è ora comandato dalla parola **return** (in italiano, "restituisci"), che restituisce, come risultato, il valore che segue tale parola, associandolo al nome FRICERCA. Avremo cioè il risultato FRICERCA = **true** o FRICERCA = **false** a seconda dei casi. Facendo uso della procedura FRICERCA, l'algoritmo 2.5 può così essere riformulato nella forma 2.7 (si noti anche qui la semplice forma assunta dalla condizione nella frase **if**).

Algoritmo 2.7. Verifica dell'appartenenza di un elemento y a due insiemi B e C, per mezzo della procedura-funzione FRICERCA:

if FRICERCA $(y, B) \cap$ FRICERCA (y, C)
  then stampa "y appartiene a $B \cap C$"
  else stampa "y non appartiene a $B \cap C$";

Notiamo infine che il concetto di procedura offre vantaggi che vanno ben oltre quello immediato di evitare ripetizioni di parti del programma, laddove si debba eseguire la stessa computazione. Infatti, in virtù della distinzione tra parametri formali e reali, le procedure costituiscono entità indipendenti, e possono essere progettate, collaudate e utilizzate da persone diverse. Ciò favorisce lo sviluppo di algoritmi modulari e ne rende assai più semplice la costruzione nel caso di problemi complessi.

F. LUCCIO
"LA STRUTTURA DEGLI ALGORITMI"

Capitolo 3

Divide (et impera?)

Nel primo capitolo abbiamo visto come una adeguata rappresentazione della struttura di un algoritmo (in quel caso si trattava di mettere ordine nei confronti fra monete) permetta di fare importanti considerazioni sul numero minimo di passi da compiere e in ultima analisi guidi a interessanti scelte sul tipo e sull'ordine dei confronti.

A parte il carattere volutamente introduttivo dell'esempio, le cui implicazioni sono state lasciate su un piano meramente intuitivo, la struttura di albero di decisione adottata in quel caso non è uno strumento per l'impostazione di algoritmi, ma, piuttosto, un mezzo per stabilire un limite inferiore al numero di decisioni successive, e una guida per eseguire scelte bilanciate.

Strumenti ben più generali sono necessari per stabilire in che direzione muoversi dinanzi a un problema arbitrario, e dobbiamo ammettere che non si conoscono ancora criteri universali. Vi sono tuttavia situazioni caratteristiche nelle quali l'algoritmo si imposta su linee standard. Studiamo ora la situazione più importante, che si verifica quando un problema si ripropone al suo interno in sottoproblemi uguali all'originale, ma applicati a sottoinsiemi dei dati, e la soluzione globale si ottiene come combinazione delle soluzioni dei sottoproblemi.

3.1. La ricorsività

Per chiarire i nostri intendimenti, immaginiamo di dover calcolare il fattoriale di un numero dato N. Possiamo far uso del seguente algoritmo 3.1, che è organizzato come procedura-funzione e assegna al nome FATTORIALE il valore di $N!$.

Algoritmo 3.1. Procedura-funzione per il calcolo del fattoriale di un intero $N \geqslant 1$:

```
procedure FATTORIALE(N):
    begin k←1;
        i←1,
        while i≤N do
            begin k←k×i;
                i←i+1
            end;
        return k
    end;
```

L'algoritmo 3.1 assegna successivamente a k i valori 1, 1×1, 1×2, $1\times 2\times 3$, ..., $1\times 2\times 3\times \ldots \times N$, e quest'ultimo valore è a sua volta assegnato a FATTORIALE dalla frase **return** k. Partendo dall'ultimo stadio del calcolo, l'algoritmo può essere descritto come segue:

Per calcolare il fattoriale di N, si calcola il fattoriale di N-1 e lo si moltiplica per N.

Una simile affermazione deve essere presa con cautela. Per eseguire un calcolo ($N!$) non si assegna una esplicita serie di azioni, ma si invoca l'esecuzione del calcolo stesso su un dato più piccolo ($(N-1)!$). La difficoltà pare non risolta, ma solo rimandata. E in effetti una descrizione siffatta è sostenibile solo se corredata da una clausola di chiusura, che stabilisca come eseguire il calcolo per valori limite dei dati. Nel nostro caso:

$$N!=1, \quad \text{per} \quad N=1,$$
$$N!=N\times(N-1)!, \quad \text{per} \quad N>1.$$

In questo modo il calcolo di $N!$ viene condizionato a quello di $(N-1)!$, che a sua volta è condizionato al calcolo di $(N-2)!$, e così via, finché il calcolo di 1! è eseguito assegnando direttamente il valore 1. Si ottiene così l'algoritmo 3.2, organizzato anch'esso come procedura-funzione.

Algoritmo 3.2. Calcolo del fattoriale di N mediante una procedura ricorsiva:

```
procedure RFATTORIALE(N):
    begin
        if N=1 then return 1
               else return N× RFATTORIALE(N−1)
    end;
```


Una procedura che chiama sé stessa al suo interno è detta *ricorsiva*. La RFATTORIALE è una procedura ricorsiva, e per poterla introdurre ammetteremo che il nostro linguaggio accetti la ricorsività. Vi sono da fare allora alcune considerazioni preliminari. E' sotto la completa responsabilità di chi formula l'algoritmo porvi in posizione opportuna la clausola di chiusura (**if** $N=1$), onde assicurarsi che le chiamate successive della procedura abbiano termine. E' invece responsabilità del sistema di elaborazione organizzare i calcoli richiesti dalla procedura, che devono essere eseguiti aggregando man mano dati elementari. Se applichiamo per esempio l'algoritmo 3.2 al caso $N=5$ (calcolo di RFATTORIALE(5)), il test "**if** $N=1$" non è soddisfatto, e il calcolo di 5! è ridefinito come $5\times 4!$. La valutazione del prodotto è rimandata in attesa di eseguire RFATTORIALE(4), ove il calcolo di 4! è a sua volta ridefinito come $4\times 3!$. Anche questa moltiplicazione è rimandata, finché di passo in passo si giunge a valutare RFATTORIALE(1): Il test "**if** $N=1$" è ora soddisfatto, e si ottiene direttamente RFATTORIALE(1)=1. A questo punto si esegue l'ultimo calcolo lasciato in sospeso, cioè si rientra nell'ambito di RFATTORIALE(2), e si calcola il valore $2\times 1!=2$. E' ora la volta di RFATTORIALE(3), ove si produce il calcolo $3\times 2!=3\times 2=6$. Si valutano quindi RFATTORIALE(4) come $4\times 3!=4\times 6=24$, e RFATTORIALE(5) come $5\times 4!=5\times 24=120$.

E' opportuno sottolineare che *ogni parametro di una procedura ricorsiva può assumere allo stesso tempo più valori indipendenti*, relativi alle esecuzioni della procedura aperte contemporaneamente. Il valore impiegato in una esecuzione muore con il completamento di questa, ed è sostituito dal valore relativo all'ultima esecuzione lasciata in sospeso. Tali assegnazioni di valori, come il complesso gioco di calcoli lasciati in sospeso ed eseguiti poi al momento opportuno, sono totalmente gestiti dall'elaboratore, mediante meccanismi che verranno chiariti in seguito. Il programmatore formula l'algoritmo nella forma 3.2, che non è altro che l'espressione in termini programmativi della definizione ricorsiva del fattoriale, e deve solo controllare che tale formulazione corrisponda alla funzione da ottenere, e che la clausola di chiusura interrompa correttamente la sequenza di chiamate, l'una interna all'altra, su una condizione limite dei dati.

Notiamo anche come la formulazione dell'algoritmo vada per così dire dal generale al particolare, poiché descrive la funzione sulla globalità dei dati (RFATTORIALE(N)) in termini della funzione stessa su dati disaggregati o comunque più semplici (RFATTORIALE($N-1$)); invece l'esecuzione dell'algoritmo segue la via opposta, poiché lascia in sospeso le operazioni globali e inizia il calcolo vero e proprio da dati atomici (valutazione di RFATTORIALE(1), quindi di RFATTORIALE(2)=2× RFAT-

TORIALE(1) ecc.). Ciò significa che, se il linguaggio di programmazione non accetta la ricorsività, l'autore dell'algoritmo deve ricorrere a una formulazione come la 3.1, ove il calcolo è esplicitamente comandato per successive aggregazioni dei dati. Vedremo come, se il problema la consente, la formulazione ricorsiva sia la più semplice per il programmatore, poiché lascia all'elaboratore gran parte dell'organizzazione dei calcoli.

Il calcolo del fattoriale si presta naturalmente a una formulazione ricorsiva. Gli algoritmi 3.1 e 3.2, seppure di impostazione filosoficamente diversa, conducono alla esecuzione delle stesse moltiplicazioni nello stesso ordine, e sono quindi egualmente efficienti, almeno in prima approssimazione. Egualmente semplice in questo caso è costruire una dimostrazione per accertarsi ragionevolmente che tali algoritmi sono corretti, cioè calcolano $N!$ per ogni intero $N \geqslant 1$. In particolare, la correttezza dell'algoritmo ricorsivo 3.2 si dimostra per induzione, notando per ispezione diretta che esso è corretto per $N=1$, e che, se corretto per $N-1$ (cioè se RFATTORIALE$(N-1)=(N-1)!$), allora è anche corretto per N (poiché genera RFATTORIALE$(N)=N\times(N-1)!$).

Non si deve però credere che l'impiego della ricorsività si limiti a casi così ovvi. Per il semplicissimo problema che segue si aprirà una via di risoluzione insospettata, che potremo poi condurre a grande generalità.

Consideriamo un insieme A di n elementi distinti, indicati con $A(1)$, $A(2), \ldots, A(n)$, tra cui sia definita una relazione di ordinamento totale indicata con $<$ (cioè per ogni coppia $A(i), A(j)$, con $i \neq j$, abbiamo $A(i) < A(j)$ oppure $A(j) < A(i)$, e la relazione è transitiva). Vogliamo anzitutto trovare l'elemento massimo di A, che chiameremo anche *primo*: cioè l'elemento $A(k)$ tale che $A(i) < A(k)$ per ogni $i \neq k$. L'algoritmo 3.3 risolve in modo ovvio il problema tramite una scansione dell'insieme, ad ogni passo della quale si individua il massimo corrente ("max") tramite confronto tra il nuovo elemento in esame e il massimo trovato fino al passo precedente. Si eseguono così $n-1$ confronti; né potrebbero eseguirsene meno, poiché ciascuno degli $n-1$ elementi diversi dal massimo deve uscire perdente in almeno un confronto, senza di che non può concludersi che detto elemento non è il massimo. E poiché da ogni confronto esce esattamente un perdente, dovremo eseguire almeno tanti confronti quanti elementi devono essere provati perdenti.[1]

[1] Per eseguire i confronti immaginiamo di disporre di un meccanismo elementare che stabilisce la relazione tra ogni coppia di elementi, e valutiamo l'efficienza dell'algoritmo in base al numero di confronti eseguiti. La ragionevolezza di questo criterio di misura apparirà chiara in un prossimo capitolo.