Algoritmi e Strutture Dati

Grafi e visite di grafi

Fabio Patrizi

Grafo: definizione

Un grafo **G=(V,E)** consiste in:

- un insieme **V** di vertici (o nodi)
- un insieme **E** di coppie di vertici, detti **archi** (o *spigoli*): ogni arco connette due vertici

Grafo orientato

Grafo orientato (o diretto): ogni arco è *orientato* e rappresenta relazioni *orientate* tra coppie di oggetti

Esempio:

V={persone che vivono in Italia},

E={(x,y) tale che x ha inviato una mail a y}

Esempio:

V={persone che vivono in Italia},

E={(x,y) tale che x è padre di y}

Grafo non orientato

Grafo non orientato (o non diretto): gli archi non hanno un orientazione (relazioni simmetriche)

Esempio:

V={persone che vivono in Italia},
E={coppie di persone che si sono strette la mano}

Esempio:

V={persone che vivono in Italia},
E={coppie di fratelli}

Terminologia

relazione **simmetrica** → grafo **non orientato**

relazione **non simmetrica** → grafo **orientato**

Sia **G** un grafo. Denotiamo con:

- **n** il numero di vertici di **G**
- **m** il numero di archi di **G**
- **V(G)** l'insieme dei nodi di **G**
- **E(G)** l'insieme degli archi di **G**

- Quindi: **|V(G)| = n** e **|E(G)| = m**

Adiacenza ed Incidenza

- Sia **(x,y)** un arco del grafo **G**, diciamo che **(x,y)** è **incidente** sui nodi **x** e **y**;
- Se **G** è un grafo *orientato*, allora diciamo che **(x,y)** **esce** da **x** ed **entra** in **y**;
- Se **G** è un grafo *non orientato* allora diremo che **x** è **adiacente** ad **y** e che **y** è **adiacente** a **x**, ovvero **x** ed **y** sono **adiacenti**;
- Dato un nodo **v** chiameremo i nodi adiacenti a **v** i **vicini** di **v**

Grado di un vertice (1/2)

•Il **grado** di un vertice **v** in un grafo è dato dal numero di archi **incidenti** su **v.**

•Denotiamo con δ**(v)** il grado del vertice **v.**

•Se sommiamo il grado di tutti i nodi di un grafo **G** *non orientato*, andiamo a contare ogni arco **2** volte (es: (x,y) una volta per x ed una volta per y).

•Quindi:

$$\sum_{v \in V} \delta(v) = 2m$$

Grado di un vertice (2/2)

Se il grafo G è *orientato,* parleremo di:

- ***Grado in uscita*** di **v**, intendendo con questo il numero di archi che escono da **v** e lo indicheremo con $\delta_{out}(v)$
- ***Grado in entrata*** di **v**, intendendo con questo il numero di archi che ertano in **v** e lo indicheremo con $\delta_{in}(v)$
- Il ***grado*** di un vertice sarà dato dalla somma dei due:

$$\delta(v) = \delta_{in}(v) + \delta_{out}(v)$$

Cammini

Un **cammino** in un grafo **G** da un vertice **x** ad un vertice **y** è dato da una sequenza di vertici $[v_0, v_1, \ldots, v_k]$, con $v_0 = x$ e $v_k = y$, tale che per ogni $1 \leq i \leq k$, la coppia (v_{i-1}, v_i) appartiene a **E(G)**

In tal caso diremo che il cammino è di **lunghezza k.**
La lunghezza del più **corto cammino** tra due vertici è la **distanza** tra i vertici

Diremo che un cammino è ***semplice*** se tutti i suoi vertici sono distinti.

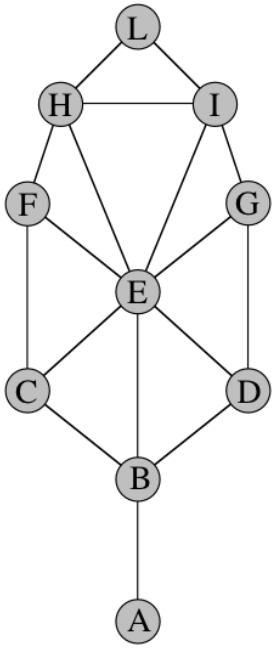
Indichiamo il cammino da **x** a **y** con **x** → **y**

Se esiste un cammino da **x** ad **y** diremo che **y** è **raggiungibile** da **x**

Cicli

- Un cammino **[v_0, v_1, … , v_k]** tale che **v_0= v_k** , con k ≥ 1, è detto ***ciclo***
- Il ciclo è detto **semplice** se tutti i nodi **v_1, … , v_{k-1}** sono distinti.
- Un grafo è ***aciclico*** se **non** contiene cicli.

Esempio



Il grafo in figura è un grafo **non** orientato

Il vertice **H** ha grado 4: $\delta(\mathbf{v}) = \mathbf{4}$

Esiste un cammino semplice di lunghezza 4 tra il vertice **A** ed il vertice **L**, contenente i vertici **A, B, E, I, L** e gli archi **(A,B), (B,E), (E,I),** ed **(I,L)**

Dato che 4 è anche la lunghezza del cammino più corto tra **A** ed **L**, allora **A** ed **L** sono a ***distanza*** **4**.

Connettività e connettività forte

•Un grafo *non orientato* **G = (V, E)** si dice ***connesso*** se esiste un cammino tra ogni coppia di vertici in **G**.

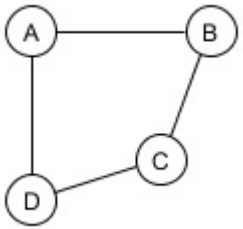
•Un grafo *orientato* **G = (V, E)** si dice ***fortemente connesso*** se esiste un cammino (orientato) tra ogni coppia di vertici in **G**.

•Diremo che un vertice **x** è ***fortemente connesso*** ad un vertice **y** se esiste un cammino (orientato) da **x** ad **y** ed un cammino (orientato) da **y** a **x.**

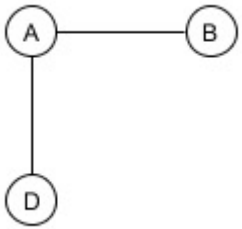
Sottografi

- Diciamo che un grafo **G' = (V', E')** è un ***sottografo*** di un grafo **G = (V, E)** se **V' ⊆ V** e **E' ⊆ E**.
  In questo caso useremo la notazione **G' ⊆ G**.

- Dato un grafo **G = (V, E)** ed un insieme di vertici **V' ⊆ V**, il ***sottografo*** di **G** ***indotto*** da **V'** è definito come il grafo **G' = (V', E')** dove **E' = { (x,y) ∈ E tale che x,y ∈ V' }.**



G



G'

G' è il sottografo di G indotto da V' = {A, B, D}

Strutture dati
per rappresentare grafi

Operazioni sui grafi

tipo `Grafo`:
dati:
un insieme di vertici (di tipo *vertice*) e un insieme di archi (di tipo *arco*).

operazioni:

`numVertici`() → *intero*
restituisce il numero di vertici presenti nel grafo.

`numArchi`() → *intero*
restituisce il numero di archi presenti nel grafo.

`grado`(*vertice* v) → *intero*
restituisce il numero di archi incidenti sul vertice v.

`archiIncidenti`(*vertice* v) → $\langle$*arco*, *arco*, ..., *arco*$\rangle$
restituisce, uno dopo l'altro, gli archi incidenti sul vertice v.

`estremi`(*arco* e) → $\langle$*vertice*, *vertice*$\rangle$
restituisce gli estremi x e y dell'arco $e = (x, y)$.

`opposto`(*vertice* x, *arco* e) → *vertice*
restituisce y, l'estremo dell'arco $e = (x, y)$ diverso da x.

`sonoAdiacenti`(*vertice* x, *vertice* y) → *booleano*
restituisce `true` se esiste l'arco (x, y), e `false` altrimenti

`aggiungiVertice`(*vertice* v)
inserisce un nuovo vertice v.

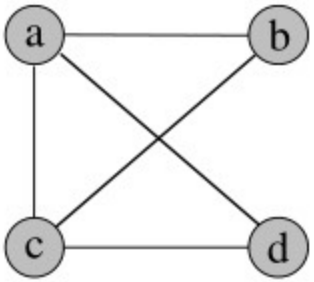
`aggiungiArco`(*vertice* x, *vertice* y)
inserisce un nuovo arco tra i vertici x e y.

`rimuoviVertice`(*vertice* v)
cancella il vertice v e tutti gli archi ad esso incidenti.

`rimuoviArco`(*arco* e)
cancella l'arco e.

A titolo di esempio consideriamo le seguenti operazioni su grafi non orientati

Lista di archi



(a,b)
(c,a)
(b,c)
(c,d)
(a,d)

I vertici del grafo sono rappresentati come record in una lista. Ogni record contiene le informazioni che si voglio associare al nodo (Es. etichetta)

Gli archi del grafo sono rappresentati come record di una lista. Ogni record contiene i puntatori ai vertici che sono gli estremi dell'arco

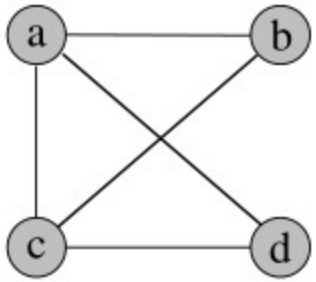
Lo spazio utilizzato sarà O(n+m)

Rappresentazione poco efficiente → molte operazioni richiedono di scandire l'intera lista degli archi.

Prestazioni della lista di archi

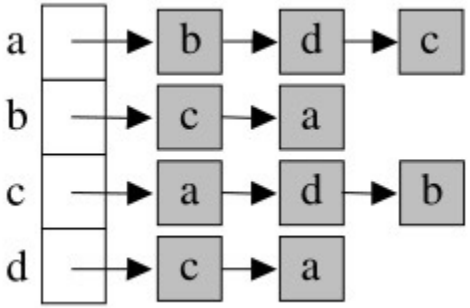
Operazione	Tempo di esecuzione
`grado`(v)	$O(m)$
`archiIncidenti`(v)	$O(m)$
`sonoAdiacenti`(x, y)	$O(m)$
`aggiungiVertice`(v)	$O(1)$
`aggiungiArco`(x, y)	$O(1)$
`rimuoviVertice`(v)	$O(m)$
`rimuoviArco`(e)	$O(m)$

Liste di adiacenza



Ogni vertice **v** ha una lista contenente i suoi vertici adiacenti, ovvero tutti i vertici **u** per cui esiste un arco **(v,u)**.

Vi saranno **n** liste, una per ogni vertice.
Lo spazio utilizzato sarà O(n+m)



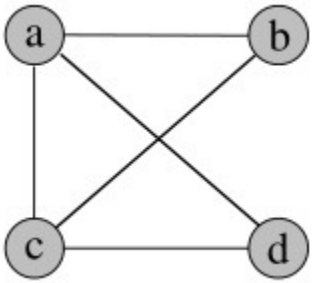
Efficiente per quelle operazioni in cui si chiede di visitare un grafo esaminando gli archi incidenti sui vertici.

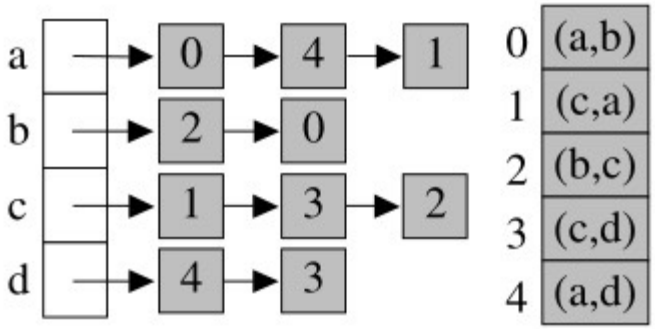
Contro: per sapere se due nodi sono adiacenti devo scandire tutta la lista di adiacenza.

Prestazioni delle liste di adiacenza

Operazione	Tempo di esecuzione
`grado`(v)	$O(\delta(v))$
`archiIncidenti`(v)	$O(\delta(v))$
`sonoAdiacenti`(x, y)	$O(\min\{\delta(x), \delta(y)\})$
`aggiungiVertice`(v)	$O(1)$
`aggiungiArco`(x, y)	$O(1)$
`rimuoviVertice`(v)	$O(m)$
`rimuoviArco`$(e = (x, y))$	$O(\delta(x) + \delta(y))$

Liste di incidenza





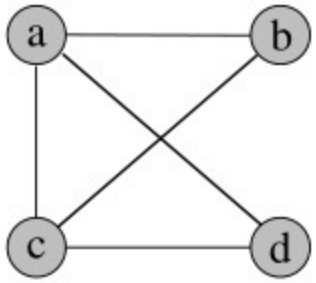
Combinazione tra la rappresentazione con liste di archi e liste di adiacenza: in aggiunta alla rappresentazione con lista di archi, memorizziamo per ogni vertice **v** una lista di puntatori agli archi incidenti a **v.**

Lo spazio utilizzato sarà leggermente maggiore, ma comunque nell'ordine di O(n+m)

Prestazioni della lista di incidenza

Operazione	Tempo di esecuzione
`grado`(v)	$O(\delta(v))$
`archiIncidenti`(v)	$O(\delta(v))$
`sonoAdiacenti`(x, y)	$O(\min\{\delta(x), \delta(y)\})$
`aggiungiVertice`(v)	$O(1)$
`aggiungiArco`(x, y)	$O(1)$
`rimuoviVertice`(v)	$O(m)$
`rimuoviArco`$(e = (x, y))$	$O(\delta(x) + \delta(y))$

Matrici di adiacenza



	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	1
d	1	0	1	0

Assumiamo che i vertici del grafo corrispondano a numeri interi da 1 ad n.

Sia **M** una matrice **n×n** le cui righe e colonne sono "indicizzate" dai vertici del grafo. Avremo:

$$M[u,v] = \begin{cases} 1 \text{ se } (u,v) \text{ è un arco di } G \\ 0 \text{ altrimenti} \end{cases}$$

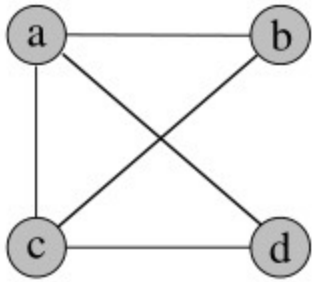
Utilizziamo molto spazio (matrice **n×n**) però possiamo verificare la presenza di un arco (u,v) in tempo contante.

Utile se si voglio adottare tecniche basate su calcoli algebrici (ES: prodotto tra matrici)

Prestazioni della matrice di adiacenza

Operazione	Tempo di esecuzione
`grado`(v)	$O(n)$
`archiIncidenti`(v)	$O(n)$
`sonoAdiacenti`(x, y)	$O(1)$
`aggiungiVertice`(v)	$O(n^2)$
`aggiungiArco`(x, y)	$O(1)$
`rimuoviVertice`(v)	$O(n^2)$
`rimuoviArco`(e)	$O(1)$

Matrici di incidenza



	(a,b)	(c,a)	(b,c)	(c,d)	(a,d)
a	1	1	0	0	1
b	1	0	1	0	0
c	0	1	1	1	0
d	0	0	0	1	1

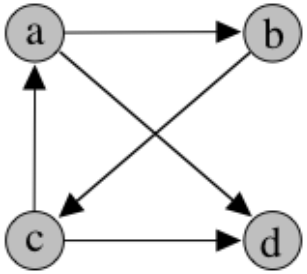
La matrice di incidenza è una matrice **n×m** le cui righe sono "indicizzate" dai vertici e le colonne sono "indicizzate" dagli archi. La posizione **M[v,e]** sarà uguale ad **1** se l'arco **e** incide sul vertice **v**.

- ogni colonna ha esattamente due 1: la colonna corrispondente all'arco **(x,y)** avrà un 1 nella riga corrispondente ad **x** ed un 1 nella riga corrispondente ad **y**

Prestazioni della matrice di incidenza

Operazione	Tempo di esecuzione
`grado`(v)	$O(m)$
`archiIncidenti`(v)	$O(m)$
`sonoAdiacenti`(x, y)	$O(m)$
`aggiungiVertice`(v)	$O(nm)$
`aggiungiArco`(x, y)	$O(nm)$
`rimuoviVertice`(v)	$O(nm)$
`rimuoviArco`(e)	$O(n)$

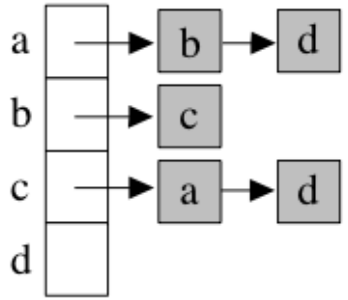
Rappresentazione di grafi orientati



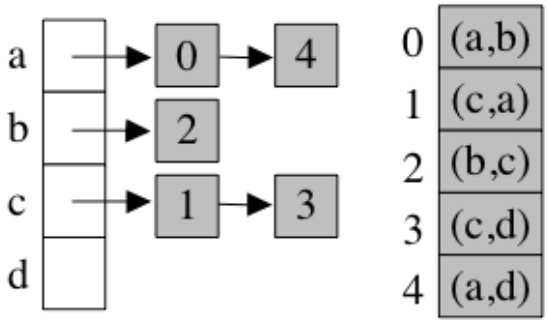
(a) Grafo orientato G

(a,b)
(c,a)
(b,c)
(c,d)
(a,d)

(b) Lista di archi di G



(c) Liste di adiacenza di G



(d) Liste di incidenza di G

	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	1
d	0	0	0	0

(e) Matrice di adiacenza di G

	(a,b)	(c,a)	(b,c)	(c,d)	(a,d)
a	1	-1	0	0	1
b	-1	0	1	0	0
c	0	1	-1	1	0
d	0	0	0	-1	-1

(f) Matrice di incidenza di G

Visite di grafi

Scopo e tipi di visita

- Una visita (o attraversamento) di un grafo G permette di esaminare i nodi e gli archi di G in modo sistematico
- Problema di base in molte applicazioni
- Esistono vari tipi di visite con diverse proprietà: in particolare:
  - **visita in ampiezza** (BFS=breadth first search)
  - **visita in profondità** (DFS=depth first search)
- Definiamo prima un algoritmo di visita "*generico*".

Algoritmo di visita generica

Assumiamo che ogni nodo di **G** sia raggiungibile da **s**

algoritmo visitaGenerica *(vertice s)*
    M,F: insiemi vuoti di vertici
    aggiungi *s* ad *F* e *M*
    while ($F \neq \emptyset$) **do**
        estrai un vertice *u* da F
        visita *u*
        for each (arco *(u,v)* in *G*) **do**
            if ($v \notin M$) **then**
                aggiungi *v* ad *F* e *M*

Algoritmo di visita generica

1. L'insieme *F* è detto **frangia** (o frontiera)

2. I nodi di *M* sono detti **marcati**: sono i nodi già inseriti nella frangia

3. Il sistema di marcatura garantisce che nessun nodo venga inserito nella frangia (e quindi visitato) più di una volta

Costo della visita

Il tempo di esecuzione dipende dalla struttura dati usata per rappresentare il grafo.

In particolare, la visita di un grafo G = (V,E) con **n** vertici e **m** archi costerà:

- **O(mn)** se il grafo è rappresentato come una **lista di archi**
- **O(m+n)** se il grafo è rappresentato come una **lista di adiacenza o di incidenza**
- **O(n^2)** se il grafo è rappresentato come una **matrice di adiacenza**

La rappresentazione mediante lista di adiacenza (o incidenza) è quindi la più efficiente rispetto alla visita di una grafo

Visite particolari

- Se la frangia F è implementata come **coda** si ha la visita in ampiezza (BFS)
- Se la frangia F è implementata come **pila** si ha la visita in profondità (DFS)

Visita in ampiezza

Visita in ampiezza

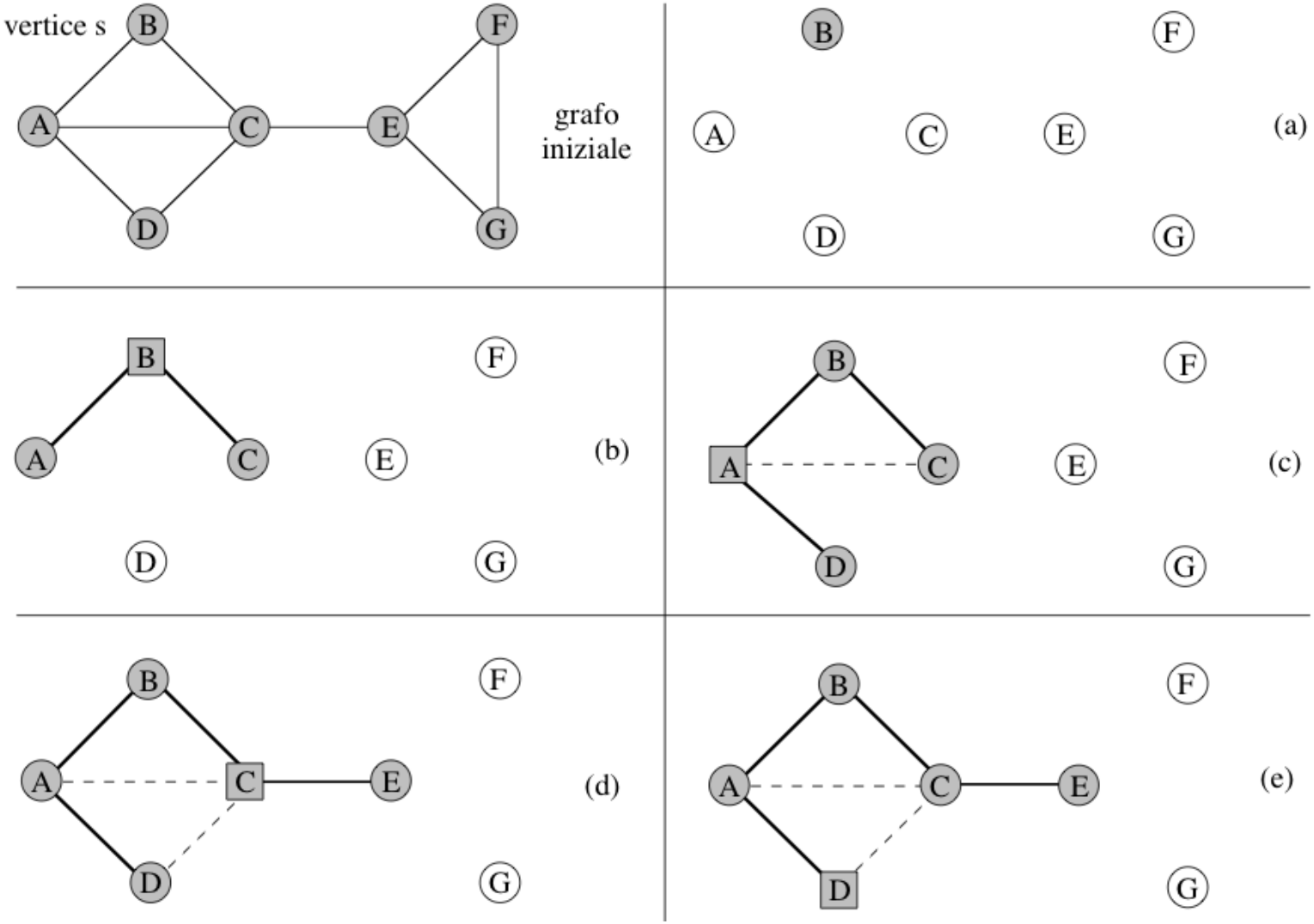
La visita procede in ampiezza sul grafo a partire dai vertici incontrati

La frangia è implementata come una **coda** (politica di accesso FIFO)

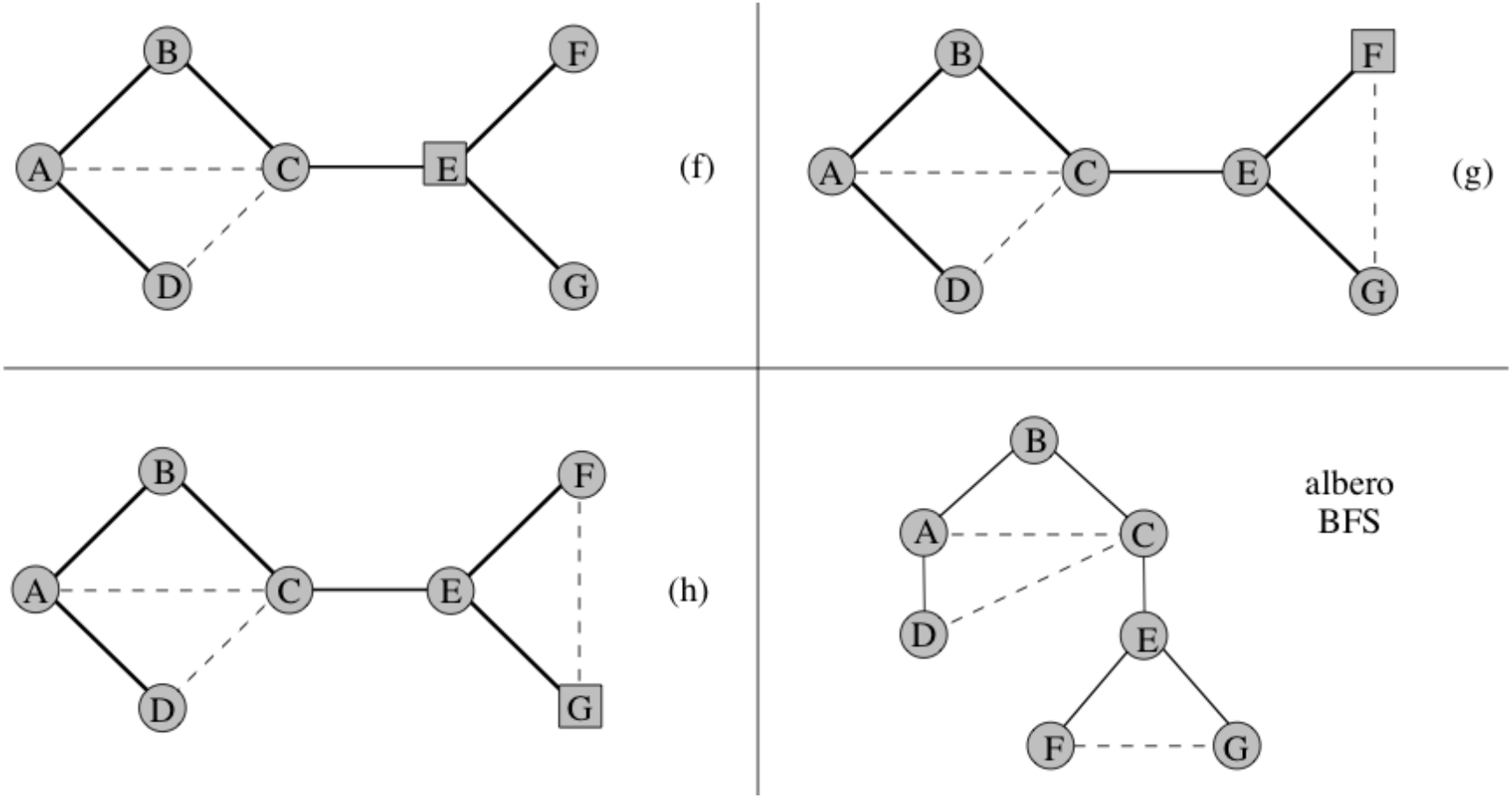
Visita in ampiezza: algoritmo

algoritmo visitaBFS *(vertice s)*
    M: insieme vuoto di vertici
    F: coda vuota di vertici
    F.enqueue(s)
    aggiungi *s* ad *M*
    while ($F \neq \emptyset$) **do**
        u = F.dequeue()
        visit *u*
        for each (arco *(u,v)* in *G*) **do**
            if ($v \notin M$) **then**
                F.enqueue(v)
                aggiungi *v* ad *M*

Esempio: grafo non orientato (1/2)



Esempio: grafo non orientato (2/2)

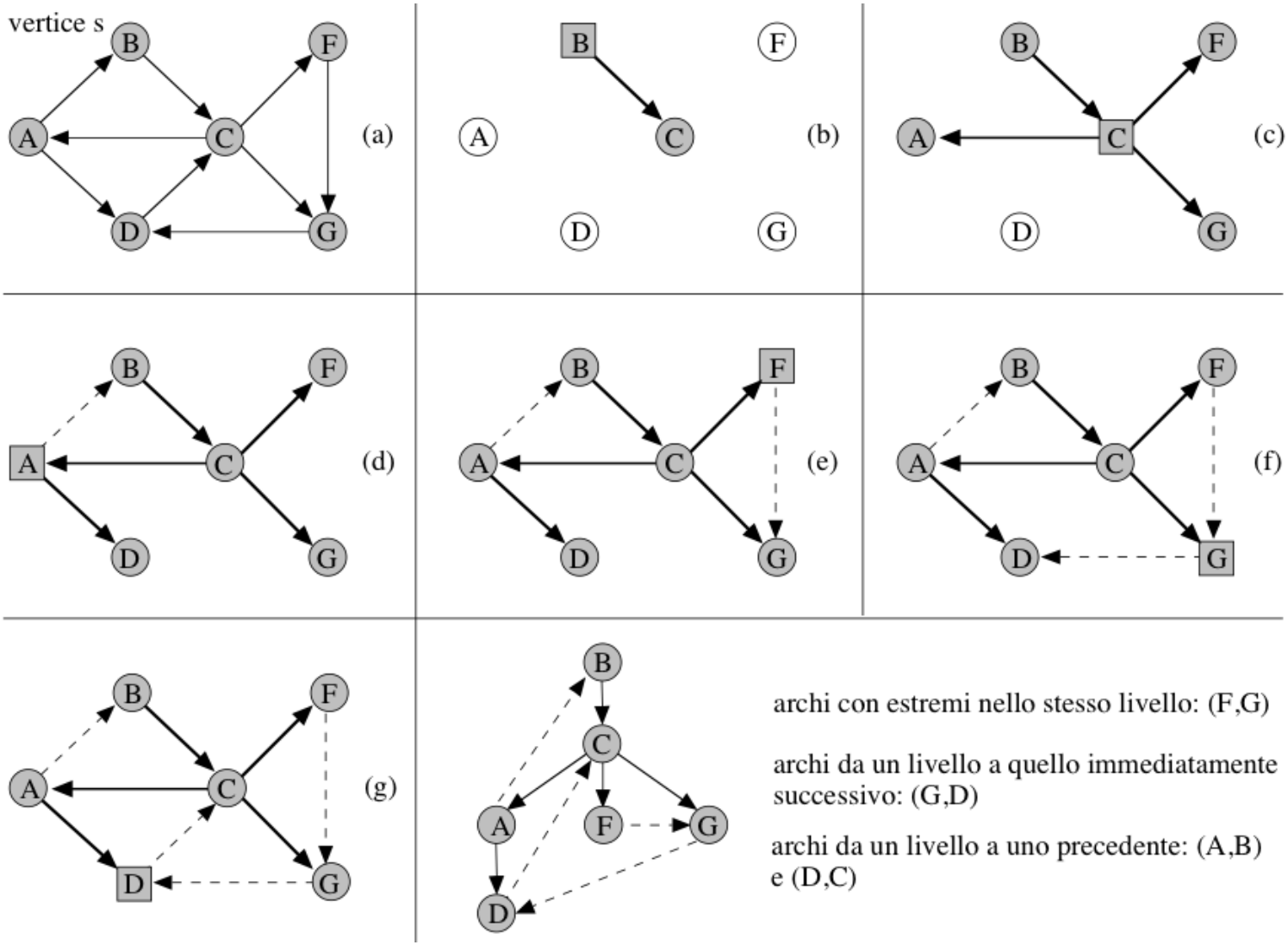


Visita in ampiezza: grafi orientati

La visita in ampiezza può essere applicata anche ai grafi orientati.

Nell'operazione di scelta dei vicini di un vertice **v** (implementata nel passo 8 dell'algoritmo) si dovranno scegliere **solo** gli archi **(v, w)** *uscenti* da **v**

Esempio: grafo orientato



Visita in profondità

Visita in profondità

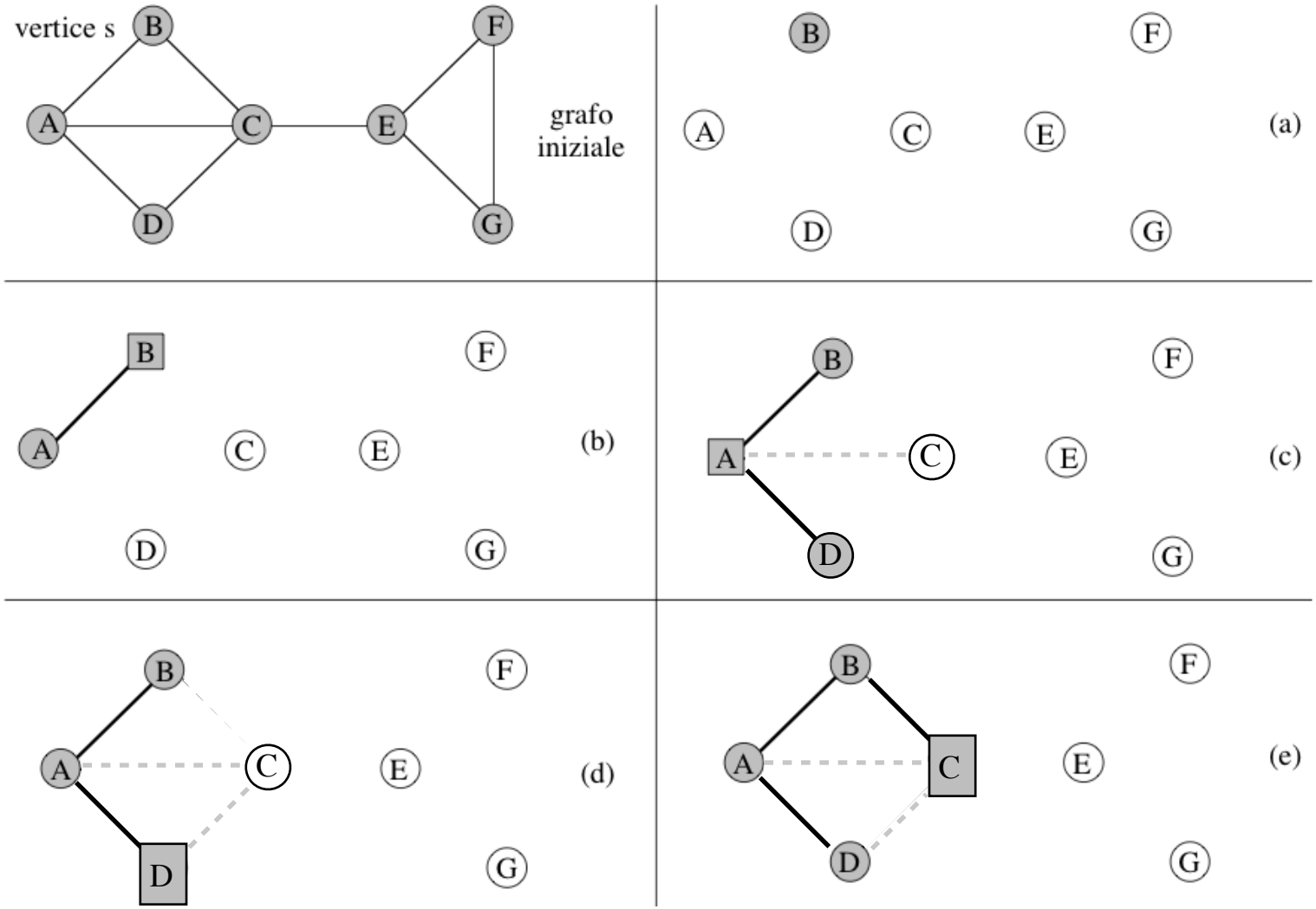
La visita procede in profondità sul grafo a partire dai vertici incontrati.

La frangia *F* è implementata come una **pila** (politica di accesso LIFO).

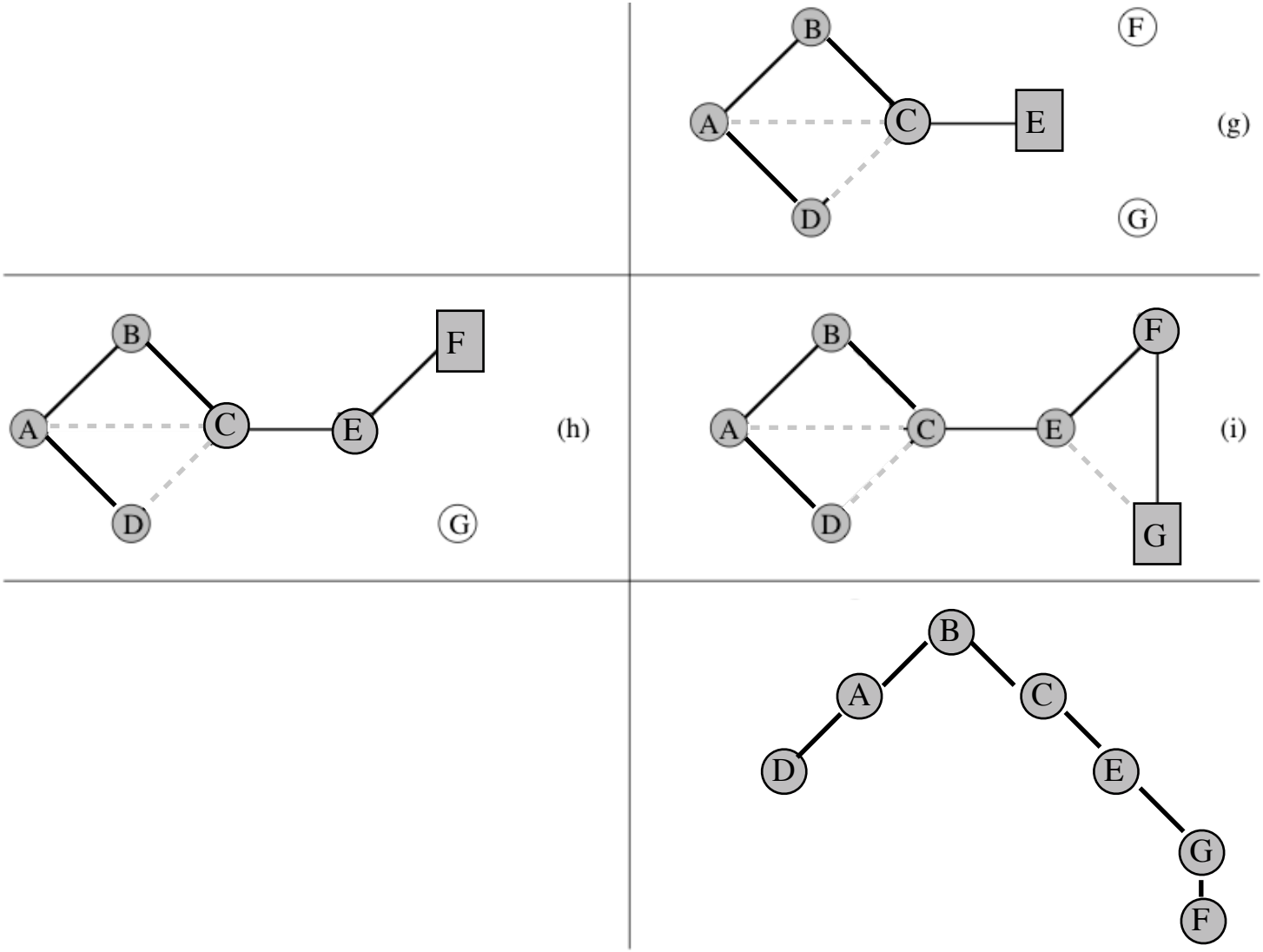
Visita in profondità

algoritmo visitaDFS *(vertice s)*
    M: insieme vuoto di vertici
    F: pila vuota di vertici
    F.push(s)
    aggiungi *s* ad *M*
    while ($F \neq \emptyset$) **do**
        u = F.pop()
        visita *u*
        for each (arco *(u,v)* in *G*) **do**
            if ($v \notin M$) **then**
                F.push(v)
                aggiungi *v* ad *M*

Esempio: grafo non orientato (1/2)



Esempio: grafo non orientato (2/2)



Visita in profondità: versione ricorsiva

Quando effettuiamo la chiamata ricorsiva, dobbiamo passare l'insieme dei nodi marcati

Questo permette alla nuova chiamata di verificare se un nodo è già stato marcato

Visita in profondità: versione ricorsiva

procedura visitaDFSricorsiva *(vertice u, insieme di vertici M)*

visita *u*
aggiungi *u* ad *M*
for each *(arco (u,v)* $\in$ *G)* **do**
if *(v* $\notin$ *M)* **then**
visitaDFSricorsiva(*v*,*M*)

algoritmo visitaDFS(*vertice u*)

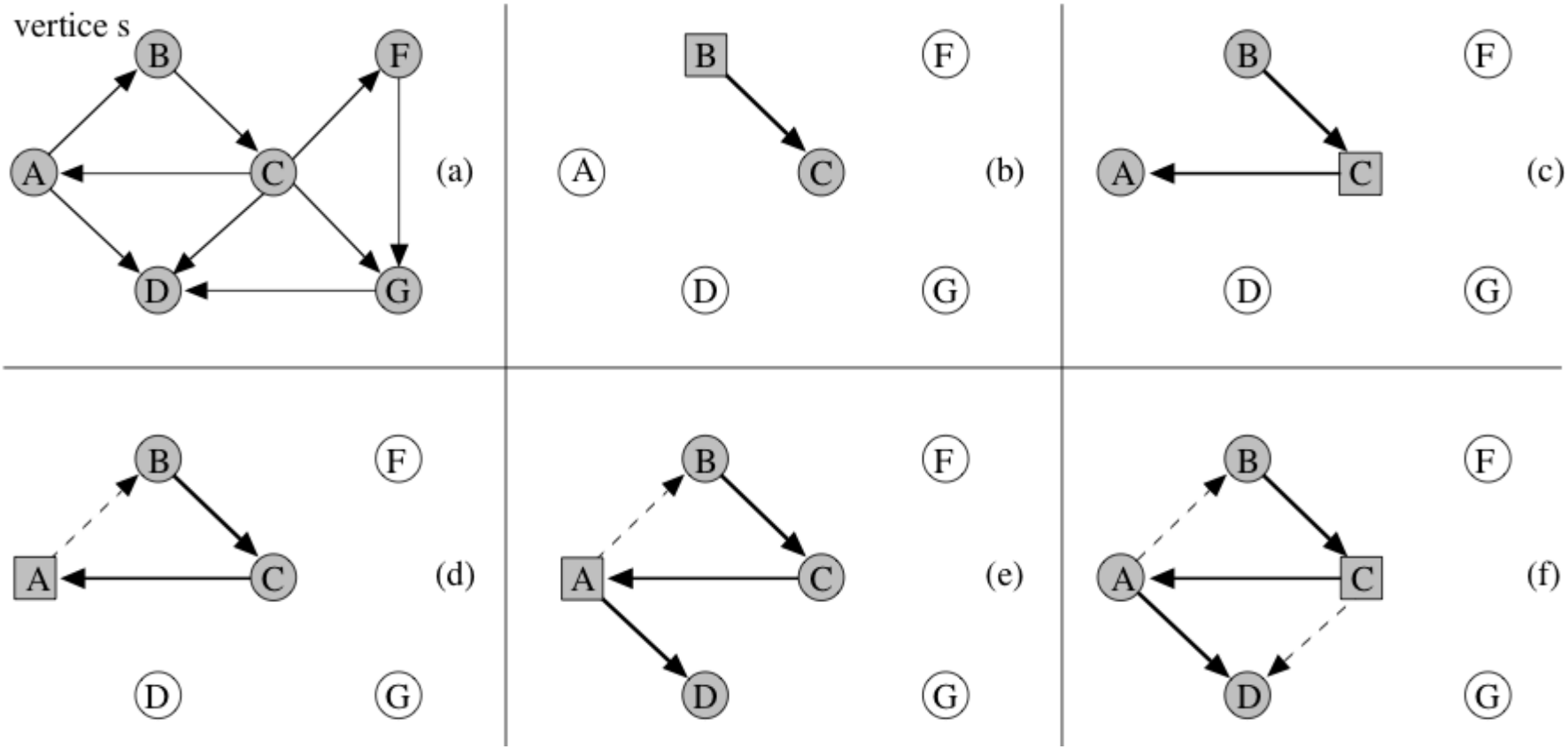
M: insieme vuoto di nodi
visitaDFSricorsiva(*u*,*M*)

Visita in profondità: grafi orientati

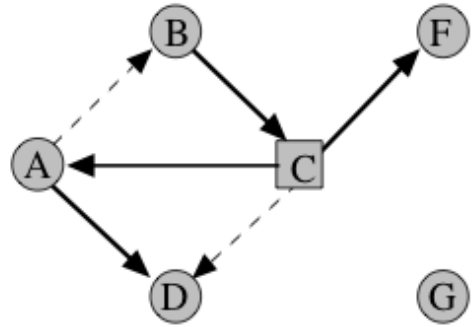
Anche la visita in profondità può essere applicata ai ***grafi orientati***.

Anche in questo caso nell'operazione di scelta dei vicini di un vertice **v** si dovranno scegliere **solo** gli archi **(v, w)** *uscenti* da **v**

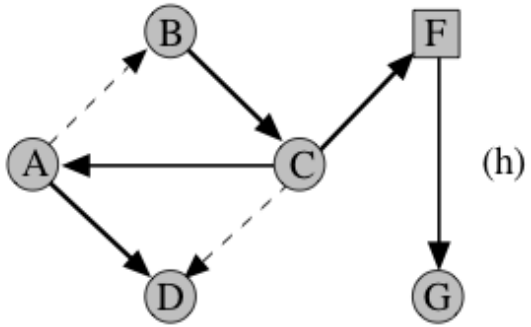
Esempio: grafo orientato (1/2)



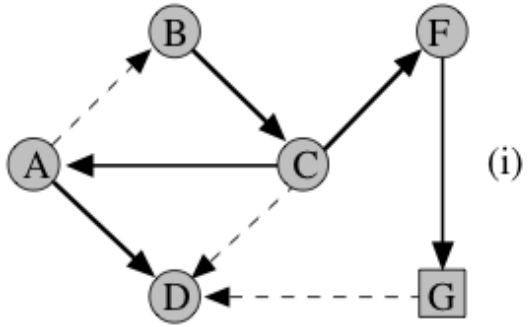
Esempio: grafo orientato (2/2)



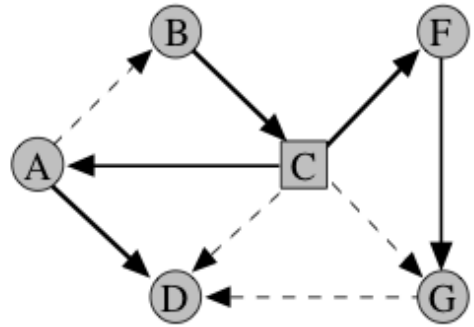
(g)



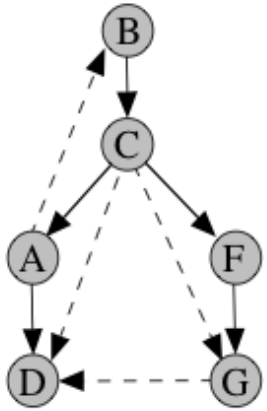
(h)



(i)



(j)



archi in avanti: (C,D) e (C,G)

archi all'indietro: (A,B)

archi trasversali a sinistra: (G,D)

La relazione "raggiungibile da"

Proprietà: La relazione di ***raggiungibilità*** tra vertici di un grafo orientato è una relazione di ***equivalenza***.

- **Riflessività**: ogni nodo è *raggiungibile da* se stesso;
- **Simmetria**: se **v** è *raggiungibile da* **u**, allora **u** è *raggiungibile da* **v**
- **Transitività**: se **v** è *raggiungibile da* **u** e **u** è *raggiungibile da* **w** allora **v** è *raggiungibile da* **w**

Componenti connesse

Le **componenti connesse di un grafo non orientato G=(V,E)** sono le *classi di equivalenza* dei suoi vertici rispetto alla relazione "raggiungibile da"

Ricordiamo che:

Un grafo *non orientato* **G = (V, E)** si dice ***connesso*** se esiste un cammino tra ogni coppia di vertici in **G**.

Un grafo *non orientato* è **connesso** se e solo se ha una sola **componente connessa**.

Componenti connesse

Per verificare se un grafo **G** con **n** nodi è connesso, è sufficiente eseguirne una visita *memorizzando l'insieme M dei nodi marcati*

Se *M* contiene **n** nodi, allora *tutti i nodi sono stati raggiunti*, quindi **G** è connesso

Altrimenti, **G** *non è* connesso

Se un grafo non è connesso, per visitarne tutti i nodi è sufficiente eseguire la visita partendo prima da un nodo **s**, calcolando così *M*, poi da un nodo s', non in *M*, calcolando un nuovo *M,* e così via

La relazione "fortemente connesso"

Un vertice **x** è ***fortemente connesso*** ad un vertice **y** se esiste un cammino (orientato) da **x** ad **y** ed un cammino (orientato) da **y** a **x**.

Proprietà: La relazione di ***connettività forte*** tra vertici in un grafo orientato è una relazione di ***equivalenza*** (*riflessiva*, *simmetrica* e *transitiva*)

Componenti fortemente connesse

Definizione: Sia G = (V,E) un grafo orientato. Le componenti fortemente connesse di G sono le classi di equivalenza della relazione di connettività forte:

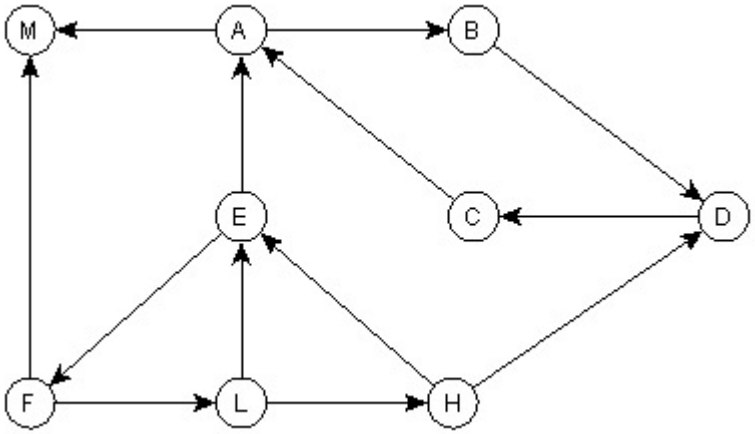
[v] = { u tale che **u** è un nodo di **G** fortemente connesso a **v }**

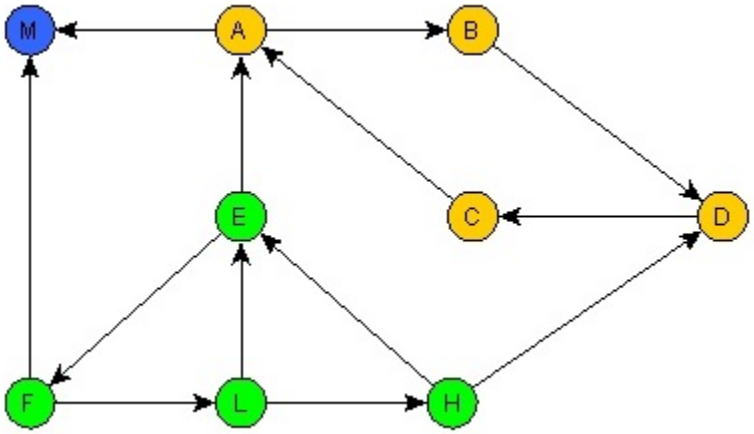
Dalla definizione di *componenti fortemente connesse* segue che un grafo **G** è ***fortemente connesso*** se e solo se ha una sola *componente fortemente connessa*

Componenti fortemente connesse

Esempio:

notare che una componente fortemente connessa può essere formata anche da un solo nodo (nodo M)





Calcolare le componenti fortemente connesse

Per calcolare *la componente fortemente connessa* contenente il vertice **x** operiamo calcolando i seguenti due insiemi:

- I discendenti **D(x)**, ovvero tutti i nodi di **G** raggiungibili da **x** (per calcolarli eseguiamo una visita partendo da **x**)
- Gli antenati **A(x)**, ovvero tutti i nodi di G che raggiungono **x** (per calcolarli prima invertiamo la direzione di tutti gli archi in **G** e poi eseguiamo una visita partendo da **x**)

La *componente fortemente connessa* contenente **x** sarà data dai nodi che sono nell'intersezione tra **D(x)** e **A(x)**.

Per calcolare tutte le *componenti fortemente connesse* di **G** iteriamo il procedimento per tutti i nodi di **G**.

Riepilogo

- Concetto di grafo e terminologia
- Diverse strutture dati per rappresentare grafi nella memoria di un calcolatore
- L'utilizzo di una particolare rappresentazione può avere un impatto notevole sui tempi di esecuzione di un algoritmo su grafi (ad esempio, nella visita di un grafo)
- Algoritmo di visita generica e due casi particolari: visita in ampiezza e visita in profondità
- Componenti connesse e fortemente connesse di un grafo.