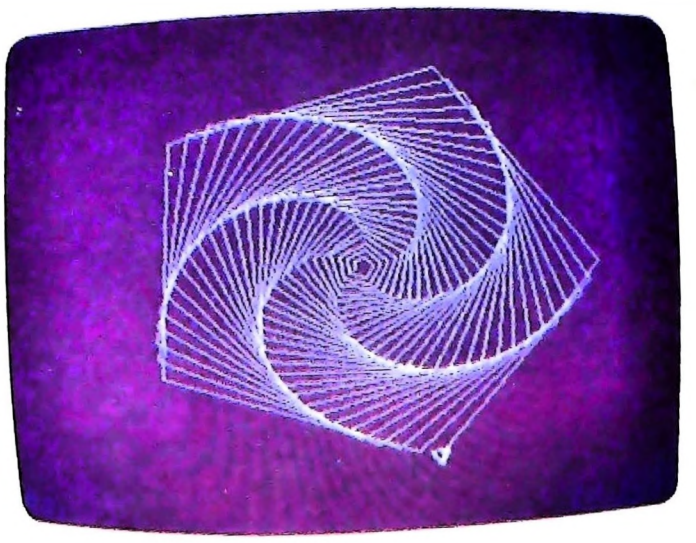
```
TO SPIRALI :LATO :ANGOLO :INCR
   IF :LATO > 150 THEN STOP                      ; termina se è il caso
   FORWARD :LATO                                 ; avanza di lato
   RIGHT :ANGOLO                                 ; ruota di angolo
   SPIRALI (:LATO + :INCR) :ANGOLO :INCR         ; ripeti il processo
```

La figura 4 mostra il disegno ottenuto con la procedura SPIRALI con input 5 71 1; dove il lato iniziale è 5 e viene incrementato di 1 ad ogni passo,

1

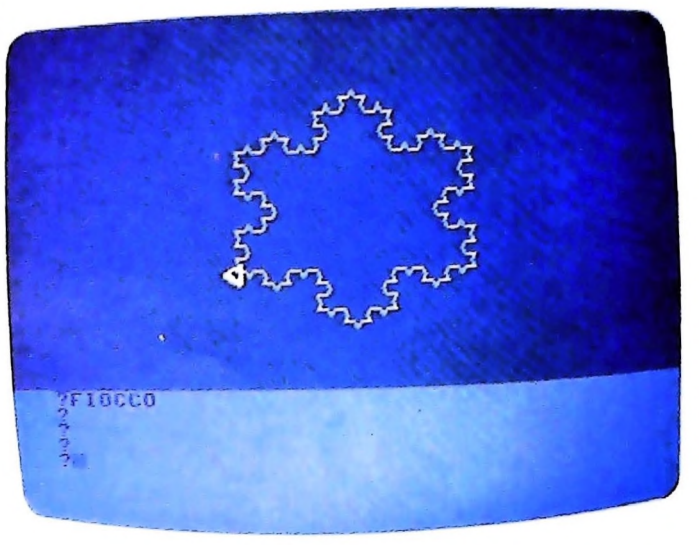


3



2

4



mentre 71 è l'angolo di rotazione. La spirale mostra un pattern quasi pentagonale poiché 71 è prossimo a 72, angolo di rotazione con cui otteniamo pentagoni in procedure tipo la POLIGONI.

Infine, per rendersi conto della potenza della ricorsione si consideri la seguente procedura:

```
TO FIOCCODINEVE
   SETUP
   REPEAT 3 [TRIADE 243 27 RIGHT 120]
```

che richiama a sua volta la procedura ricorsiva:

```
TO TRIADE :LATO :LIM
IF :LATO<:LIM THEN FORWARD :LATO STOP      ; avanza e termina
                                             se necessario
   TRIADE :LATO/3 :LIM                     ; richiama il processo
   LEFT 60                                 ; ruota a sinistra di 60
   TRIADE :LATO/3 :LIM                     ; ripeti il processo
   RIGHT 120                               ; ruota a destra di 120
   TRIADE :LATO/3 :LIM                     ; ripeti il processo
   LEFT 60                                 ; ruota a sinistra di 60
   TRIADE :LATO/3 :LIM                     ; e ripeti ancora
```

e la procedura di inizializzazione:

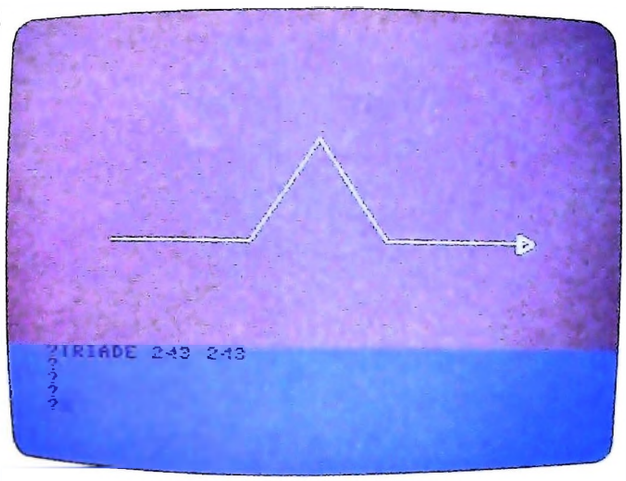
```
TO SETUP
   CLEARSCREEN       ; cancella lo schermo e posiziona la tartaruga
   PENUP             ; all'origine alza la penna
   LEFT 90           ; posiziona la tartaruga nella zona
   FORWARD 120       ; a sinistra dello schermo
   RIGHT 120         ; ruota a destra di 120
   PENDOWN           ; abbassa la penna
```

Per capire la procedura TRIADE si provi a eseguirla con carta e matita e si osservi la sequenza delle figure 5, 6 e 7 riprodotte a lato, ottenute con diversi input.
La prima immagine, TRIADE 243 243, rappresenta la prima curva della famiglia generata dalla procedura TRIADE, cui sono stati dati valori di input diversi.
Con input 243 243, per esempio, alle quattro diverse chiamate interne di TRIADE vengono passati i valori 243/3 come lato e 243 come limite; poiché il lato è minore del limite, la procedura termina subito dopo aver eseguito il solo avanzamento di 243/3, cioè di 81 passi.
La seconda e la terza fotografia mostrano, rispettivamente, le figure ottenute con la procedura TRIADE 243 81 e TRIADE 243 27. Le immagini sono più articolate poiché il processo base, illustrato nel paragrafo precedente, viene ripetuto su ogni lato della figura base un certo numero di volte, a seconda degli input che sono stati dati alla tartaruga in partenza, e che si possono quindi variare di volta in volta.

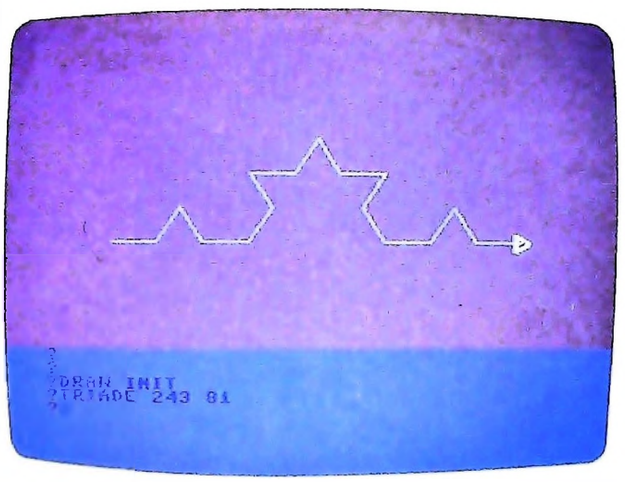
Una procedura assolutamente analoga alla TRIADE è la seguente:

```
TO QUADR :lato :limite
   IF :lato < : limite     THEN FORWARD :lato STOP
```

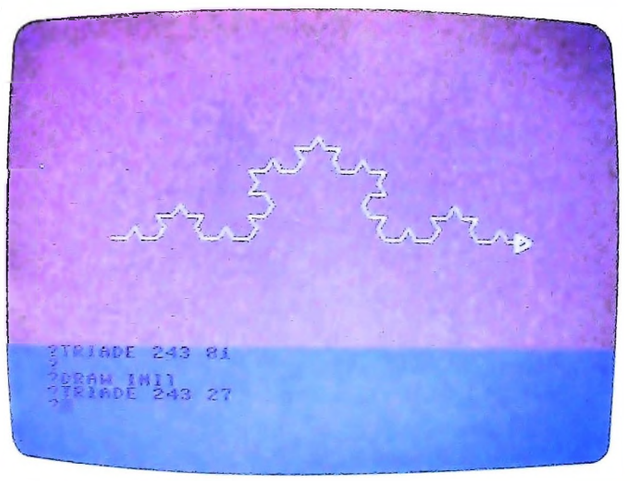
5



6

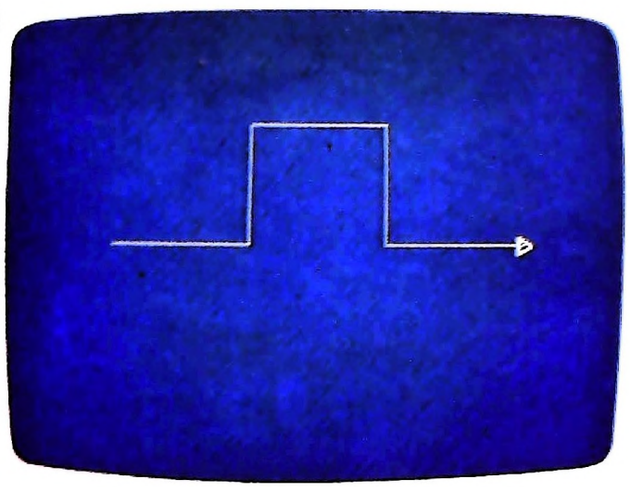


7



► segue

8

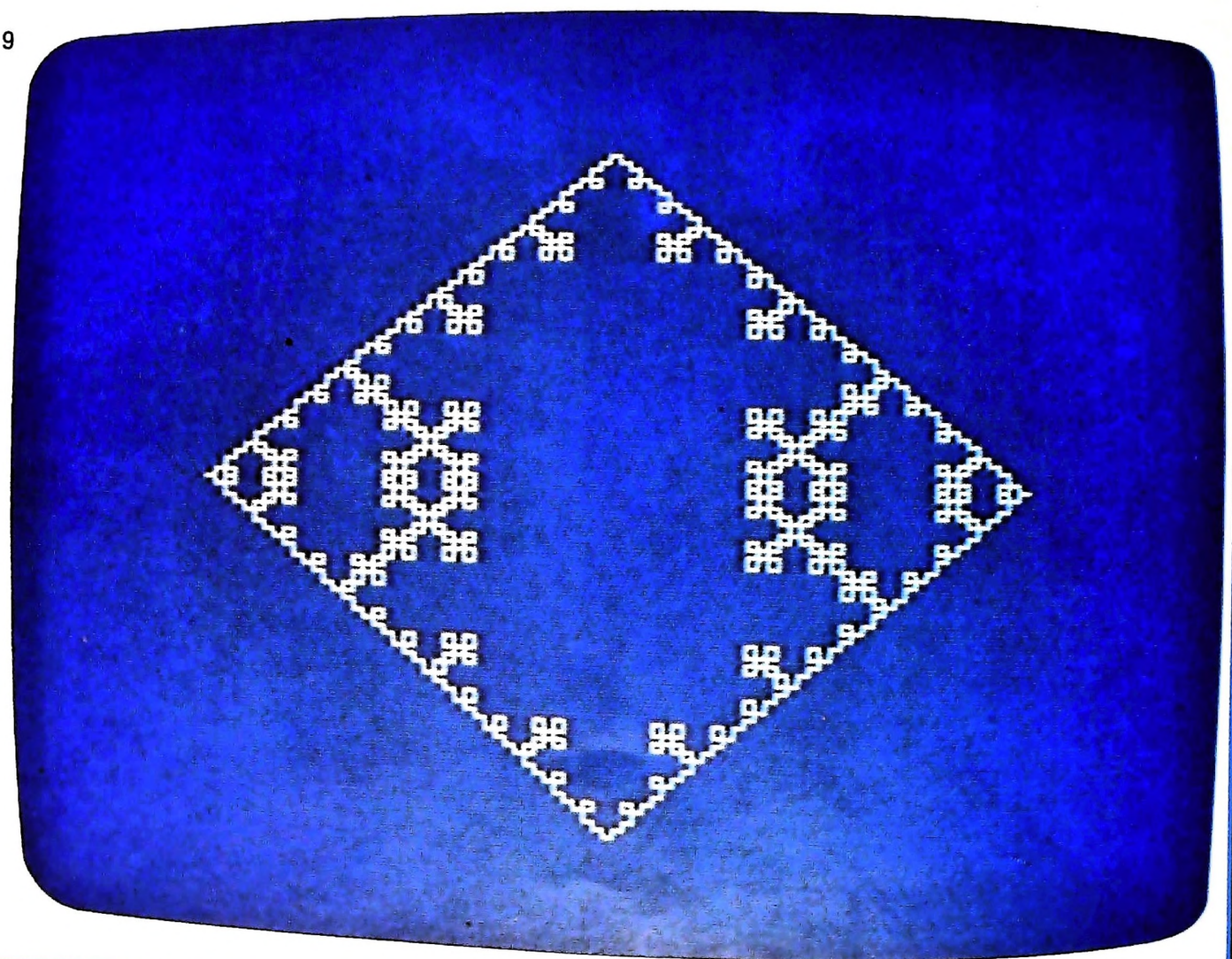


```
QUADR :lato/3 :limite     LEFT 90
QUADR :lato/3 :limite     RIGHT 90
QUADR :lato/3 :limite     RIGHT 90
QUADR :lato/3 :limite     LEFT 90
QUADR :lato/3 :limite
```

La curva più semplice appartenente alla famiglia di curve generata dalla procedura QUADR è, per esempio, ottenuta eseguendo QUADR 243 243 (figura 8).
Ma si possono ottenere immagini molto complesse, combinando curve di questa famiglia. Come nella procedura PIZZO (figura 9):

```
TO PIZZO
   SETUP
   REPEAT 2     QUADR 81   9     RIGHT 180
```

9



Per realizzare il suo ambizioso programma, Papert ha messo a punto una proposta educativa complessiva in cui confluiscono la teoria dello sviluppo cognitivo elaborata da J. Piaget e alcune delle idee elaborate nell'area dell'Intelligenza Artificiale, e ha progettato e realizzato un apposito linguaggio di programmazione, LOGO, semplice, ma potente, in grado di facilitare il contatto tra lo studente e la macchina, come strumento operativo essenziale di questa teoria.
LOGO quindi non è solo un linguaggio di programmazione, ma è soprattutto uno strumento che mette a disposizione dello studente un ambiente ricco di stimoli, in cui è possibile non tanto imparare una particolare disciplina, ma più in generale "imparare a imparare": citando H. Abelson, possiamo dire che "LOGO è il nome di una filosofia dell'educazione e di una famiglia di linguaggi di programmazione, in continua evoluzione, che aiuta a realizzarla".

La "geometria della tartaruga"

La "geometria della tartaruga" è certamente l'aspetto più immediatamente e più intensamente seducente di LOGO, ed è oggi ampiamente imitato anche da altri linguaggi di programmazione come Pascal, Forth, Pilot.
La "tartaruga" vera e propria è un robot meccanico controllato dal computer che risponde ad alcuni semplicissimi comandi primitivi:

```
FORWARD X       ; va avanti di X passi
BACKWARD X      ; va indietro di X passi
RIGHT X         ; gira a destra di X gradi
LEFT X          ; gira a sinistra di X gradi.
```

Più spesso, la tartaruga meccanica da pavimento è, per evidenti motivi economici e di maneggevolezza, sostituita da una simulazione sullo schermo.
Muovendosi, la tartaruga lascia una traccia su un foglio di carta steso sul pavimento o sullo schermo, realizzando così disegni anche complessi.
Il comando PENUP consente alla tartaruga di muoversi senza lasciare traccia, in modo da disegnare figure separate; per farle riprendere la matita, basterà usare il comando PENDOWN, mentre il comando PENERASE ha l'effetto di sostituire la matita con una gomma, in modo da cancellare tracce lasciate in precedenza.
Sul piano pedagogico, la geometria della tartaruga presenta almeno due aspetti decisamente positivi, ambedue legati alla "naturalezza" dei comandi primitivi.
Anzitutto, questi comandi sono immediatamente riferibili dal bambino al proprio corpo, alla propria percezione dello spazio. Partendo dalle proprie intuizioni geometriche inconsce e dalla conoscenza del proprio corpo, in relazione allo spazio circostante, il bambino guida la tartaruga immedesimandosi in essa, e così facendo esplicita e razionalizza tali intuizioni. Certo questo processo, che aiuta il bambino a divenire cosciente dei propri movimenti sul piano, costringe anche a decentrare l'attenzione da se stesso alla tartaruga, cosa non sempre facile per i più piccoli, cui si aggiunge la difficoltà della ulteriore astrazione dovuta alla trasposizione del piano orizzontale nel piano verticale dello schermo, nel caso della tartaruga simulata.
In secondo luogo, con la grafica il bambino trae immediata gratificazione dalla possibilità di realizzare progetti relativamente complessi senza doversi sottoporre a un lungo e noioso periodo di addestramento per impratichirsi del linguaggio. Anche l'*interattività* di LOGO, con la visualizzazione immediata dell'effetto di un comando grafico, contribuisce a rendere naturale il rapporto tra il bambino e la macchina, evitandogli la difficoltà di dover definire programmi solo per poter visualizzare semplici costruzioni.
La grafica infine costituisce un ottimo strumento per evidenziare lo svolgimento del programma e renderlo trasparente al bambino. Infatti a ogni istruzione corrisponde un effetto grafico visibile, ed è quindi possibile seguire passo passo lo svolgersi del programma.
Non si pensi tuttavia che LOGO sia un linguaggio solo per bambini; in realtà, esso è un linguaggio estremamente potente, che in un certo senso cresce con l'allievo, che in ogni momento, dalla scuola elementare all'università, può usarlo al livello di potenza adeguato alle sue capacità.

Programmare in LOGO

In LOGO è possibile definire un programma, utilizzando il comando TO (è la preposizione che precede il modo infinito di un verbo in inglese, traducibile con "PER FARE") seguito dal nome prescelto, e farlo eseguire semplicemente invocandone il nome. All'interno di ciascun programma se ne possono inoltre richiamare altri, chiamandoli per nome. Questo rende molto semplice la costruzione di una figura di arbitraria complessità, sia che si voglia procedere in modo top-down, se si ha già un progetto preelaborato, sia che si voglia procedere in modo bottom-up, se, a partire da programmi già fatti, si vogliano elaborare nuove complicate immagini.
Il bambino, cercando di far realizzare alla tartaruga un certo disegno, impara a formulare le proprie idee in modo preciso, a smontare problemi complessi, suddividendoli in problemi più piccoli, risolvibili direttamente, a correggere e migliorare progressivamente la procedura di soluzione, impara insomma una tecnica molto generale per risolvere problemi, e la impara giocando.
Inoltre impara a comunicare il procedimento di soluzione del problema in un linguaggio formale, ancorché intuitivo, e si abitua a manipolare simboli, distinguendo tra il segno e il significato.
All'interno di un programma è possibile ripetere n volte una sequenza di istruzioni, con il comando:

REPEAT n [⟨lista istr.⟩];

e anche alterare la sequenza di esecuzione con l'istruzione condizionale:
IF ⟨condizione⟩ THEN ⟨istr. 1⟩ ELSE ⟨istr. 2⟩.

Se la condizione è vera, si esegue la prima istruzione; nel caso contrario si esegue la seconda, sempre, ovviamente, che questa sia presente.
Benché questi siano i soli costrutti di controllo, forniti come primitivi, è possibile definirne di nuovi, che possono poi venire usati esattamente come i primitivi. Per esempio è possibile definire un costrutto WHILE [⟨condizione⟩] [⟨lista istr.⟩] come segue:

```
TO WHILE: condizione: istruzione
   TEST RUN: condizione          ; test sulla condizione
   IFFALSE [stop]                ; termina se è falsa
   RUN: istruzione               ; esegui l'istruzione
   WHILE: condizione: istruzione; ripeti il processo
```

Ma certamente il meccanismo più potente che LOGO mette a disposizione è la *ricorsione*, cioè la possibilità di richiamare all'interno di un programma il programma stesso. Questo concetto è già noto dalla matematica: si pensi alla definizione del fattoriale di un numero (fattoriale di n è definito: n *

Progetti grafici

Supponiamo di aver preparato in precedenza i seguenti programmi che costruiscono un quadrato e un triangolo equilatero di lati variabili, e che traslano a destra e a sinistra la tartaruga, senza alterarne la direzione e senza lasciare traccia sullo schermo:

```
TO QUADRATO :lato
   REPEAT 4 [FORWARD :lato RIGHT 90]

TO TRIANGOLO :lato
   REPEAT 3 [FORWARD :lato RIGHT 120]

TO TRASLAD :quanto
   RIGHT 90
   PENUP FORWARD :quanto PENDOWN
   LEFT 90

TO TRASLAS :quanto
   LEFT 90
   PENUP FORWARD :quanto PENDOWN
   RIGHT 90
```

A partire da queste semplici figure regolari e dalle procedure di servizio, per traslare la tartaruga, possiamo impegnarci nel costruire una casa di dimensione variabile. Il programma "casa" potrà essere composto da due sottoprogrammi, che disegnino separatamente la facciata ed il tetto.

```
TO CASA :dim
   FACCIATA :dim
   TETTO :dim
```

Evidenziandone le componenti, come abbiamo fatto per la "casa", a sua volta il programma "facciata" si può scomporre nel contorno (disegnato dal programma "quadrato"), nella "porta" e nelle "finestre". Il "tetto" invece, sarà un "triangolo" a meno di un riposizionamento iniziale, che tenga conto di dove si trova la tartaruga dopo aver disegnato la "facciata", e uno finale per riportare la tartaruga nella posizione in cui era all'inizio della procedura.

```
TO FACCIATA :dim
   QUADRATO :dim
   PORTA :dim/3
   PIANOALTO (2 * :dim)/3
   FINESTRE :dim/5
   PIANOTERRENO (2 * :dim)/3

TO PORTA :dim
   TRASLAD :dim
   RIGHT 30
   TRIANGOLO :dim
   LEFT 30
   TRASLAS :dim

TO FINESTRE :lato
   REPEAT 2 [TRASLAD :lato RIGHT 30 TRIANGOLO :lato LEFT 30 TRA-
      SLAD :lato]
   TRASLAS 4 * :lato

TO TETTO :dimensione
   PIANOALTO :dimensione
   RIGHT 30
   TRIANGOLO :dimensione
   LEFT 30
   PIANOTERRENO :dimensione
```

Occorre ancora definire le procedure di servizio "pianoalto" e "pianoterreno", che non sono niente altro che sinonimi delle parole primitive "forward" e "back", e che vengono usate dalle procedure "finestre" e "tetto" per posizionare la tartaruga alla giusta altezza.

```
TO  PIANOALTO :altezza
   FORWARD :altezza

TO PIANOTERRENO :altezza
   BACK :altezza
```

Si noti che la procedura "facciata", per la modularità delle singole sottoprocedure, potrebbe essere riscritta alterando l'ordine delle chiamate delle singole sottoprocedure, senza modificare il risultato finale, oppure introducendo più piani alla casa:

```
TO FACCIATA.BIS :lato
   PIANOALTO (2 * :lato)/3
   FINESTRE :lato/5
   PIANOTERRENO (2 * :lato)/3
   PORTA :lato/3
   QUADRATO :lato

TO FACCIATA.TRIS :lato
   QUADRATO :lato
   PORTA :lato/3
   REPEAT 2 [PIANOALTO :lato/3 FINESTRE :lato/5]
   PIANOTERRENO (2 * :lato)/3
```

Ora siamo in grado di disegnare un intero paese con il programma:

```
TO PAESE
   CLEARSCREEN
   TRASLAS 100
   CASA 60 TRASLAD 60
   CASA 80 TRASLAD 110
   CASA 40
```

Il progetto può essere arricchito con alberi, un sole, un volo di uccelli.

Se la condizione è vera, si esegue la prima istruzione; nel caso contrario si esegue la seconda, sempre, ovviamente, che questa sia presente.
Benché questi siano i soli costrutti di controllo, forniti come primitivi, è possibile definirne di nuovi, che possono poi venire usati esattamente come i primitivi. Per esempio è possibile definire un costrutto WHILE [⟨condizione⟩] [⟨lista istr.⟩] come segue:

```
TO WHILE: condizione: istruzione
   TEST RUN: condizione          ; test sulla condizione
   IFFALSE [stop]                ; termina se è falsa
   RUN: istruzione               ; esegui l'istruzione
   WHILE: condizione: istruzione; ripeti il processo
```

Ma certamente il meccanismo più potente che LOGO mette a disposizione è la *ricorsione*, cioè la possibilità di richiamare all'interno di un programma il programma stesso. Questo concetto è già noto dalla matematica: si pensi alla definizione del fattoriale di un numero (fattoriale di n è definito: n *

Progetti grafici

Supponiamo di aver preparato in precedenza i seguenti programmi che costruiscono un quadrato e un triangolo equilatero di lati variabili, e che traslano a destra e a sinistra la tartaruga, senza alterarne la direzione e senza lasciare traccia sullo schermo:

```
TO QUADRATO :lato
   REPEAT 4 [FORWARD :lato RIGHT 90]

TO TRIANGOLO :lato
   REPEAT 3 [FORWARD :lato RIGHT 120]

TO TRASLAD :quanto
   RIGHT 90
   PENUP FORWARD :quanto PENDOWN
   LEFT 90

TO TRASLAS :quanto
   LEFT 90
   PENUP FORWARD :quanto PENDOWN
   RIGHT 90
```

A partire da queste semplici figure regolari e dalle procedure di servizio, per traslare la tartaruga, possiamo impegnarci nel costruire una casa di dimensione variabile. Il programma "casa" potrà essere composto da due sottoprogrammi, che disegnino separatamente la facciata ed il tetto.

```
TO CASA :dim
   FACCIATA :dim
   TETTO :dim
```

Evidenziandone le componenti, come abbiamo fatto per la "casa", a sua volta il programma "facciata" si può scomporre nel contorno (disegnato dal programma "quadrato"), nella "porta" e nelle "finestre". Il "tetto" invece, sarà un "triangolo" a meno di un riposizionamento iniziale, che tenga conto di dove si trova la tartaruga dopo aver disegnato la "facciata", e uno finale per riportare la tartaruga nella posizione in cui era all'inizio della procedura.

```
TO FACCIATA :dim
   QUADRATO :dim
   PORTA :dim/3
   PIANOALTO (2 * :dim)/3
   FINESTRE :dim/5
   PIANOTERRENO (2 * :dim)/3

TO PORTA :dim
   TRASLAD :dim
   RIGHT 30
   TRIANGOLO :dim
   LEFT 30
   TRASLAS :dim

TO FINESTRE :lato
   REPEAT 2 [TRASLAD :lato RIGHT 30 TRIANGOLO :lato LEFT 30 TRA-
      SLAD :lato]
   TRASLAS 4 * :lato

TO TETTO :dimensione
   PIANOALTO :dimensione
   RIGHT 30
   TRIANGOLO :dimensione
   LEFT 30
   PIANOTERRENO :dimensione
```

Occorre ancora definire le procedure di servizio "pianoalto" e "pianoterreno", che non sono niente altro che sinonimi delle parole primitive "forward" e "back", e che vengono usate dalle procedure "finestre" e "tetto" per posizionare la tartaruga alla giusta altezza.

```
TO  PIANOALTO :altezza
   FORWARD :altezza

TO PIANOTERRENO :altezza
   BACK :altezza
```

Si noti che la procedura "facciata", per la modularità delle singole sottoprocedure, potrebbe essere riscritta alterando l'ordine delle chiamate delle singole sottoprocedure, senza modificare il risultato finale, oppure introducendo più piani alla casa:

```
TO FACCIATA.BIS :lato
   PIANOALTO (2 * :lato)/3
   FINESTRE :lato/5
   PIANOTERRENO (2 * :lato)/3
   PORTA :lato/3
   QUADRATO :lato

TO FACCIATA.TRIS :lato
   QUADRATO :lato
   PORTA :lato/3
   REPEAT 2 [PIANOALTO :lato/3 FINESTRE :lato/5]
   PIANOTERRENO (2 * :lato)/3
```

Ora siamo in grado di disegnare un intero paese con il programma:

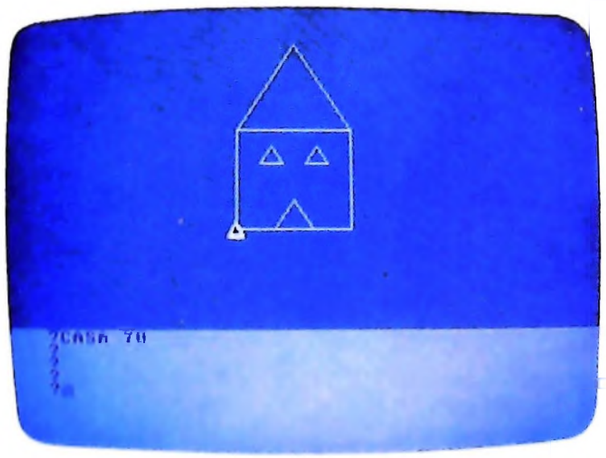
```
TO PAESE
   CLEARSCREEN
   TRASLAS 100
   CASA 60 TRASLAD 60
   CASA 80 TRASLAD 110
   CASA 40
```

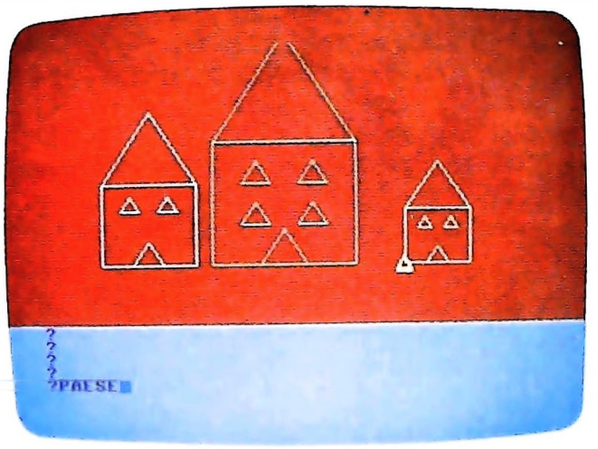
Il progetto può essere arricchito con alberi, un sole, un volo di uccelli.

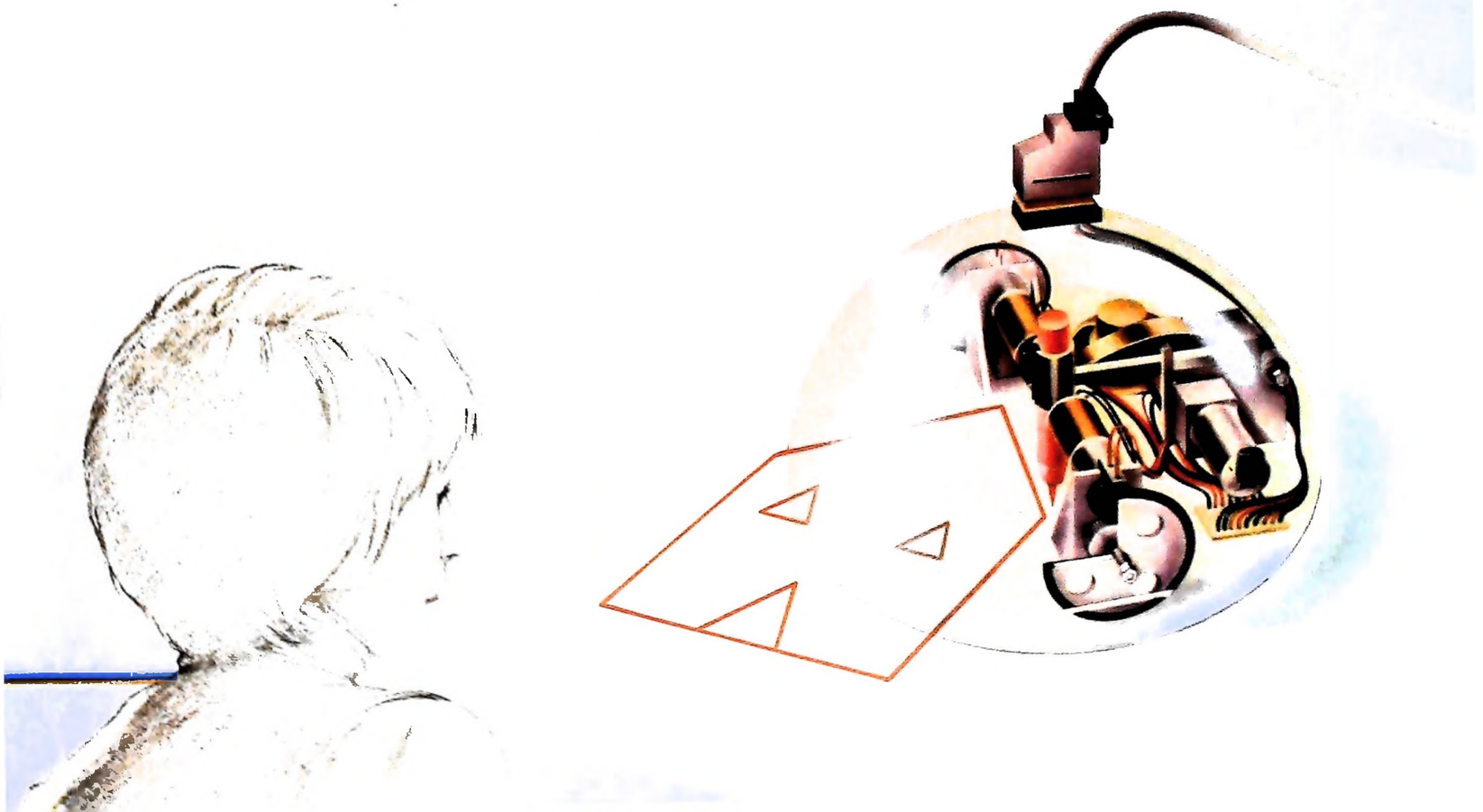
fattoriale di n-1, e fattoriale di 0 è definito 1), ma è anche presente in filastrocche da bambini, come "c'era una volta un re, che disse alla sua serva: raccontami una storia. La serva raccontò: c'era una volta un re,...".
L'uso della ricorsione rende molto semplice scrivere programmi per diverse applicazioni, in particolare in quei casi in cui si vuol risolvere un problema che può essere ridotto a un problema dello stesso tipo con dati di dimensione inferiore: esempi classici sono giochi come la torre di Hanoi o il nim, oltre a numerosi problemi tipici dell'Intelligenza Artificiale.

Liste e parole

LOGO, come il linguaggio LISP da cui discende direttamente, dispone di primitive per manipolare liste e parole.
Una parola è una sequenza di caratteri, mentre una lista è una collezione di oggetti che possono essere parole o altre liste. Le parole vanno indicate premettendo le virgolette: "PAROLA; gli elementi di una lista vengono invece racchiusi tra parentesi quadre.
LOGO dispone di alcune primitive che consentono di estrar-







re parti di una parola.
FIRST "PAROLA e LAST "PAROLA estraggono rispettivamente il primo e l'ultimo carattere di "PAROLA; BUTFIRST "PAROLA e BUTLAST "PAROLA forniscono in uscita la parola data privata del primo e dell'ultimo carattere rispettivamente.
Le stesse primitive operano anche su liste, e si riferiscono rispettivamente al primo e all'ultimo elemento di una lista.
Ad esempio, BUTFIRST [LA MIA CASA È BELLA] dà in uscita la lista [MIA CASA È BELLA].
Altre primitive, WORD e SENTENCE, consentono invece di concatenare parole o liste ottenendo parole o liste più complesse. Ad esempio, WORD "SCHIACCIA "SASSI genera la parola SCHIACCIASASSI; analogamente SENTENCE [LA MIA CASA] [È BELLA] produce la lista [LA MIA CASA È BELLA].
Poiché in LOGO i programmi, come i dati, sono rappresentati da liste, di istruzioni ovviamente, queste primitive consentono anche di scrivere programmi che manipolano, modificano, costruiscono liste di istruzioni, a cui si può poi dare un nome (con la primitiva DEFINE) e di cui si può richiedere l'esecuzione (con RUN come nell'esempio della procedura WHILE).
Questa possibilità di trattare i programmi come dati manipolabili da altri programmi e di decidere di farli eseguire al momento opportuno è ciò che ha fatto del linguaggio LISP (progenitore di LOGO) il linguaggio privilegiato nelle applicazioni di Intelligenza Artificiale.

Per saperne di più

La filosofia dell'educazione da cui nasce LOGO è presentata in modo chiaro e avvincente da Seymour Papert nel libro *Mindstorms*, pubblicato in Italia da Emme Edizioni.
Per chi volesse invece imparare a programmare in LOGO, oltre ai manuali che accompagnano i dischetti di LOGO esistono numerosi testi in inglese. Tra questi, possiamo citare *LOGO for the Apple II* e *Apple LOGO* di H. Abelson, pubblicati da McGraw Hill, che sono due versioni dello stesso testo adattate ai due principali "dialetti" di LOGO.
In italiano è di recente pubblicazione il libro *LOGO: ALI PER LA MENTE*, edito da Edizioni Elettroniche VIFI-Mondadori.
Applicazioni molto avanzate della geometria della tartaruga, fino alla simulazione dello spazio-tempo curvo e di fenomeni relativistici, sono contenuti in *Turtle geometry* di H. Abelson e A. di Sessa, pubblicato da MIT Press.

La manipolazione di testi

Vogliamo realizzare un programma di crittografia, seguendo le seguenti semplici regole: se una parola inizia per vocale, allora le appendiamo la sillaba "se", se la parola inizia per consonante, allora rimuoviamo la consonante e l'appendiamo alla fine della parola e rielaboriamo la parola così ottenuta.
Chiamiamo "codifica" il programma di crittografia, che codifica una intera frase; il programma potrà essere usato per esempio con l'istruzione:

```
PRINT CODIFICA [questa storia non mi piace]
```

e otterremo in uscita la frase crittografata:

```
uestaqse oriastse onnse imse iacepse
```

La procedura "codifica" è un esempio di uso della tecnica della ricorsione ed esemplifica la ricchezza di primitive di LOGO per la manipolazione di stringhe di caratteri (parole) e di liste (le frasi).

```
TO CODIFICA :frase
   IF :frase = [ ] THEN OUTPUT
   OUTPUT SENTENCE (COD.PAROLA FIRST :frase) (CODIFICA BUTFIRST
   :frase)

TO COD.PAROLA :parola
   IF MEMBERP FIRST :parola [A E I O U] THEN OUTPUT WORD :parola
   "SE OUTPUT COD.PAROLA (WORD BUTFIRST :parola FIRST :parola)
```

La codifica della singola parola è affidata alla procedura COD.PAROLA, che controlla se la prima lettera della parola è una vocale.
Questo compito è svolto utilizzando le primitive FIRST, che seleziona la prima lettera, e MEMBERP, che darà il valore "vero" se il suo primo input appartiene alla lista, che le viene passata come secondo input.
Se la prima lettera è una vocale allora la COD.PAROLA produce in uscita la parola data + la sillaba "se" (WORD :parola "se); altrimenti la consonante viene rimossa dalla prima posizione e appesa alla fine della parola (WORD BUTFIRST :parola FIRST :parola); la parola così ottenuta viene rielaborata dalla COD.PAROLA. Il processo, quindi, termina solo quando la prima lettera è una vocale.

Sapendo ora codificare una singola parola, possiamo codificare un'intera frase. Passiamo la frase data in input alla procedura CODIFICA, che produrrà in uscita la frase che si ottiene componendo in una frase la codifica della prima parola della frase originale, con la frase codificata prodotta dalla procedura stessa, applicata alla frase originale tranne la prima parola (CODIFICA BUTFIRST :frase).
In conclusione la procedura CODIFICA svuota la frase parola per parola ricorsivamente; fino a che la frase è vuota, nel qual caso produce come output la lista vuota, e ricompone poi la codifica delle singole parole in una nuova lista, cioè la frase crittografata.

Scrivere programmi ricorsivi per risolvere problemi significa riconoscere la natura intimamente ricorsiva del problema stesso. Per esempio, nel caso del nostro problema, crittografare una frase significa crittografare una parola e poi crittografare la frase data tranne la prima parola. L'importante è definire una condizione di uscita dal processo, nel nostro caso quando la frase è diventata vuota.

Simulazione di una lista per l'ordinamento di N valori

Riprendiamo in questa lezione il concetto di lista introdotto nella precedente ed esaminiamo il procedimento di costruzione di un programma che effettua l'ordinamento di N valori con l'uso di una lista. Poiché però il BASIC non dispone di strutture di dati dinamiche simuleremo tale struttura con l'uso di array. La simulazione di strutture di dati dinamiche tramite strutture statiche è, come si vedrà, un argomento di non limitata complessità. Tuttavia tale complessità può essere affrontata e ridotta grazie all'uso di strumenti di analisi e di disegno di programmi, come il TOP DOWN, appositamente pensati per graduare il livello di complessità dei problemi affrontandoli uno alla volta e da un livello generale scendendo poi gradatamente al particolare. Sarà dunque un duplice vantaggio affrontare le difficoltà che la lettura di queste pagine può proporre: da un lato per i nuovi argomenti introdotti e ancor più per gli aspetti metodologici. Veniamo dunque alla costruzione del programma.

Il programma BASIC

Il programma sarà dunque costituito di una parte che effettua l'inserimento dei valori nella lista e di una parte che la scandisce per visualizzare i valori ordinati.
Seguiamo la costruzione con un procedimento TOP DOWN:

```
10 INPUT "Quanti valori";N
20 DIM V(N),P(N)
30 GOSUB 100 'Inserimento
40 GOSUB 1000 'Scansione
50 END
```

Quello che abbiamo scritto non è che lo scheletro del programma, che mostra però in modo immediato quali sono le funzioni che vogliamo realizzare e soprattutto rispecchia già una suddivisione logica tra le due funzioni fondamentali di inserimento ordinato e scansione.
Si noti che è stata introdotta una nuova istruzione (END) che determina la fine dell'esecuzione.
Dobbiamo ora realizzare i due sottoprogrammi di inserimento e di scansione. Con questa suddivisione abbiamo dimezzato la difficoltà iniziale e possiamo ora porci il problema della costruzione della lista e della scansione in modo separato.

Costruzione del modulo di INSERIMENTO

Prima di definire la struttura del modulo riassumiamo i tre casi di inserimento precedentemente descritti:
• Inserimento in testa alla lista: si verifica quando l'elemento da inserire è il minore tra quelli già inseriti ed è quindi in particolare più piccolo del valore contenuto nella radice;
• inserimento in posizione intermedia: si verifica quando l'elemento inserito ha un

predecessore e un successore tra gli elementi esistenti nella lista;
• inserimento in coda: si verifica quando l'elemento inserito è il maggiore tra i già presenti e in particolare è più grande dell'ultimo elemento della lista individuato dal puntatore di fine lista.

Osserviamo dunque lo schema dell'algoritmo di inserimento ordinato:

```
V(I):= valore ' Inserimento nella prima posizione libera
J:=PTRADICE;
EXECUTE UNTIL INTESTA, INS, INSCODA;
        EVENT INSTESTA IF V (I)⟨V(J) AND     J=PTRADICE;
        EVENT INS IF V(I)⟨V(J)
        EVENT INSCODA IF P (J)=-1
        K:=J;
        J:=P(J)
THENCASE
        WHEN INTESTA
              PTRADICE:=I; Radice all'elemento inserito
              P(I):=J ' Puntatore elem. inserito a quello che era primo
        WHEN INS
              P(I):=J; ' Puntatore elem. ins. a quello trovato
              P(K):=I ' Puntatore del precedente a quello inserito
        WHEN INSCODA
              P(I):=-1 ' Puntatore vuoto
              P(J):=I ' punt. dell'ex ultimo a quello inserito
ENDEXUTE.
I:=I+1
```

Si noti che nel caso dell'inserimento intermedio è stato necessario tener memoria del puntatore all'elemento immediatamente inferiore, che è stato "salvato" nella variabile K. Poiché però non è possibile sapere qual è l'elemento immediatamente inferiore se non quando si individua il successore dell'ultimo inserito, ad ogni scansione di elemento di lista siamo costretti a "ricordare" il puntatore all'elemento prima di assegnare all'indice di scansione il valore del puntatore all'elemento seguente.

Il modulo inserimento in BASIC

Vediamo ora la realizzazione in BASIC dell'algoritmo sopra presentato:

```
100 ' Inserimento
103 ' Crea radice
105 INPUT "Valore";V(1)
107 LET P(1)=-1
110 LET R=1 ' Valore iniziale puntatore radice
115 LET I=2
120 ' While I<=N do
122 IF I>N GOTO 165
125 ' Begin
130 INPUT "Valore";V(I)
```

```
140 GOSUB 250 'Posizionamento indici
150 LET I=I+1
155 ' End
160 GOTO 120
165 '
170 RETURN
```

Come si può facilmente notare il sottoprogramma:
- crea la radice della lista;
- realizza il ciclo iterativo per l'inserimento dei valori nella lista e richiama a sua volta un sottoprogramma che provvede all'operazione più complessa di posizionamento dei puntatori.

In tal modo, il sottoprogramma ottenuto mantiene dimensioni limitate ed è quindi più facilmente leggibile; la definizione dell'algoritmo e quindi del programma in moduli di livello via via più basso è inoltre un metodo per "decomporre" la complessità del problema: abbiamo infatti così isolato la parte più complessa dell'algoritmo che consiste nella gestione dei puntatori. Da un punto di vista metodologico abbiamo anche isolato "fisicamente" la parte più delicata del programma: in questo modo ci sarà anche più facile affrontare i problemi di "debugging", cioè i problemi di messa a punto del programma stesso. In caso infatti di cattivo funzionamento avremo immediatamente isolate le parti di codice che con maggiore probabilità possono contenere errori.

Vediamo dunque il sottoprogramma che gestisce il posizionamento dei puntatori:

```
250 'Posizionamento indici
255 LET J=R
260 ' Exec until intesta,ins,inscoda
270 ' Event instesta if V(I)<V(J) and J=R
280 IF V(I)<V(J) AND J=R GOTO 330
285 ' Event ins if V(I)<V(J)
290 IF V(I)<V(J) GOTO 400
295 'Event inscoda if P(J)=-1
298 IF P(J)=-1 GOTO 370
300 LET K=J
305 LET J=P(J)
310 ' Thencase
320 GOTO 260
325 ' Thencase
330 ' When instesta
340 LET R=I
350 LET P(I)=J
360 GOTO 450
370 ' When inscoda
375 LET P(I)=P(J)
380 LET P(J)=I
390 GOTO 450
400 ' When ins
410 LET P(I)=J
```

```
420 LET P(K)=I
450 ' Endexec
460 RETURN
```

Si noti l'uso della struttura di ZAHN, che è stata ancora una volta simulata, non essendo disponibile nel linguaggio e spiegata attraverso opportuni commenti.

Visualizzazione

Passiamo ora al passo successivo che consiste nella visualizzazione ordinata dei valori inseriti.
Procediamo come segue:
- prendiamo il puntatore alla radice e visualizziamo l'elemento da esso puntato;
- prendiamo quindi il valore contenuto nella corrispondente posizione dell'array dei puntatori e visualizziamo il valore da esso puntato;
- continuiamo così fino a quando troviamo l'indicatore di fine lista.

Vediamo lo schema dell'algoritmo:

```
ISCAN:=PIRADICE;
WHILE P(ISCAN)⟨⟩—1 DO
  BEGIN
        visualizza V(ISCAN)
        ISCAN:=P(ISCAN)
  END;
```

che tradotto in BASIC risulta così:

```
1000 ' Scansione per ordinamento
1010 IF R=-1 GOTO 1050
1020 LET J=R
1025 ' Repeat
1030 PRINT V(J)
1040 LET J=P(J)
1050 ' Until J=-1
1060 IF J<>-1 GOTO 1025
1070 RETURN
```

Cosa abbiamo imparato

In questa lezione abbiamo visto:

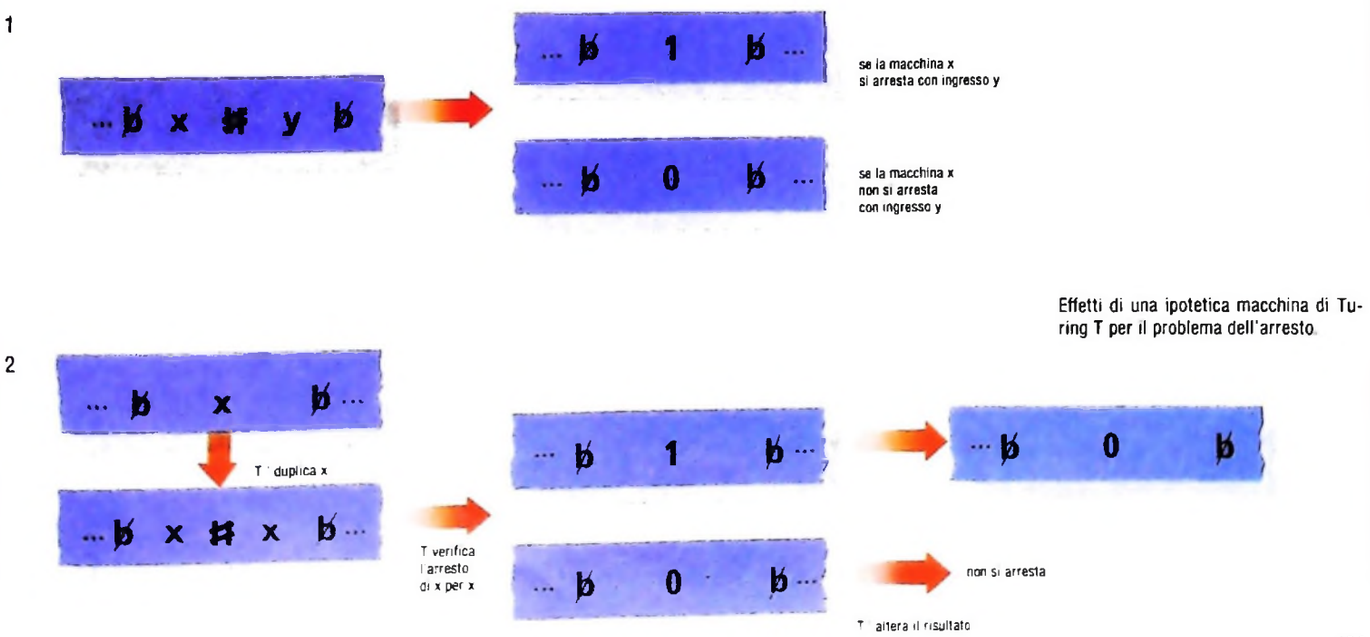
- come si realizza in modo TOP DOWN un programma di simulazione di una lista.

PROBLEMI INSOLUBILI E TESI DI CHURCH

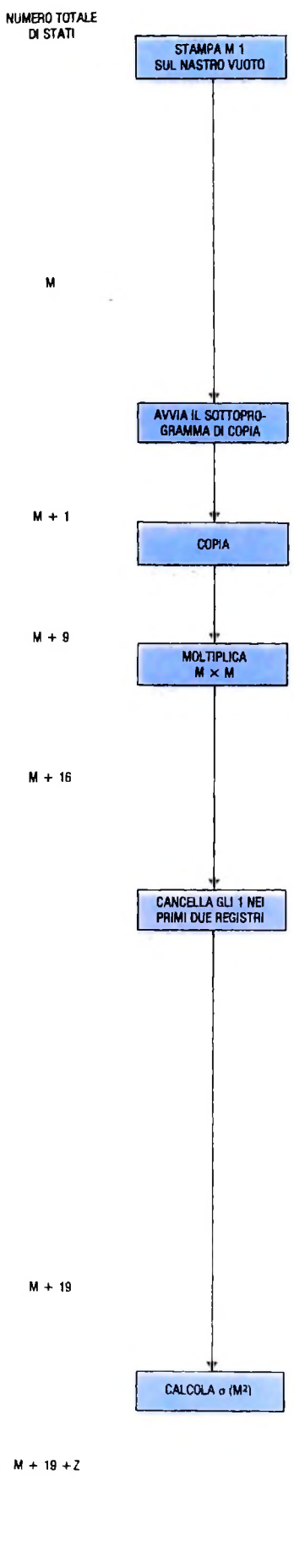
Una precisa delimitazione del dominio del computabile.

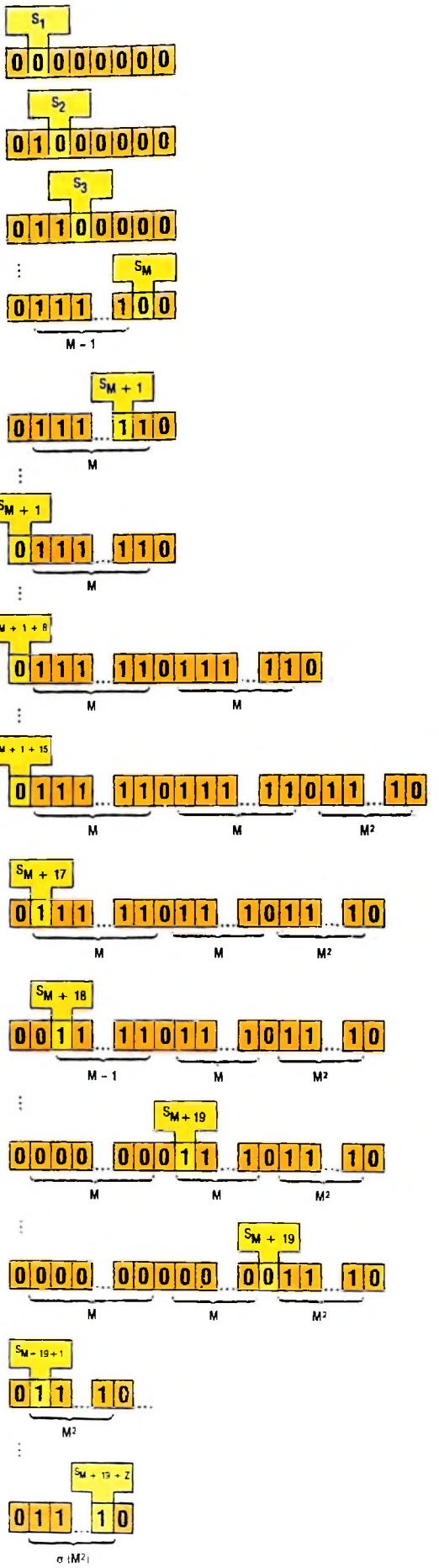
Problemi solubili e problemi insolubili

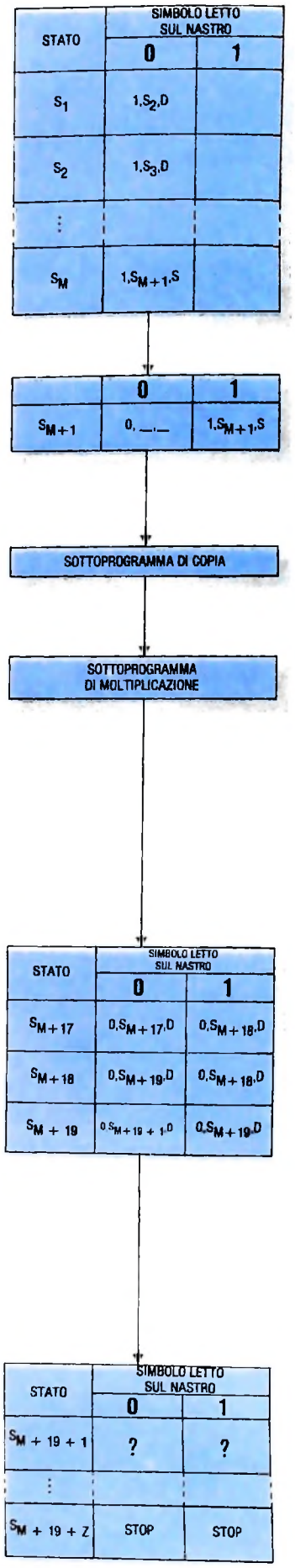
Ora che disponiamo di una nozione di algoritmo e di funzione computabile estremamente precisa, in termini di macchine di Turing, possiamo riprendere la questione centrale della computabilità: esistono funzioni non computabili (o problemi insolubili)?
La risposta è affermativa, e possiamo vederlo in due modi. Cominciamo dal primo. La dimostrazione si basa sul confronto tra il numero totale di funzioni da N in se stesso e il numero di macchine di Turing; poiché queste ultime risultano essere meno numerose, si può concludere che non ci sono abbastanza macchine di Turing per computare tutte le possibili funzioni, anche se non sappiamo quali siano le funzioni non computabili.
Si è detto in precedenza che è possibile, utilizzando un metodo proposto da Gödel, "numerare" le macchine di Turing, cioè rappresentare ognuna di esse con un numero intero positivo che la identifica biunivocamente, cioè le macchine di Turing sono tante quanti sono i numeri interi positivi.
Se ora ci chiediamo quante sono le funzioni da N in se stesso, scopriamo che sono molto più numerose dei numeri interi, e quindi delle possibili macchine di Turing (che pure sono in numero infinito; in matematica, succede anche questo: non c'è un solo numero infinito, ma ci sono diversi "gradi" di infinito): sono tante quanti sono i numeri reali (tecnicamente, hanno la potenza del continuo).
La dimostrazione di questo fatto può essere data per assurdo. Supponiamo di poter numerare le funzioni da N in N. Potremo costruire una tabella infinita in cui la riga i-esima rappresenta i valori che la funzione che sta al posto i nella nostra numerazione associa ai diversi numeri interi. Costruiamo adesso una particolare funzione g secondo questa regola: per ogni numero n, prendiamo la funzione f_n che sta al posto n nel nostro elenco, cerchiamo nella tabella il valore che f associa al numero n, cioè $f_n(n)$, e facciamo in modo che g(n) sia diverso da questo valore. Riusciamo a stabilire in che posizione si trova, nel nostro ipotetico elenco (che dovrebbe contenere tutte le funzioni), la funzione g?
Vediamo: g non può trovarsi al primo posto perché $g(1) \neq f_1(1)$; non può trovarsi al secondo, perché $g(2) \neq f_2(2)$;...; non può trovarsi nemmeno all'n-esimo, perché $g(n) \neq f_n(n)$;... Evidentemente g non può far parte del nostro elenco, quindi il nostro elenco non contiene tutte le fun-



Effetti di una ipotetica macchina di Turing T per il problema dell'arresto.


NUMERO TOTALE DI STATI
STAMPA M 1 SUL NASTRO VUOTO
M
AVVIA IL SOTTOPRO-GRAMMA DI COPIA
M + 1
COPIA
M + 9
MOLTIPLICA M × M
M + 16
CANCELLA GLI 1 NEI PRIMI DUE REGISTRI
M + 19
CALCOLA σ (M²)
M + 19 + Z


S1
0 0 0 0 0 0 0
S2
0 1 0 0 0 0 0
S3
0 1 1 0 0 0 0
SM
0 1 1 1 1 0 0
M - 1
SM + 1
0 1 1 1 1 1 0
M
SM + 1
0 1 1 1 1 1 0
M
SM + 1 + 8
0 1 1 1 1 1 0 1 1 1 1 1 0
M M
SM + 1 + 15
0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0
M M M²
SM + 17
0 1 1 1 1 1 0 1 1 1 1 0
M M M²
SM + 18
0 0 1 1 1 1 0 1 1 1 1 0
M - 1 M M²
SM + 19
0 0 0 0 0 0 0 1 1 1 1 0
M M M²
SM + 19
0 0 0 0 0 0 0 0 0 1 1 1 0
M M M²
SM - 19 + 1
0 1 1 1 0
M²
SM + 19 + Z
0 1 1 1 0
σ (M²)


STATO
SIMBOLO LETTO SUL NASTRO
0
1
S1
1,S2,D
S2
1,S3,D
SM
1,SM+1,S
SM+1
0, —,—
1,SM+1,S
SOTTOPROGRAMMA DI COPIA
SOTTOPROGRAMMA DI MOLTIPLICAZIONE
SM+17
0,SM+17,D
0,SM+18,D
SM+18
0,SM+19,D
0,SM+18,D
SM + 19
0,SM+19+1,D
0,SM+19,D
SM + 19 + 1
?
?
SM + 19 + Z
STOP
STOP

Il problema consiste nel trovare il massimo numero di uni che può essere stampato da una macchina di Turing ad N stati che parte con il nastro pieno di zeri e prima o poi si arresta. Il numero, che dipende da N, è il valore di una funzione indicata con σ (N). Nel 1962 Tibor Rado dell'Università dell'Ohio ha dimostrato che la funzione cresce troppo rapidamente per essere computabile. La tabella a destra mostra i valori minimi stimati per la funzione per piccoli valori di N. Il limite minimo di σ (10) può essere espresso attraverso un numero a che vale circa 1/8: σ (10) è una pila di a, uno all'esponente dell'altro, in cui il numero di a nella pila è espresso da un'altra pila di a. Il processo di definizione dell'altezza di una pila attraverso un'altra pila viene ripetuto 10 volte. La dimostrazione della non computabilità di σ (N) inizia (figura a sinistra) con una stima del numero di stati occorrenti per generare una stringa di M^2 uni su un nastro inizialmente pieno di zeri. Si combina una macchina di Turing che copia il proprio ingresso con una che moltiplica, ottenendo una macchina con M+16 stati. Tre stati ulteriori consentono di cancellare le due sequenze di uni più a sinistra, lasciando una stringa di M^2 uni. Assumiamo poi che esista una macchina con Z stati che, data una stringa di M^2 uni, genera una stringa di σ (M^2) uni. Ma da questa ipotesi deriva una contraddizione, come mostrato nella figura in basso di questa pagina. Ne consegue che σ (N) non è computabile. (L'esempio, dovuto a John E. Hopcroft, è ripreso da «Le Scienze», luglio 1984.)

4

NUMERO DI STATI	MASSIMO NUMERO DI 1 STAMPATI	LIMITE INFERIORE PER IL VALORE DI σ
3	σ (3)	6
4	σ (4)	12
5	σ (5)	17
6	σ (6)	35
7	σ (7)	22.961
8	σ (8)	$3^{92} \cong 7.9 \times 10^{43}$
9	σ (9)	$3^{82}+1$
10	σ (10)	$a^{a^{\cdot^{\cdot^{a}}}}\}a^{a^{\cdot^{\cdot^{a}}}}\} \cdots \}a^{a}$

zioni da N in N, pertanto queste funzioni sono più numerose dei numeri interi.

Il problema dell'arresto per le macchine di Turing

Il secondo tipo di ragionamento, pur essendo perfettamente valido e rigoroso, non è molto soddisfacente; oltre a sapere che esistono problemi insolubili, vorremmo anche vederne uno in concreto, possibilmente semplice da formulare e che si presenti in modo "naturale" in qualche campo della matematica, come, ad esempio, il problema dell'arresto per le macchine di Turing.

Se facciamo partire una macchina di Turing M con un ingresso x, può darsi che essa si fermi dopo un certo numero di passi, ma che anche non si arresti mai. Naturalmente, non possiamo stabilire che non si arresterà mai in modo sperimentale.

Possiamo allora enunciare il problema dell'arresto: esiste una macchina di Turing T che, ricevendo in ingresso la codifica di un'altra macchina di Turing M e di una sua configurazione w, è in grado di decidere se la macchina M, partendo nella configurazione w, prima o poi si arresta oppure no?

La risposta è negativa: il problema dell'arresto è insolubile (secondo Turing). La dimostrazione può essere data per assurdo.

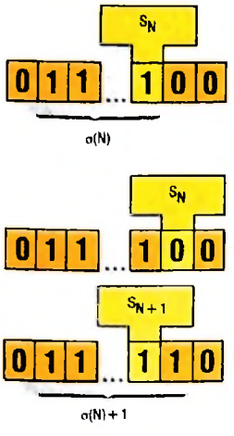
Supponiamo che il problema dell'arresto possa essere risolto algoritmicamente; ciò significa supporre l'esistenza di una macchina di Turing T che, se parte con una descrizione della macchina M (il numero di catalogo x) e con una codifica della configurazione w (un secondo numero y) sul proprio nastro, esegue una computazione che arriva ad una delle seguenti conclusioni:

— se M partendo da w si arresta, allora T cancella ogni simbolo dal proprio nastro, scrive 1 e si arresta;

— se M partendo da w non si arresta, allora T cancella ogni simbolo dal proprio nastro, scrive 0 e si arresta.

La figura 1 rappresenta il nastro di T all'inizio e alla fine del-

5



STATO	SIMBOLO LETTO SU NASTRO	
	0	1
S_N	$1,S_{N+1},S$	$1,S_N,D$
S_{N+1}	STOP	STOP

Si noti innanzitutto che σ è monotona crescente: cioè, se $X>Y, \sigma(X)>\sigma(Y)$. Qualunque sia il numero di 1 stampati da una macchina di Turing a N stati che si ferma, è sempre possibile costruire una macchina di Turing a $(N+1)$ stati che aggiunga un 1 alla parola di 1 e quindi si fermi.

Supponiamo che esista una macchina di Turing a Z stati che calcoli $\sigma(M^2)$.

Per la definizione di σ e per il calcolo mostrato nell'illustrazione della pagina a fronte, $\sigma(M+19+Z) \geq \sigma(M^2)$. (*)

Se M è abbastanza grande, però, l'enunciato (*) contraddice il fatto che σ è monotona crescente. Per dimostrarlo, poniamo $M = Z + 20$, ovvero $Z = M - 20$.

Allora $M + 19 + Z = M + 19 + M - 20 = 2M - 1$.

Per le leggi dell'algebra elementare $2M - 1 = M^2 - (M-1)^2$ e pertanto $M + 19 + Z = M^2 - (M-1)^2$.

Poiché $(M-1)^2 > 1$, $M^2 - 1 > M^2 - (M-1)^2$.

Pertanto, poiché σ è monotona crescente,

$\sigma(M^2-1) > \sigma(M^2 - [M-1]^2) = \sigma(M+19+Z) \geq \sigma(M^2)$.

Dal momento che $\sigma(M^2-1) > \sigma(M^2)$, però, σ non può essere monotona crescente e pertanto l'ipotesi che esista una macchina di Turing a Z stati in grado di calcolare $\sigma(M^2)$ porta a una contraddizione.

la computazione, nelle due ipotesi.
Possiamo ora utilizzare T per costruire una nuova macchina T′ le cui computazioni si possono dividere in tre fasi:
— T′ riceve in ingresso un numero x e nella prima fase della computazione lo duplica;
— dopo aver duplicato x, T′ inizia a comportarsi come T: quindi verifica se la macchina di numero di catalogo x partendo nella configurazione codificata dal numero x si arresta;
— infine, se il risultato della computazione di T è 0, T′ lo cambia in 1 e si arresta, se invece è 1 inizia a muoversi avanti e indietro sul nastro senza mai arrestarsi (figura 2).
Fino a questo punto, sembra che l'intera costruzione, per quanto strana, possa essere effettivamente realizzata. In realtà, un'analisi più accurata della situazione mostra che non è così. Infatti, sia la macchina T che la macchina T′ possono ricevere in ingresso il numero di catalogo di una qualsiasi macchina di Turing, e quindi anche di T′. Ma osserviamo cosa succede se a T′ diamo come ingresso il proprio numero di catalogo, z:
— T′ duplica z;
— T′ richiama T, che stabilisce se T′ con ingresso z si arresta oppure no;
— supponiamo che la risposta sia positiva, cioè che T stampi 1; ma allora T′ non si arresta, e pertanto la risposta data da T è scorretta;
— supponiamo che la risposta sia negativa, cioè che T stampi 0; in questo caso, T′ stampa 1 e si arresta, e di nuovo la risposta di T è scorretta.
Come si vede, l'assunzione che esista una macchina di Turing che risolve il problema dell'arresto porta a una contraddizione, e quindi una tale macchina non può esistere.
Nelle figure 3, 4 e 5 è presentato un altro problema insoluto.
La sostanziale analogia tra macchina di Turing e calcolatori reali rende questo risultato estremamente importante, anche se in negativo, ai fini pratici: esso esclude la possibilità di scrivere un programma P che sia in grado, per ogni altro programma P′, di stabilire se P′ con un certo dato iniziale si arresta oppure entra in loop.
Un secondo problema algoritmicamente insolubile è il problema dell'equivalenza tra programmi:

> Dati due programmi P e P′, essi calcolano la stessa funzione?

Insolubili sono anche problemi del tipo:

> Dato un arbitrario programma P, la settima istruzione verrà eseguita per un qualche dato iniziale o non verrà mai eseguita?

È evidente l'importanza che avrebbe per la programmazione la possibilità di rispondere in modo effettivo a domande del tipo di quelle enunciate sopra. D'altra parte, è anche importante rilevare che se questi problemi sono insolubili non significa che non si può dare una risposta in nessun caso, ma semplicemente che non esiste un metodo universale che consente di dare una risposta in tutti i casi con lo stesso procedimento. Resta comunque aperta la possibilità di trovare procedimenti particolari che si applicano a un solo caso o a un insieme ristretto di casi.

La tesi di Church

Il modello di Turing è indubbiamente un modello formalmente rigoroso di computazione; ed inoltre proprio per la sua semplicità è chiaro che una funzione calcolabile secondo Turing è anche calcolabile in senso intuitivo.
Tuttavia, il nostro problema di partenza riguardava anche il viceversa, cioè fornire una definizione rigorosa di funzione computabile che includesse tutte le funzioni computabili in senso intuitivo.
Ora, nessuno ci può assicurare che non si possa inventare qualche metodo di computazione più potente della macchina di Turing per calcolare un insieme più ampio di funzioni.
In effetti, la macchina di Turing non è l'unica definizione formale di funzione computabile proposta, ma ne esistono numerose altre. Tra queste, possiamo ricordare quelle dovute a S. Kleene, basata sulle equazioni funzionali, ad A. Church, basata sul lambda calcolo, a E. Post, basata sui sistemi canonici, ad A. Markov, basata sugli algoritmi normali, e infine la definizione delle macchine a registri, molto più simili ai calcolatori reali di quanto non lo siano le macchine di Turing, proposta da Shepherdson e Sturgis.
Una particolare importanza per l'informatica ha avuto, e ha tuttora, il lambda calcolo di Church.
Il lambda calcolo è un linguaggio formale per descrivere funzioni da N in N sviluppato da Church, Kleene e Rosser; in seguito, Church ha identificato le funzioni computabili con le funzioni esprimibili nel lambda calcolo, dimostrando che se esistesse una funzione esprimibile nel lambda calcolo, ma non computabile, allora non sarebbe possibile stabilire se una data formula logica è dimostrabile oppure no. Il fatto più interessante dal punto di vista dell'informatica è che il lambda calcolo è stato tradotto in un linguaggio di programmazione reale, e notevolmente diffuso in aree di ricerca come l'intelligenza artificiale, il LISP.
A ognuna delle definizioni proposte corrisponde ovviamente una classe di funzioni computabili. Il confronto tra queste classi, definite in modo diverso, porta a una conclusione di fondamentale importanza:

> Tutte le definizioni formali di funzione effettivamente calcolabile proposte caratterizzano la stessa classe di funzioni.

Il significato di questo risultato è chiaro: la classe delle funzioni computabili è molto stabile rispetto al modello di computazione, per cui possiamo assumere un unico formalismo, ad esempio quello di Turing, come riferimento. Questo è il contenuto della cosiddetta "tesi di Church":

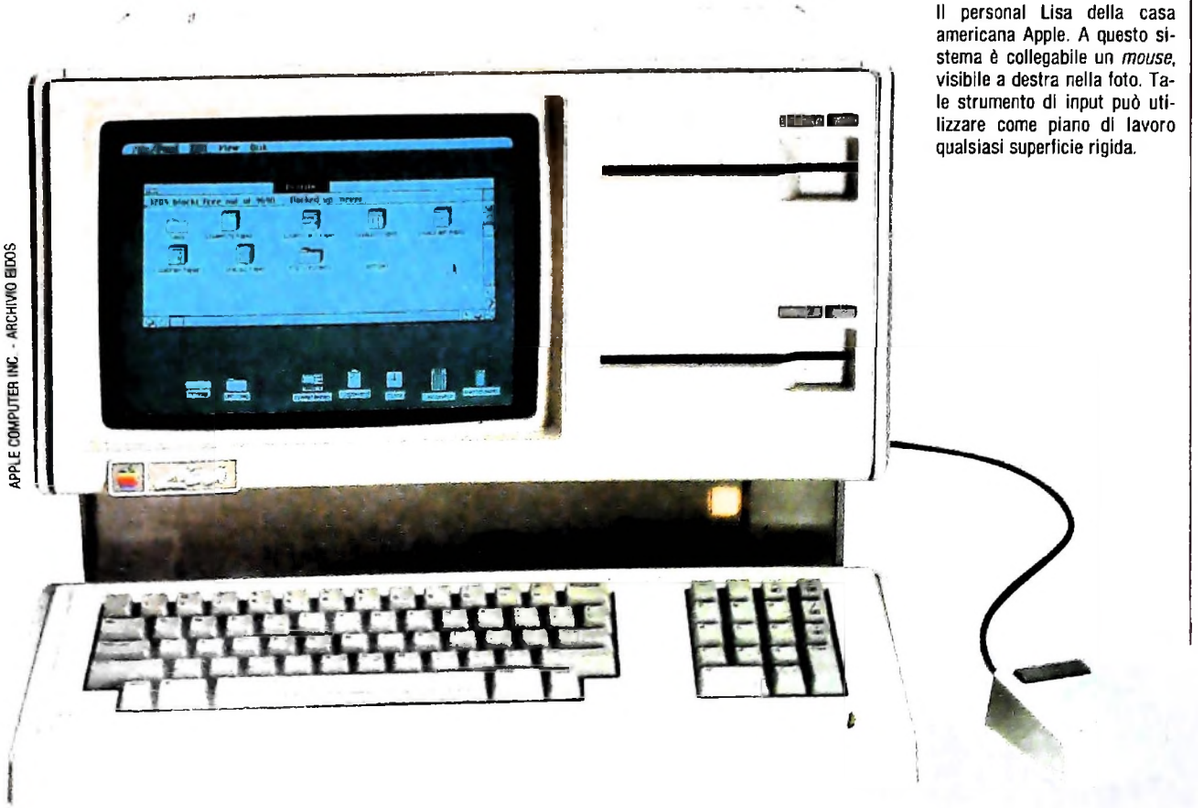
> La classe delle funzioni parziali effettivamente calcolabili in senso intuitivo coincide con la classe delle funzioni computabili secondo Turing.

L'affermazione precedente non è evidentemente un teorema dimostrabile in modo rigoroso, né d'altra parte potrebbe esserlo, ma si basa su una serie di fatti empirici come l'equivalenza tra tutti i formalismi fino ad oggi proposti, oppure il fatto che tutti gli algoritmi che sono stati sviluppati in millenni di storia della matematica, sono descrivibili attraverso macchine di Turing.

HARDWARE GRAFICO: LE UNITÀ DI INTERAZIONE

I display grafici che permettono di raggiungere il maggiore equilibrio fra qualità, contrasto e stabilità dell'immagine.

L'unità di interazione per eccellenza è il terminale grafico che ha la funzione di visualizzare le informazioni elaborate dal computer. L'apparecchiatura principale di un terminale è lo schermo grafico (spesso chiamato "display"), costituito da un tubo a raggi catodici (CRT, dall'inglese *Cathode Ray Tube*) simile a quello del televisore, che può essere di varie dimensioni, in bianco e nero o a colori.
Un CRT è uno speciale tubo in grado di emettere un fascio di elettroni e di proiettarli sullo schermo del display, che è ricoperto di fosfori, i quali, quando sono colpiti dal fascio elettronico, si eccitano e danno origine a un processo di emissione di radiazioni luminose, detto "fosforescenza". La misura della capacità del fosforo di restare nello stato di fosforescenza viene detta "persistenza": essa è diversa a seconda dei tipi di fosfori utilizzati e dall'intensità del fascio elettronico che li colpisce; può variare da qualche microsecondo a qualche secondo. Nei casi in cui il decadimento dei fosfori dallo stato eccitato è rapido, l'immagine riprodotta sul display dalla loro fluorescenza è visibile solo momentaneamente. Per questo motivo si rende necessario un rinfrescamento periodico (*refresh*, in inglese) dell'intera immagine, ossia un'eccitazione periodica, mediante il fascio elettronico dei fosfori costituenti l'immagine. Se la persistenza dei fosfori è breve e il periodo di rinfrescamento non è sufficientemente elevato (almeno 25 volte al secondo), l'occhio umano è in grado di percepire la poca stabilità dell'immagine: fenomeno chiamato "sfarfallio" (*flicker*, in inglese).
Il grado di eccitazione del fosforo determina il grado di visibilità dell'immagine sullo schermo, mentre il grado di persistenza determina la frequenza con cui è necessario rinfrescare il fosforo per mantenere visibile l'immagine sullo schermo. Il fascio di elettroni è indirizzato verso punti elementari della superficie del display: una maggiore quantità di punti, a parità di dimensioni, vuol dire maggiore risoluzione e quindi maggiore qualità dell'immagine, ma anche più tempo necessario al fascio elettronico per spazzarli tutti.



APPLE COMPUTER INC. - ARCHIVIO BIDOS

Il personal Lisa della casa americana Apple. A questo sistema è collegabile un *mouse*, visibile a destra nella foto. Tale strumento di input può utilizzare come piano di lavoro qualsiasi superficie rigida.

La differenza maggiore fra lo schermo di un terminale grafico e quello di un comune televisore sta nel fatto che mentre nel primo si riproducono generalmente immagini statiche, il secondo è tipicamente utilizzato per immagini animate, ossia in movimento. Questo fatto impone che la qualità dello schermo del terminale debba essere molto superiore a quello di un normale apparecchio televisivo, proprio perché sono necessari un maggiore contrasto e una maggiore stabilità. Maggiore qualità e contrasto perché in una rappresentazione grafica i contorni e i particolari sono più importanti che in una normale rappresentazione televisiva, dove l'occhio è concentrato solo sul soggetto in primo piano. La maggiore stabilità dell'immagine è invece necessaria perché il fenomeno dello sfarfallio, trascurabile sul televisore domestico, diventa fondamentale sul terminale grafico per non affaticare l'occhio dell'operatore.
Il problema di trovare un compromesso ottimale fra intensità del fascio elettronico, tipo di fosforo e velocità di scansione per generare un'immagine qualitativa, ben contrastata e visibile sullo schermo, non è di semplice soluzione. Se poi si aggiunge la complicazione legata al tempo di persistenza dell'immagine sulla retina dell'occhio, si comprende anche come l'ottenere la massima stabilità sia un problema non indifferente. I tipi di display attualmente disponibili sul mercato cercano ognuno di risolvere questi complessi problemi per vie diverse.
Possiamo classificare le tecnologie sotto due grandi categorie, caratterizzate dal modo in cui viene visualizzata l'immagine:

— Display a vettori.
— Display raster o display a punti.

Per display a vettori si intende uno schermo dotato di una tecnologia che richiede di conoscere solo gli estremi di un segmento per visualizzarlo sullo schermo. Due sono i tipi di display a vettori attualmente disponibili sul mercato: "*storage*" e "*refresh*".
Per tecnologia "*raster*" si intende invece un metodo che procede alla scansione sequenziale di tutti i pixel dello schermo e, mediante informazioni memorizzate, accende i pixel che costituiscono il disegno. Nel display raster il disegno si mostra sempre partendo dall'alto verso il basso.

Display storage

Il *display storage* o a memorizzazione d'immagini è stata la prima tecnologia usata per rappresentazioni grafiche e il suo brevetto è di proprietà di una nota società americana produttrice di videoterminali, la Tektronix.
L'idea base è quella di memorizzare l'immagine sullo schermo, senza dover rinfrescare ogni volta i fosfori. Il fascio di elettroni del tubo catodico viene usato come una penna di colore verde chiaro che scrive su una lavagna a fondo verde più scuro. La superficie dello schermo, sulla quale sono stati depositati dei fosfori ad alta persistenza, viene usata, oltre che per visualizzare l'immagine, anche per memorizzarla: in questo modo non si richiede nessuna memoria d'immagazzinamento dati. Questa tecnologia permette di avere uno schermo ad alta risoluzione (1024 × 1024 punti indirizzabili), ma non è adatta per le situazioni di continua interazione, poiché per ogni modifica anche minima del disegno è necessario ridisegnare tutta l'immagine.
Riassumiamo in sintesi i principali vantaggi e svantaggi di questa tecnologia.

Vantaggi:
— La risoluzione è limitata solo da questioni ottiche (risoluzione dell'occhio umano) e fisiche (dimensioni dello schermo): la qualità quindi è ottima.
— Non esiste la necessità di rinfrescare l'immagine, e pertanto non c'è sfarfallio e la stabilità è perfetta.
— La velocità di tracciamento delle linee è elevatissima.


MEGATEK CORP - ARCHIVIO EIDOS

Terminali grafici dotati di *joystick* incorporato. Il joystick è uno degli strumenti di interazione con il display, diffuso ormai anche nel mondo del personal computer, dove la grafica sta suscitando sempre più interesse e curiosità.

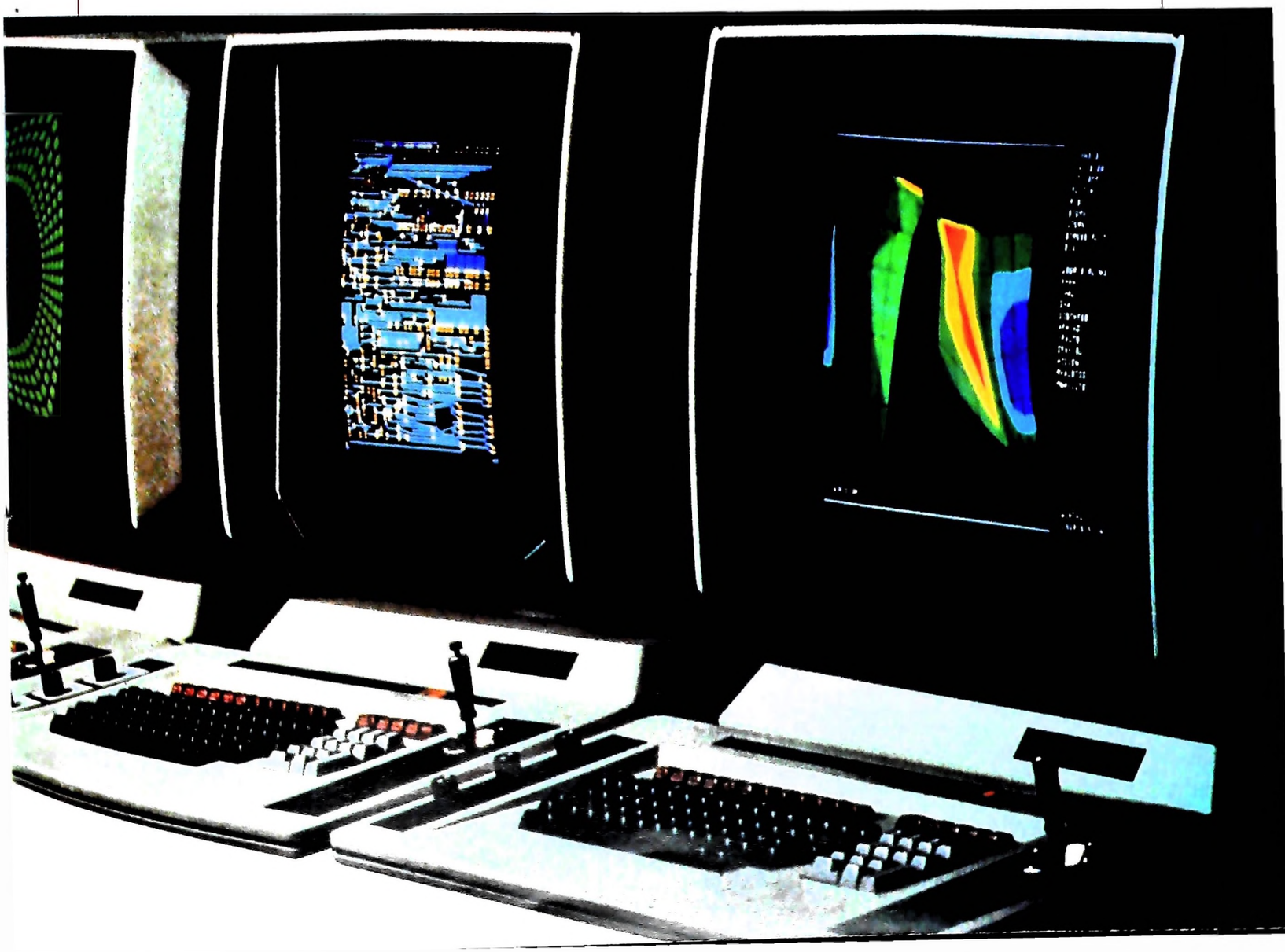
Svantaggi:
— Dato che l'immagine è memorizzata nel fosforo dello schermo, per apportare modifiche è necessario cancellare e ridisegnare tutto. Con l'elevatissima velocità di tracciamento oggi possibile questo inconveniente non costituisce più un grosso limite.
— Per assicurare buona durata ai fosfori, che si esauriscono col tempo, il livello di intensità d'emissione degli elettroni è mantenuto basso, di conseguenza la luminosità dell'immagine è ridotta e il contrasto limitato.
— È possibile, allo stato attuale della tecnologia, gestire il colore solo in maniera molto limitata.

Display refresh

Con questa tecnologia, detta anche "a rinfrescamento di vettori", invece di memorizzare le informazioni dell'immagine direttamente sullo schermo, si usa una memoria per generare l'immagine: ciò permette di modificare parti dell'immagine stessa. Ossia, l'idea base è quella di costruire l'immagine sullo schermo in maniera vettoriale e di mantenerla visibile con un processo di rinfrescamento continuo. I fosfori in questo caso sono a bassa persistenza. Se da un lato con questa tecnologia è possibile una risoluzione molto elevata, non appena l'oggetto diviene molto complesso, la velocità di rinfrescamento diminuisce e l'utente percepisce lo sfarfallio dell'immagine (*flickering*).

Vantaggi:
— Poiché l'immagine è ridisegnata continuamente, può anche essere modificata istantaneamente; quindi l'interattività è massima.
— La velocità di tracciamento è elevatissima, come per lo storage.
— Il contrasto è molto buono, poiché il fascio elettronico può essere mantenuto ad un'intensità più alta.

Svantaggi:
— Se l'immagine è complessa, con l'aumentare dei tempi di rinfrescamento, può comparire il fenomeno del flicker.
— La componentistica elettronica è molto sofisticata, pertanto i costi sono relativamente alti, soprattutto se si vuole consentire l'uso del colore.

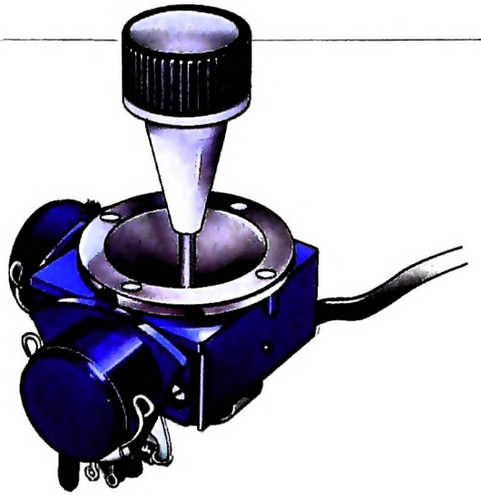


Display raster

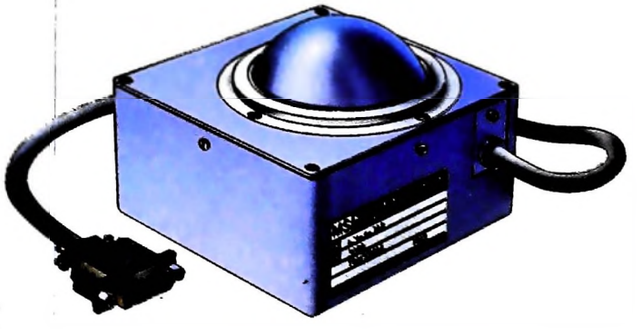
Questa tecnologia è simile a quella dei normali televisori. L'immagine sullo schermo è costruita per punti (pixel); il fascio di elettroni effettua la scansione dello schermo per righe orizzontali dall'alto verso il basso, attivandosi o disattivandosi a seconda che lo stato memorizzato dal punto sia acceso o spento.
I fosfori del display sono a bassa persistenza e l'intensità dell'immagine può essere facilmente variata, punto per punto, permettendo così una vasta gamma di livelli d'intensità. Questa tecnologia consente inoltre una gestione dei colori, come vedremo, molto più efficace delle altre.

Vantaggi:
— Grande economicità, poiché sfrutta un tipo di tecnologia, quella televisiva, già prodotta su grande scala.
— Grande possibilità di interazione, come per il *display refresh*.
— Ottimo contrasto sia in bianco e nero sia a colori.
— Qualità, stabilità e velocità indipendenti dalla complessità dell'immagine sullo schermo.

Svantaggi:
— La risoluzione è qui limitata da diversi fattori: necessità di dover rinfrescare l'immagine; esigenza di dover convertire i comandi dalla forma vettoriale, così come sono generati dal computer, alla matrice di punti, come richiesto dal display; tempi di trasferimento dei dati dalla memoria di rinfrescamento contenente la matrice dei pixel, detta anche bit-map, allo schermo.
L'evoluzione della tecnologia elettronica e dei display televisivi, la necessità di contenere i costi e l'esigenza di poter impiegare sempre più la tecnica del colore, fanno ritenere che in futuro la tecnica del display raster sarà sempre più dominante nel campo della grafica interattiva.



Meccanismi di *joystick* (sopra) e di *joydisk* (sotto). Tali strumenti sono generalmente incorporati nella tastiera, per migliorare la qualità ergonomica del lavoro, cioè ridurre il dispendio di energia.



Joystick e joydisk possono anche essere utilizzati come unità separate. Nel mercato dei personal, per esempio, vengono sovente offerti come accessori collegabili al sistema.

Gli strumenti di interazione sul display

Per completare il discorso sulle unità di interazione, dobbiamo ora descrivere quelle componenti che consentono il colloquio e l'interattività col computer.
Oltre alla tradizionale tastiera, i terminali grafici sono dotati almeno di un'apparecchiatura che consente una tale interattività. Lo scopo e le modalità di funzionamento di questi strumenti sono simili a quelli esposti per il *digitizer* nella lezione precedente. Ossia, essi devono servire all'operatore per indicare sullo schermo una entità grafica ivi riprodotta alla quale applicare una certa funzione, oppure trasmettere al computer le coordinate di uno o più punti dello schermo sui quali effettuare delle elaborazioni grafiche.
Tutti i terminali grafici sono equipaggiati con un reticolo a croce, visualizzabile sul display sempre in modo dinamico, che è normalmente chiamato cursore. I due segmenti di retta ortogonali che costituiscono il cursore possono essere mossi sullo schermo e posizionati su qualsiasi suo punto. Generalmente un punto è sufficiente per individuare qualsiasi entità grafica di cui è costituito il disegno, purché sia stata definita come tale nel computer. Il cursore può essere attivato o disattivato a piacere; non appena attivato e definita la funzione grafica da eseguire, tramite tastiera o menù, la coordinata della posizione del cursore viene trasferita al computer dove il programma grafico l'associerà all'entità grafica di appartenenza già esistente. Le metodologie per lo spostamento sul display del cursore determinano i vari tipi di strumenti in questione collegabili al terminale grafico:
— *tablet*, un digitizer di piccole dimensioni;
— *joystick*, costituito da una levetta che ruota di 360 gradi e che si inclina di 45;
— *joydisk*, costituito da una semisfera che opera con lo stesso principio;
— *mouse*, una scatoletta con due rotelle disposte ortogonalmente.

Anche in leasing con Olivetti Leasing

LA FAMIGLIA DEI PERSONAL COMPUTER OLIVETTI

FRIENDLY & COMPATIBLE

Questa famiglia di personal compatibili tra loro e con i più diffusi standard internazionali, non ha rivali per espandibilità e flessibilità. Prestazioni che su altri diventano opzionali, sui personal computer Olivetti sono di serie. Per esempio M24 offre uno schermo ad alta definizione grafica, ricco di 16 toni o di 16 colori e con una risoluzione di 600x400 pixel; mentre la sua unità base dispone di 7 slots di espansione, fatto questo che gli consente di accettare schede di espansione standard anche se utilizza un microprocessore a 16 bit reali (INTEL 8086). Ma ricchi vantaggi offrono anche tutti gli altri modelli.

Basti pensare che tutte le unità base includono sia l'interfaccia seriale che quella parallela. Oppure basti pensare all'ampia gamma di supporti magnetici: floppy da 360 a 720 KB o un'unità hard disk (incorporata o esterna) da 10 MB.
La loro compatibilità, inoltre, fa sì che si possa far uso di una grande varietà di software disponibile sul mercato. Come, ad esempio, la libreria PCOS utilizzabile anche su M24. Come le librerie MS-DOS®, CP/M-86® e UCSD-P System®, utilizzabili sia da M20 che da M21 e M24.

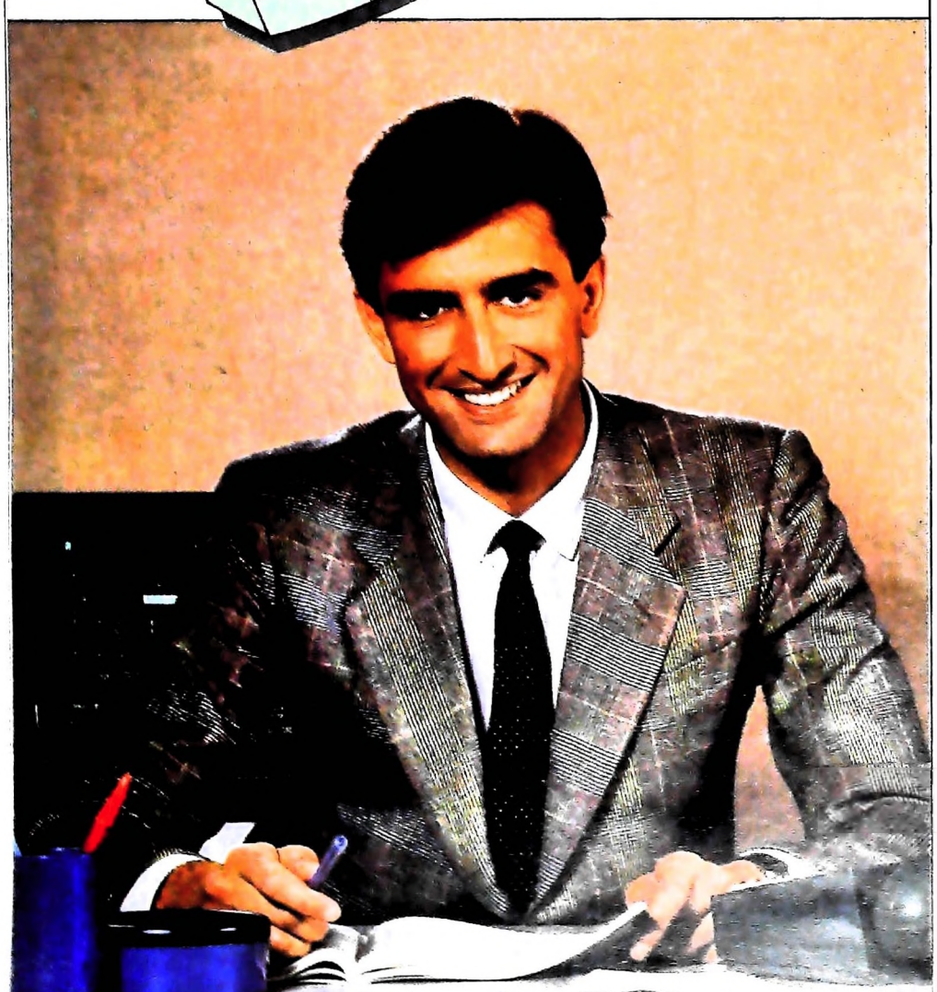
olivetti

Per ulteriori informazioni inviare il coupon a Olivetti
Divisione Personal Computer Via Maravigli 12 20123 Milano

NOME
INDIRIZZO
CITTÀ
TELEFONO

UN NUOVO MODO DI USARE LA BANCA.

Consulenza



GLI INVESTIMENTI CON VOI E PER VOI DEL BANCO DI ROMA.

Il Banco di Roma non si limita a custodire i vostri risparmi. Vi aiuta anche a farli meglio fruttare. Come? Mettendovi a disposizione tecnici e analisti in grado di offrirvi una consulenza di prim'ordine e di consigliarvi le forme di investimento piú giuste. Dai certificati di deposito ai titoli di stato, dalle obbligazioni alle azioni, il Banco di Roma vi propone professionalmente le varie opportunità del mercato finanziario. E grazie ai suoi "borsini", vi permette anche di seguire, su speciali video, l'andamento della Borsa minuto per minuto.

Se desiderate avvalervi di una gestione qualificata per investire sui più importanti mercati mobiliari del mondo, i fondi comuni del Banco di Roma, per titoli italiani ed esteri, vi garantiscono una ampia diversificazione.

Inoltre le nostre consociate Figeroma e Finroma forniscono consulenze per una gestione personalizzata del portafoglio e per ogni altra esigenza di carattere finanziario.

Veniteci a trovare. ci conosceremo meglio.

BANCO DI ROMA

CONOSCIAMOCI MEGLIO.