14.1 Limite degli array

Le strutture dati individuate per la risoluzione dei problemi sono dette "astratte". Una *struttura dati astratta* definisce l'organizzazione delle informazioni indipendentemente dalla traduzione in uno specifico linguaggio di programmazione; la rappresentazione che una struttura astratta prende in un linguaggio di programmazione viene detta *implementazione*.
Le strutture astratte dipendono dal problema in considerazione; alcune di esse, come le pile, le code e gli alberi, rivestono però un'importanza generale per l'uso comune che se ne fa nella soluzione di intere classi di problemi. Nella fase di analisi il programmatore si può accorgere che proprio una di queste strutture fa al caso suo; successivamente, dovrà scegliere la tecnica migliore per implementarle. Per facilitare il lavoro del programmatore, il C mette a disposizione una serie di strumenti, tra i quali gli array, le `struct` e i puntatori. In questo capitolo e nel seguente faremo pratica di programmazione implementando alcune tra le più importanti strutture dati.
Fino qui, i dati omogenei sono stati memorizzati con l'array. Questi ultimi, però, possono porre dei problemi, in merito a:

- occupazione di memoria;
- velocità di esecuzione;
- chiarezza della soluzione proposta.

Consideriamo in primo luogo l'*occupazione di memoria*. Il numero di elementi di un array viene definito nelle parti dichiarative del programma. È essenziale dunque, in fase di analisi del problema, effettuare una stima sul massimo numero di elementi che eventualmente potranno essere utilizzati. In molti casi non è possibile dare una valutazione esatta, per cui si dovrà sovrastimare l'array, cioè definirlo con un numero di elementi probabilmente molto superiore al necessario, occupando più memoria di quella realmente indispensabile.

Per esempio, si supponga di elaborare le prenotazioni di 700 studenti delle scuole medie relativamente a una delle tre gite scolastiche previste nell'anno: la prima a Vienna, la seconda a Parigi e la terza a Roma. Per formare le tre liste di studenti si possono utilizzare tre array di caratteri: `Vienna`, `Parigi` e `Roma`. Ognuno di essi dovrà essere definito con 700 elementi, perché non si conosce a priori la distribuzione delle scelte degli studenti, per cui si deve considerare il caso peggiore, in cui tutti gli studenti scelgono la stessa gita. In questo caso si avrà una occupazione di memoria pari (o superiore, se qualche studente non partecipa a nessuna gita) a tre volte quella effettivamente richiesta.
Consideriamo ora il secondo fattore: la *velocità di esecuzione*. Il fatto che gli elementi di un array siano rigidamente connessi l'uno all'altro in una successione lineare appesantisce gli algoritmi di risoluzione di alcuni problemi, rendendo non accettabili i tempi di risposta dei programmi corrispondenti.
Si supponga per esempio di dover elaborare una lista ordinata di interi e prevedere le operazioni di inserimento ed eliminazione su questa lista. Se si memorizza la lista in un array a una dimensione l'operazione di inserimento di un valore deve prevedere le fasi:

1. ricerca del punto d'inserimento;
2. spostamento di un posto di tutti gli elementi successivi;
3. inserimento dell'informazione nell'array.

Si osservi al proposito l'esempio di Figura 14.1, dove in una lista costituita dai valori 3, 7, 20, 31, 100, 200 memorizzata nei primi sei elementi di un vettore vogliamo inserire il valore 11 mantenendo l'ordinamento. In questa sede è interessante sottolineare la fase di spostamento in avanti dei valori che seguono l'11, fase in cui sono necessari ben quattro spostamenti prima di poter inserire il nuovo elemento effettuando in pratica quattro riscritture.

Poiché in generale il numero di spostamenti corrisponde al numero degli elementi del vettore successivi a quello da inserire, per ogni inserimento in un vettore che già contiene *n* valori dovremo in media effettuare *n*/2 riscritture. Analoghe considerazioni valgono per una cancellazione.

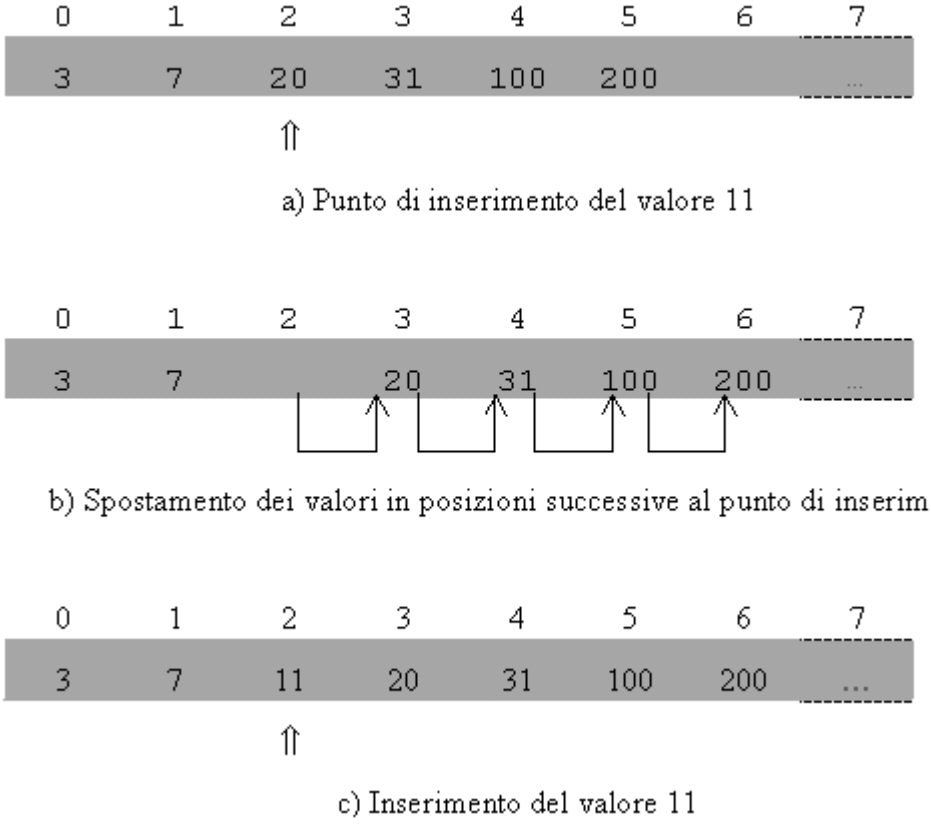

Figura 14.1 Inserimento ordinato di un elemento nella sequenza memorizzata in un array.

Consideriamo infine il terzo fattore: la *chiarezza della soluzione proposta*. Più l'organizzazione della struttura corrisponde alla logica di utilizzazione delle informazioni in essa contenute, più è chiara la soluzione. Questo fattore ha un riflesso determinante sulla bontà della programmazione, in termini di diminuzione dei tempi di revisione e modifica dei programmi. In sintesi, i limiti che vengono ravvisati nell'utilizzazione degli array si collegano alla poca elasticità/flessibilità insita in questa struttura. Non sempre l'array corrisponde alla scelta migliore.