

Grafi

Un grafo rappresenta un modello matematico della realta'.

Adatto a descrivere un qualsiasi insieme di elementi su cui esiste una relazione binaria

Esempi:

Popolazione di un paese

Insieme = abitanti

Relazione binaria = relazione di parentela diretta
(grafo non orientato)

Rete ferroviaria

Insieme = localita'

Relazione binaria = esistenza di una connessione diretta
(grafo non orientato)

Corso di algoritmi

Insieme = studenti

Relazione binaria = (Studente)_a ha copiato da (Studente)_b
(grafo orientato)

Definizione

Un **grafo non orientato** e' una coppia ordinata $G=(V,E)$ dove:

- $V=\{v_1,v_2,\dots v_n\}$ e' un insieme finito detto **insieme dei vertici di G**

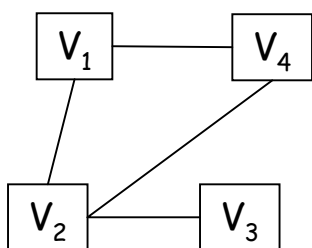
- $E=\{I_1,I_2,\dots I_m\}$, dove ciascun $I_k=(v_i,v_j)$ e' una coppia **non ordinata** di elementi di V con $V_i \neq V_j$, e' un insieme finito detto **insieme degli archi di G** .

Un **grafo orientato** e' una coppia ordinata $G=(V,E)$ dove:

- $V=\{v_1,v_2,\dots v_n\}$ e' un insieme finito detto **insieme dei vertici di G**

- $E=\{I_1,I_2,\dots I_m\}$, dove ciascun $I_k=(v_i,v_j)$ e' una coppia **ordinata** di elementi di V , e' un insieme finito detto **insieme degli archi di G** .

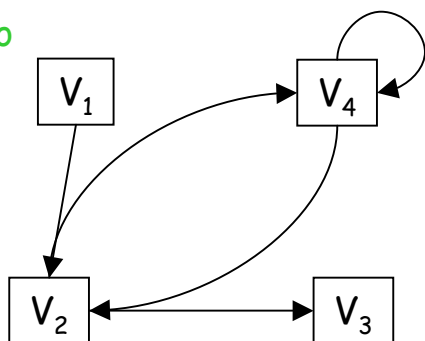
Non -Orientato



$$V=\{v_1,v_2,v_3,v_4\}$$

$$E=\{(v_1,v_2), (v_1,v_4), (v_2,v_3), (v_2,v_4)\}$$

Orientato



cappio

$$V=\{v_1,v_2,v_3,v_4\}$$

$$E=\{(v_1,v_2), (v_2,v_4), (v_4,v_2), (v_4,v_4), (v_2,v_3)\}$$

Alcune definizioni

Se $l=(u,v)$ e' un arco di un grafo si dice che l'arco l e' **incidente** sui vertici u e v .

Se $l=(u,v)$ e' un arco di un grafo si dice che i nodi u e v sono **adiacenti**.

Grado di un vertice = numero di archi incidenti

Cammino da un vertice u ad un vertice u' : sequenza di vertici $\langle v_0 \equiv u, v_1, v_2, \dots, v_k \equiv u' \rangle$ tali che $l_i=(v_{i-1}, v_i) \in E$.

- k = **lunghezza** del cammino
- Se i vertici sono distinti il cammino e' **semplice**
- Se $u \equiv u'$ il cammino viene detto **ciclo**
- Un grafo senza cicli e' detto **aciclico**

- Un grafo e' detto **connesso** se per ogni coppia di vertici v_i e v_j esiste un cammino che unisce v_i e v_j .

Rappresentazione di grafi

1) Liste di Adiacenza

Dato un grafo $G=(V,E)$ introduciamo un vettore $Adj[1.....|V|]$. Per ogni $u \in V$, $Adj[u]$ e' un puntatore alla lista dei vertici adiacenti ad u .

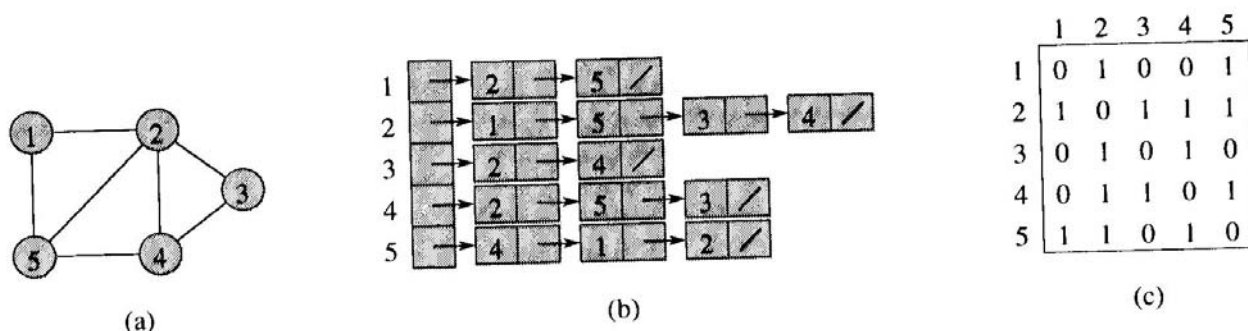


Fig. 23.1 Due rappresentazioni di un grafo non orientato. (a) Un grafo non orientato G con cinque vertici e sette archi. (b) Una rappresentazione con liste di adiacenza di G . (c) La rappresentazione con matrice di adiacenza di G .

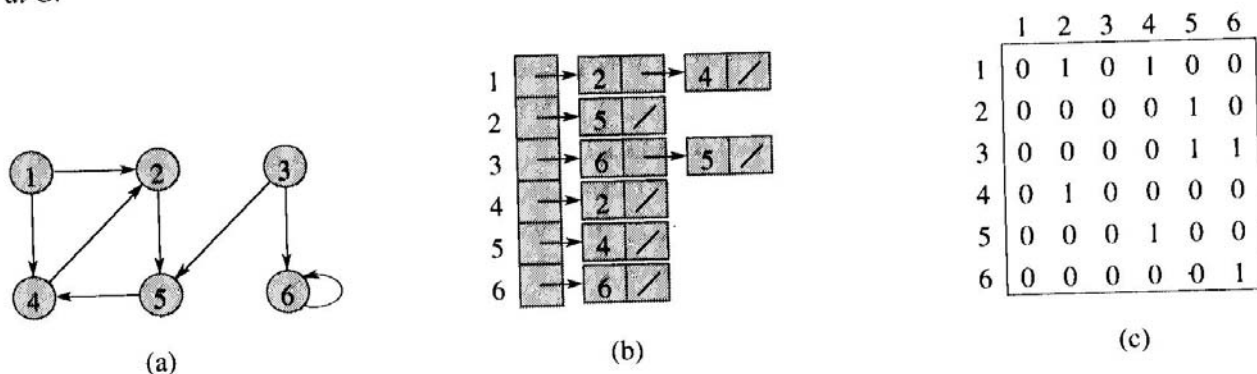


Fig. 23.2 Due rappresentazioni di un grafo orientato. (a) Un grafo orientato G con sei vertici ed otto archi. (b) Una rappresentazione con liste di adiacenza di G . (c) La rappresentazione con matrice di adiacenza di G .

Quantita' di memoria utilizzata:

Sia $n = |V|$ ed $m = |E|$ (N.B. - $m < n^2$)

$S(n,m) = O(n+m)$

Vantaggi - Complessita' spaziale ottimale $S = O(n+m)$

Svantaggi - Verificare se due vertici sono adiacenti costa $O(n)$

2) Matrice di adiacenza

Dato un grafo $G=(V,E)$ assumiamo che i vertici siano numerati in modo arbitrario. Definiamo una matrice $A=a_{ij}$ tale che:

$$a_{ij} = \begin{cases} 1 & \text{se } (v_i, v_j) \in E \\ 0 & \text{altrimenti} \end{cases}$$

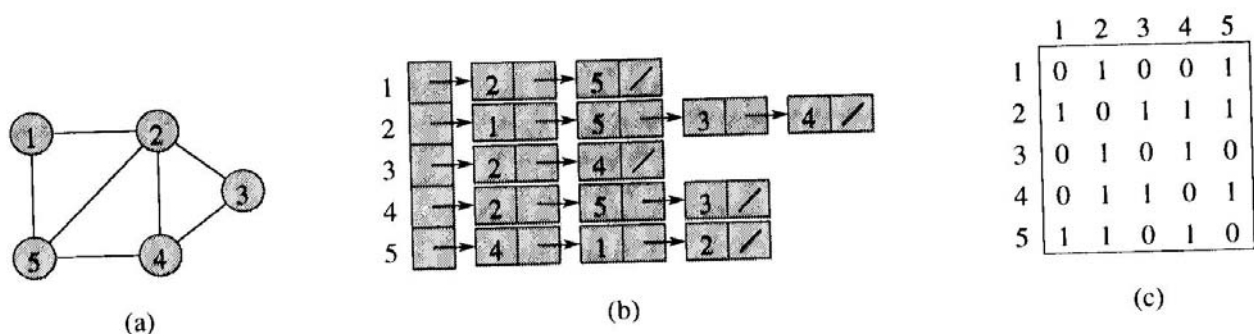


Fig. 23.1 Due rappresentazioni di un grafo non orientato. (a) Un grafo non orientato G con cinque vertici e sette archi. (b) Una rappresentazione con liste di adiacenza di G . (c) La rappresentazione con matrice di adiacenza di G .

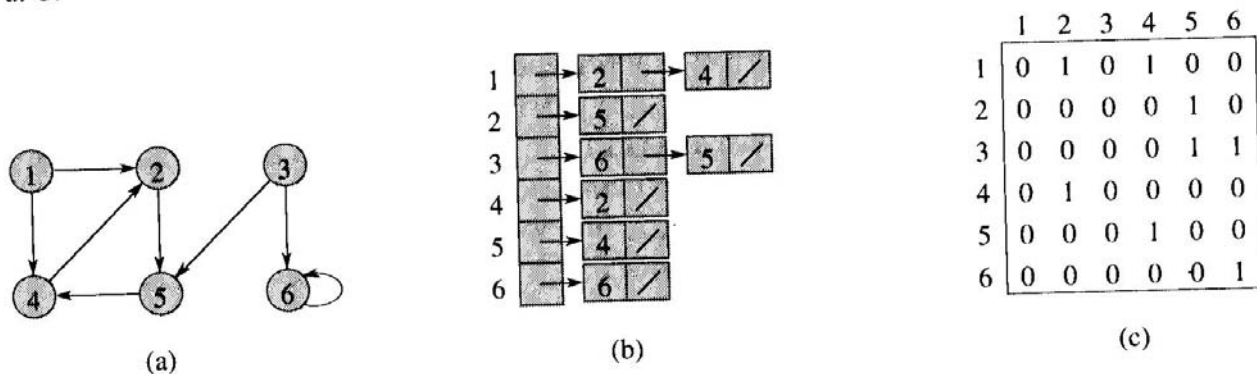


Fig. 23.2 Due rappresentazioni di un grafo orientato. (a) Un grafo orientato G con sei vertici ed otto archi. (b) Una rappresentazione con liste di adiacenza di G . (c) La rappresentazione con matrice di adiacenza di G .

Quantita' di memoria utilizzata:

$$S(n) = \Theta(n^2)$$

Vantaggi - Verificare adiacenza costa $O(1)$

Svantaggi - Se il grafo e' "sparso" (piccolo numero di archi) ho un grosso spreco di memoria

Visita in ampiezza

In inglese Breadth-First Search (BFS)

Algoritmo elementare utilizzato in molte procedure piu' complesse

Input - Un grafo $G=(V,E)$ ed un nodo sorgente $s \in V$.

Output - Un albero BFS $T=(V',E')$ tale che V' e' l'insieme dei vertici di G che sono raggiungibili da S

Notare - un albero e' un grafo connesso ed aciclico.

Proprieta' fondamentale dell'albero BFS -

Ponendo s come radice dell'albero, per ogni $v \in V'$, il cammino nell'albero da s a v e' un cammino minimo da s a v in G , ovvero un cammino che contiene il minimo numero di archi.

Strategia della BFS

La visita in ampiezza colora ogni vertice di bianco, grigio o nero:

- **bianco**: il vertice non e' stato ancora visitato;
- **grigio**: il vertice e' stato visitato ma non sono stati visitati tutti i vertici ad esso adiacenti;
- **nero** : Il vertice e tutti i suoi vertici adiacenti sono stati visitati.

- ✓ All'inizio il nodo sorgente e' grigio, mentre tutti gli altri vertici di G sono bianchi;
- ✓ Memorizzo s in una coda Q ;
- ✓ Analizzo tutti i vertici v adiacenti ad s ed effettuo le seguenti operazioni;
 - Controllo che il colore di u sia bianco (non sono gia' stati esaminati);
 - Assegno ai vertici v colore grigio;
 - $\text{predecessor}[v] \leftarrow s$
 - Inserisco i vertici v visitati nella coda Q .
- ✓ Assegno colore nero ad s ;
- ✓ Elimino s dalla coda;
- ✓ Ripeto le operazioni descritte per tutti i vertici presenti nella coda Q ;

Vado avanti fino a che ci sono vertici grigi (con vertici adiacenti non esaminati) nella coda Q

Procedura BFS

La procedura assume che il grafo sia rappresentato usando liste di adiacenza;

Mantiene diverse strutture dati aggiuntive associate ad ogni vertice del grafo - $color[u]$

$predecessor[u]$

$distanza[u]$ - distanza del vertice u dalla sorgente.

Mantiene una coda Q per la gestione dei vertici grigi.

BFS(G, s)

For ogni vertice $u \in V[G] - \{s\}$

do $colore[u] \leftarrow white$

$distanza[u] \leftarrow \infty$

$predecessore[u] \leftarrow NIL$

$Colore[s] \leftarrow grey$

$Distanza[s] \leftarrow 0$

$Predecessore[s] \leftarrow NIL$

$Q \leftarrow \{s\}$

While $Q \neq \emptyset$

do $u \leftarrow head[Q]$

for ogni $v \in Adj[u]$

do if $colore[v] = white$

then $colore[v] \leftarrow grey$

$distanza[v] \leftarrow distanza[u] + 1$

$predecessore[v] \leftarrow u$

enqueue(Q, v)

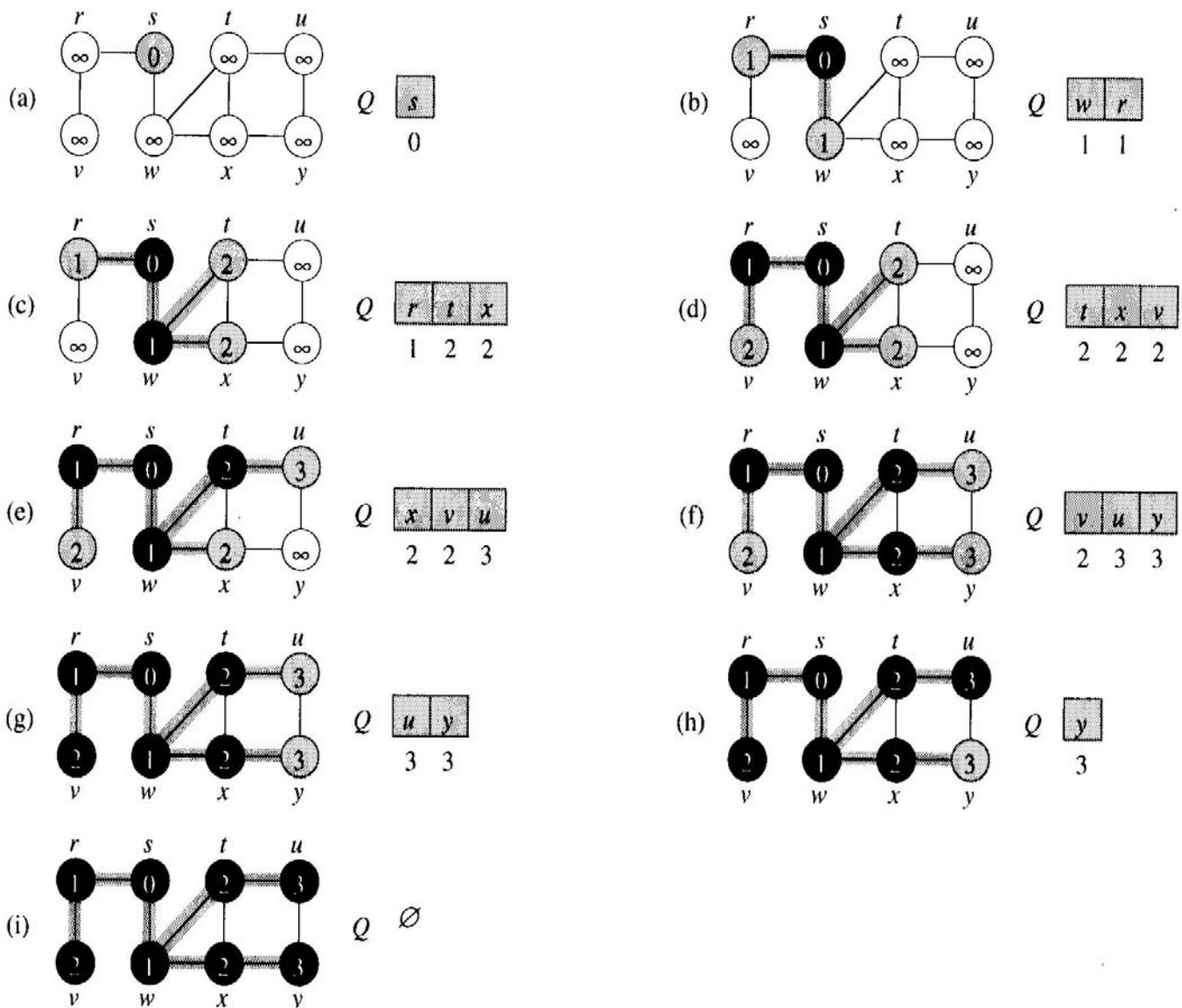
dequeue(Q)

$colore[u] \leftarrow black$

Complessita' dell'algoritmo - La lista di adiacenza di ogni vertice viene scandita al massimo una volta

$$T(n) = O(n+m)$$

Esempio



Notare -

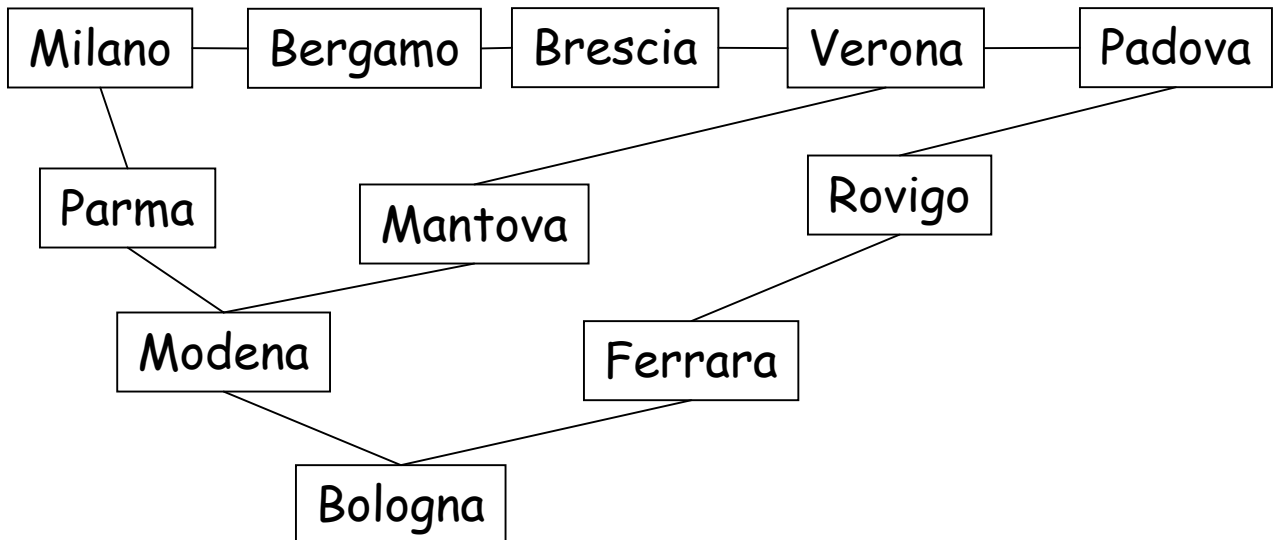
- Vengono raggiunti tutti e soli i vertici connessi ad s ;
- La distanza di un vertice v da s nell'albero BFS è pari alla minima distanza di v da s in G .

Infatti:

- prima visito *tutti* i nodi adiacenti alla radice
(**tutti e soli** i nodi con distanza minima da s uguale ad 1);
- poi *tutti* i nodi adiacenti ai nodi di distanza 1;
(**tutti e soli** i nodi con distanza minima da s uguale a 2);
- etc.

Esempio di applicazione ...

Strade e superstrade



Ipotizzando che tutti i tratti stradali siano lunghi uguali...
Qual e' la strada piu' breve da Ferrara a Milano?