

**Figura 17.2** Due diverse rappresentazioni grafiche dello stesso grafo: quella di destra fornisce visivamente piú informazioni.

riguardo, possiamo farlo con grande libertà. Ovviamente alcune rappresentazioni grafiche forniscono piú informazione di altre, come è facile dedurre dalla Figura 17.2 dove lo stesso grafo è rappresentato in due forme diverse. Sebbene le due forme siano topologicamente equivalenti, quella di destra ci permette di capire immediatamente che si tratta di una porzione di reticolo quadrato bidimensionale. Sarebbe stato ancora piú difficile capirlo se avessimo scelto una rappresentazione in cui gli archi si incrociano: purtroppo molto spesso ciò risulta inevitabile nel tentativo di “schiacciare” un grafo molto connesso su un foglio. Osserviamo infine che, nel caso peggiore, riconoscere se due grafi sono uguali a meno di una permutazione dei nomi dei loro vertici (isomorfismo dei grafi) è un problema NP (riquadro a pagina 399). Quindi in generale il problema di capire se di un dato grafo esiste una rappresentazione semplice (ad esempio con un piccolo numero di archi che si incrociano) non ha una soluzione immediata.

---

**Quesito 17.2** *Disegnare i grafi*



Quali grafi completi si possono disegnare su un foglio di carta senza che vi sia nessun incrocio tra gli archi? Se potessimo usare un ologramma per rappresentare i grafi (ossia se avessimo a disposizione tre dimensioni spaziali) quali grafi sarebbe possibile rappresentare senza che gli archi si incrocino?

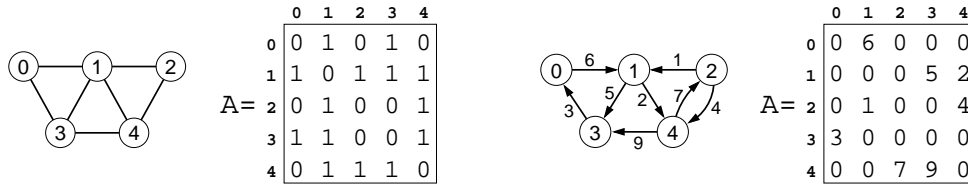
---

## 17.2 Da un grafo a una struttura di dati

Per rappresentare in un programma le informazioni relative a un grafo, sarebbe sufficiente scrivere nella memoria del computer l'insieme  $V$  dei vertici e quello  $E$  degli archi. Senza perdita di generalità possiamo assumere che per un grafo di  $N$  vertici l'insieme  $V$  sia rappresentato dai numeri interi compresi tra 0 e  $N - 1$ . Per rappresentare l'insieme  $E$  non è tuttavia consigliabile seguire la notazione matematica usata nell'equazione (17.1). Infatti, scrivere semplicemente in un array le coppie di vertici corrispondenti agli archi del grafo renderebbe complicate molte delle operazioni che tipicamente si vogliono realizzare su un grafo; ad esempio per sapere quali sono i vicini di un dato vertice dovremmo scorrere tutta la lista.

---

ESTRATTO DAL LIBRO **PROGRAMMAZIONE SCIENTIFICA** DI  
L. M. BARONE, E. MARINARI, G. ORGANTINI E F. RICCI-TERSENGHI  
I diritti di riproduzione e memorizzazione elettronica degli estratti appartengono a Pearson Education Italia S.r.l. e sono riservati per tutti i paesi. La stampa di essi è concessa gratuitamente solo a titolo personale. Ogni altro uso è vietato.



**Figura 17.3** Un grafo non orientato e non pesato (a sinistra) e un secondo grafo orientato e pesato (a destra), ognuno con la relativa matrice di adiacenza.

Esistono fondamentalmente due strutture di dati efficaci dal punto di vista informatico per rappresentare l'insieme  $E$  degli archi di un grafo: la prima usa una *matrice di adiacenza* e la seconda si basa sulle *liste di adiacenza*. Nei paragrafi seguenti studiamo pregi e difetti di queste due strutture di dati per rappresentare un grafo nel computer e quale sia conveniente adottare secondo il problema da studiare.

### 17.2.1 Matrice di adiacenza

Dato un grafo di  $N$  vertici la matrice di adiacenza  $A$  è una matrice  $N \times N$  così definita:

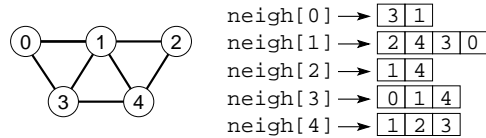
$$A_{ij} = \begin{cases} 1 & \text{se l'arco } (i, j) \in E, \\ 0 & \text{altrimenti.} \end{cases}$$

Nella parte sinistra della Figura 17.3 riportiamo un esempio di grafo non orientato e della relativa matrice di adiacenza. La matrice di adiacenza di un grafo non orientato è sempre simmetrica. È naturale conservare gli elementi di  $A$  in un array bidimensionale  $A[i][j]$ .

Il principale pregio della matrice di adiacenza sta nel fatto che è molto semplice scrivere un programma che la usa; ad esempio, per sapere se i vertici  $i$  e  $j$  sono connessi basta leggere il contenuto dell'elemento  $A[i][j]$ . Il principale svantaggio consiste, invece, nella quantità di memoria necessaria a conservare tale matrice, che è sempre uguale a  $N^2$ , indipendentemente dalla grado medio del grafo. Inoltre per determinare quali siano i vicini del sito  $i$  è necessario esaminare tutti gli elementi della  $i$ -esima riga. A causa di questi svantaggi ne sconsigliamo fortemente l'uso nel caso di grafi sparsi; la matrice risulterebbe piena di elementi nulli e sarebbe inutilmente scomodo tenerla in memoria e lavorarci. Per questo tipo di grafi sono preferibili le liste di adiacenza (Paragrafo 17.2.2).

L'uso della matrice di adiacenza è opportuno nel caso in cui si debba lavorare con grafi densi. In questo caso gli elementi non nulli nella matrice sono una buona frazione del totale e la dimensione della matrice,  $N^2$ , è dello stesso ordine di grandezza dell'insieme degli archi, e quindi non c'è nessun vero "spreco" di memoria. Inoltre, nel caso in cui il grafo denso è anche pesato, la matrice di adiacenza permette di aggiungere l'informazione sui pesi senza nessuna difficoltà: l'elemento di matrice  $A_{ij}$  è posto uguale al peso del legame  $(i, j)$  o pari a zero<sup>1</sup> se tale legame non esiste. Nel caso di grafi orientati, la matrice di

<sup>1</sup>Il valore convenzionale del peso che abbiamo scelto per indicare l'assenza di un legame, si può sostituire con un altro qualora 0 fosse un peso valido.



**Figura 17.4** Lo stesso grafo non orientato e non pesato della Figura 17.3 e la sua rappresentazione in termini di liste di adiacenza.

adiacenza permette di distinguere facilmente gli archi che *partono* del vertice  $i$  (quelli sulla  $i$ -esima riga) da quelli che *arrivano* al vertice  $i$  (quelli sulla  $i$ -esima colonna). Nella parte destra della Figura 17.3 mostriamo un esempio di grafo orientato e pesato, insieme alla sua rappresentazione con una matrice di adiacenza.

## 17.2.2 Liste di adiacenza


Nelle liste di adiacenza il grafo si rappresenta costruendo, per ogni vertice, una lista con i suoi vicini. Ogni lista ha una lunghezza pari al grado del vertice corrispondente. Generalmente l'ordine con cui compaiono nella lista i vicini di un dato vertice non conta, quindi si può usare per ognuna delle  $N$  liste la struttura di dati piú semplice in assoluto, ad esempio quella “a secchio” vista nel Paragrafo 16.1. Nella Figura 17.4 mostriamo un esempio di grafo e della sua rappresentazione in termini di liste di adiacenza: `neigh[]` è un array di puntatori (Paragrafo 6.5.1) in cui ogni elemento punta a una lista di vicini. È conveniente tenere in un array `degree[]` anche i gradi dei vertici, cosicché l'elemento `degree[i]` indica quanti vicini è possibile leggere a partire dalla locazione di memoria puntata da `neigh[i]`.

In totale questa rappresentazione del grafo usa una quantità di memoria pari a  $2(N + M)$ : un vettore di lunghezza  $N$  per i gradi, un altro di pari lunghezza per i puntatori alle liste e  $N$  vettori per le liste di adiacenza, che in totale occupano  $2M$  locazioni di memoria, visto che a ogni arco corrispondono due vertici nelle liste di adiacenza. Come abbiamo già osservato, in generale l'uso della matrice di adiacenza è inefficace se confrontato con quello delle liste di adiacenza. Per tale motivo consigliamo di usare tendenzialmente queste ultime. Nel caso di un grafo pesato le liste di adiacenza devono contenere anche il peso degli archi. Per fare ciò è sufficiente trasformare ogni elemento delle liste di adiacenza in una `struct` che contenga sia il vertice primo vicino sia il peso dell'arco.

---

### Laboratorio 17.1 Liste di adiacenza

---


 Scrivete un programma che sappia leggere un grafo da un file e scriverlo in liste di adiacenza appositamente create. Il file, formattato, contiene nella prima riga il numero dei vertici e in ognuna delle seguenti righe una coppia di numeri interi corrispondenti ai vertici su cui incide un arco del grafo. Quindi stampate l'istogramma dei gradi dei vertici del grafo.

---