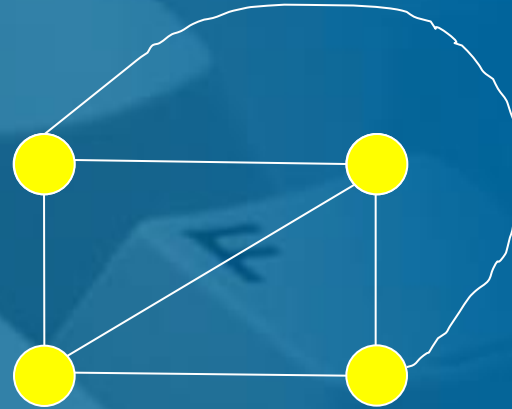
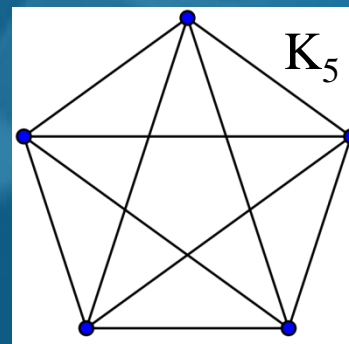
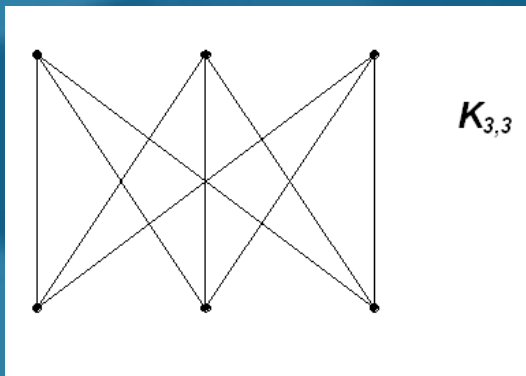


K_4 è planare?



Sì!

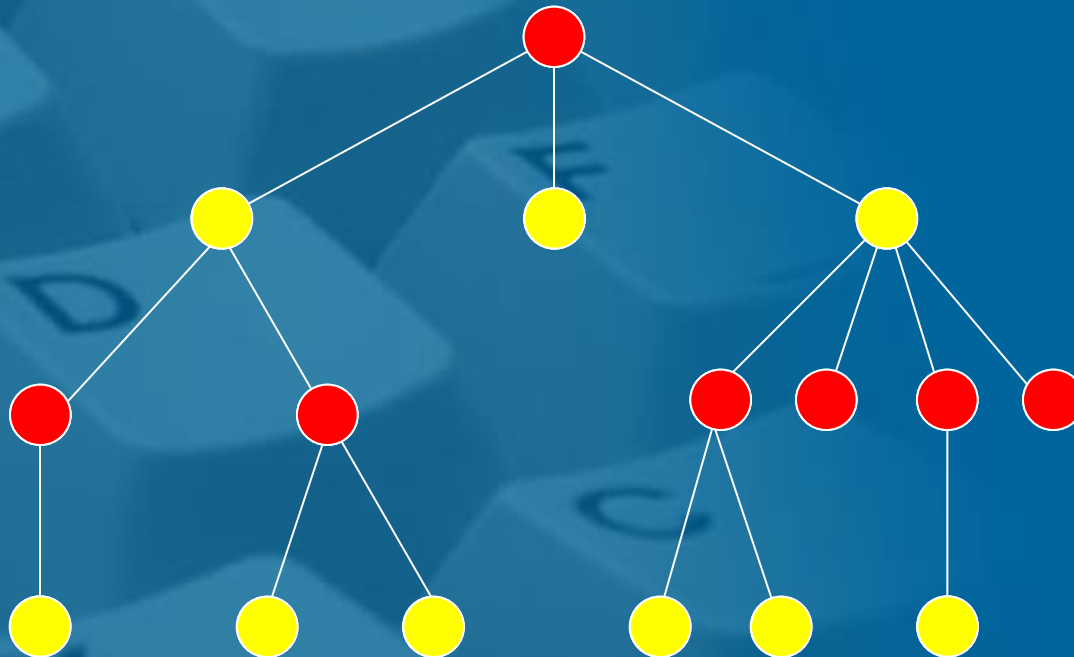
E $K_{3,3}$ e K_5 sono planari?



No! (Teorema di Kuratowski)

Un albero è un grafo bipartito?

SÌ!



Ma un grafo bipartito è sempre un albero??

Algoritmi e Strutture Dati

Capitolo 11 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**; ad esempio, un cammino euleriano è un modo particolare di visitare un grafo (euleriano!) in cui si passa una e una sola volta su ogni arco di G
- Problema di base in molte applicazioni
- Esistono vari tipi di visite con diverse proprietà: in particolare, vedremo la **visita in ampiezza** (BFS=breadth-first search) e la **visita in profondità** (DFS=depth-first search)

Componenti connesse di un grafo

- Ricordiamo che un grafo si dice **connesso** se per ogni coppia di vertici esiste un cammino che li congiunge
- Ogni grafo può essere partizionato in una o più componenti connesse **massimali**, ovvero sottografi connessi che non sono tra loro congiunti da alcun arco
- La visita di un grafo in realtà visita una dopo l'altra le componenti connesse del grafo stesso; per semplicità assumeremo nel seguito che il grafo sia connesso

Algoritmo di visita generica

- La visita parte da un vertice s qualsiasi, il quale viene aggiunto alla **frangia** F di visita, che contiene tutti i nodi da cui la visita può proseguire, ed esplora seguendo una qualche regola uno degli adiacenti di s
- Al generico passo di visita, viene scelto (secondo una qualche regola) un nodo u in F , e si visita (secondo una qualche regola) un arco (u,v) di G : se il vertice v viene scoperto per la prima volta mediante tale arco, esso viene **marcato come visitato**, e viene inserito in F ; inoltre, il nodo u viene marcato come padre di v , e l'arco (u,v) viene etichettato come **arco di visita**
- Un vertice rimane in F fintantoché non sono stati scoperti tutti i suoi adiacenti
- Se il grafo è **connesso** (come abbiamo ipotizzato), la visita genera un **albero di copertura** T del grafo, costituito da tutti i nodi del grafo e dagli **archi di visita**

Visite particolari

- Se la frangia F è implementata come una **coda** (cioè, quando un vertice si trova in testa alla coda, vengono visitati **tutti** i suoi adiacenti, si aggiungono alla coda i neo-visitati, e infine il vertice esce dalla frangia) si ha la visita in ampiezza (BFS)
- Se la frangia F è implementata come una **pila** (cioè, quando un vertice si trova in cima alla pila, viene visitato **un vertice** adiacente non ancora visitato, il quale viene aggiunto in cima alla pila e subito utilizzato per proseguire nella visita) si ha la visita in profondità (DFS)

Visita in ampiezza

Visita in ampiezza

algoritmo visitaBFS(*vertice* s) \rightarrow *albero*

1. rendi tutti i vertici non marcati
2. $T \leftarrow$ albero formato da un solo nodo s
3. Coda F
4. marca il vertice s
5. $F.enqueue(s)$
6. **while** (**not** $F.isEmpty()$) **do**
7. $u \leftarrow F.dequeue()$
8. **for each** (arco (u, v) in G) **do**
9. **if** (v non è ancora marcato) **then**
10. $F.enqueue(v)$
11. marca il vertice v
12. rendi u padre di v in T
13. **return** T

Costo della visita in ampiezza

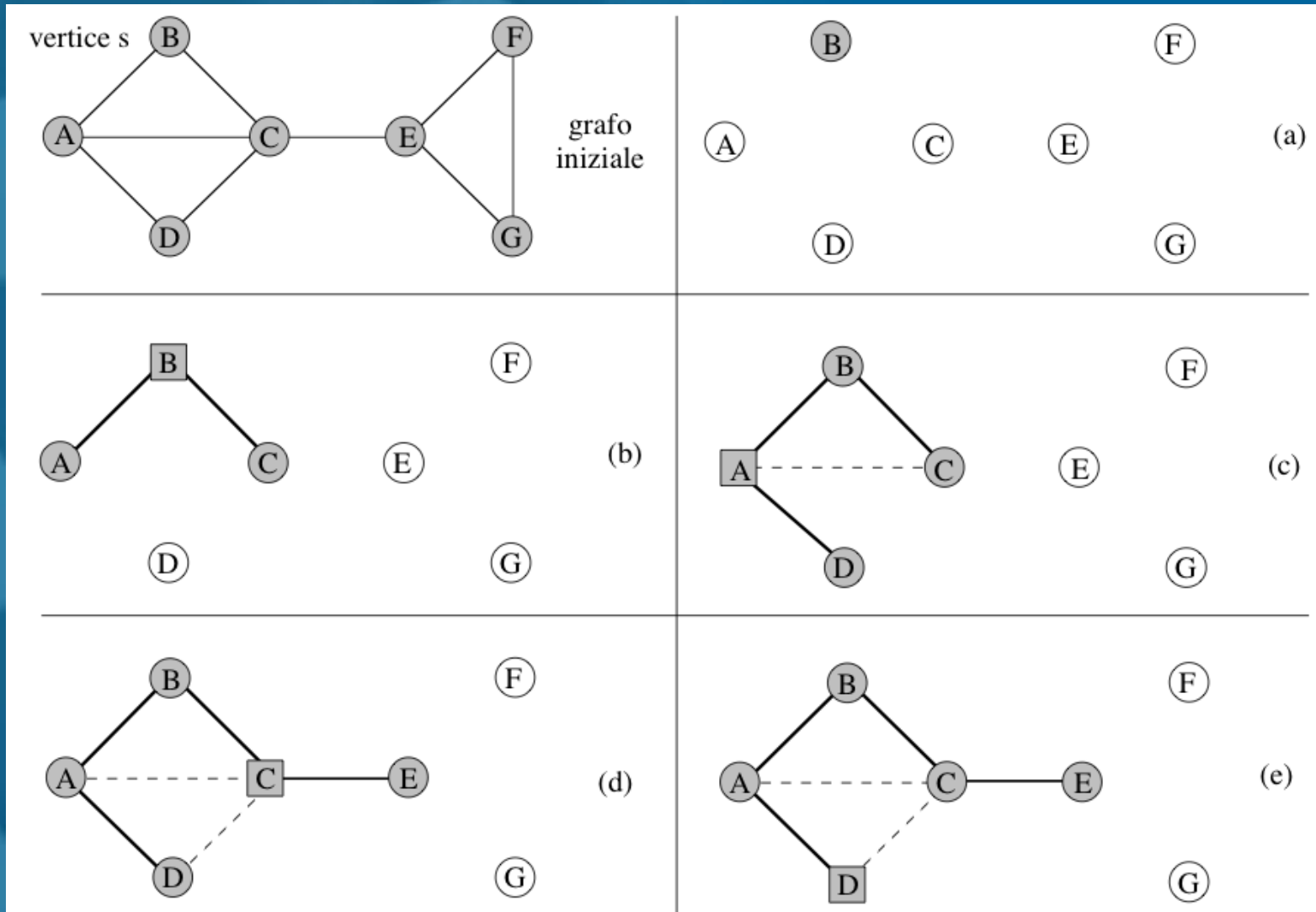
Il tempo di esecuzione dipende dalla struttura dati usata per rappresentare il grafo:

- Liste di adiacenza: $\Theta(m+n)$
- Matrice di adiacenza: $\Theta(n^2)$

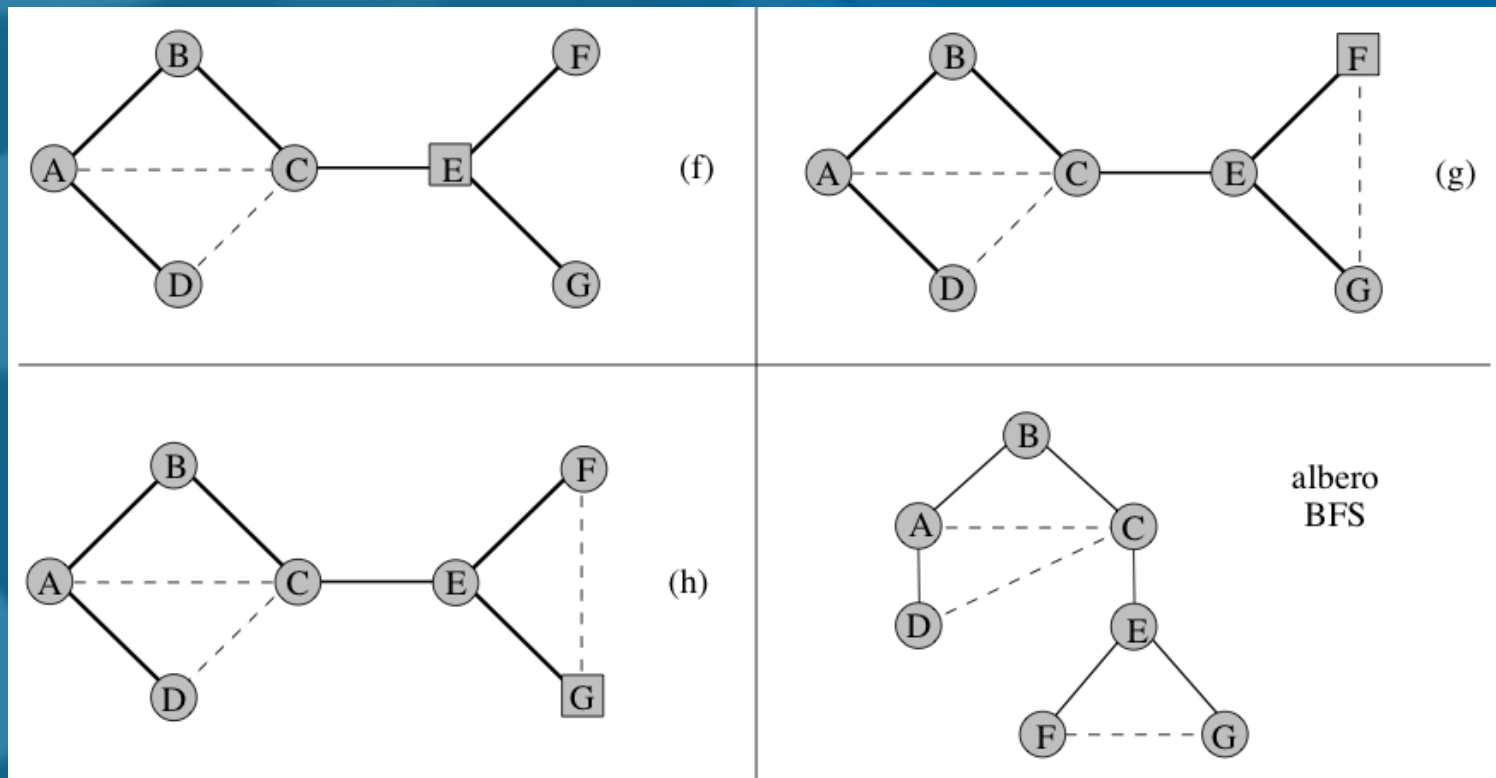
Osservazioni:

1. Si noti che se il grafo è connesso (come da noi ipotizzato) allora $m \geq n-1$ e quindi $\Theta(m+n) = \Theta(m)$
2. Ricordando che $m \leq n(n-1)/2$, si ha quindi $\Theta(m+n) = O(n^2)$
 \Rightarrow per $m = o(n^2)$ (ovverosia per tutti i grafi **non densi**) la rappresentazione mediante **liste di adiacenza** mi consente di ottenere una BFS **temporalmente più efficiente!**

Esempio: grafo non orientato (1/2)

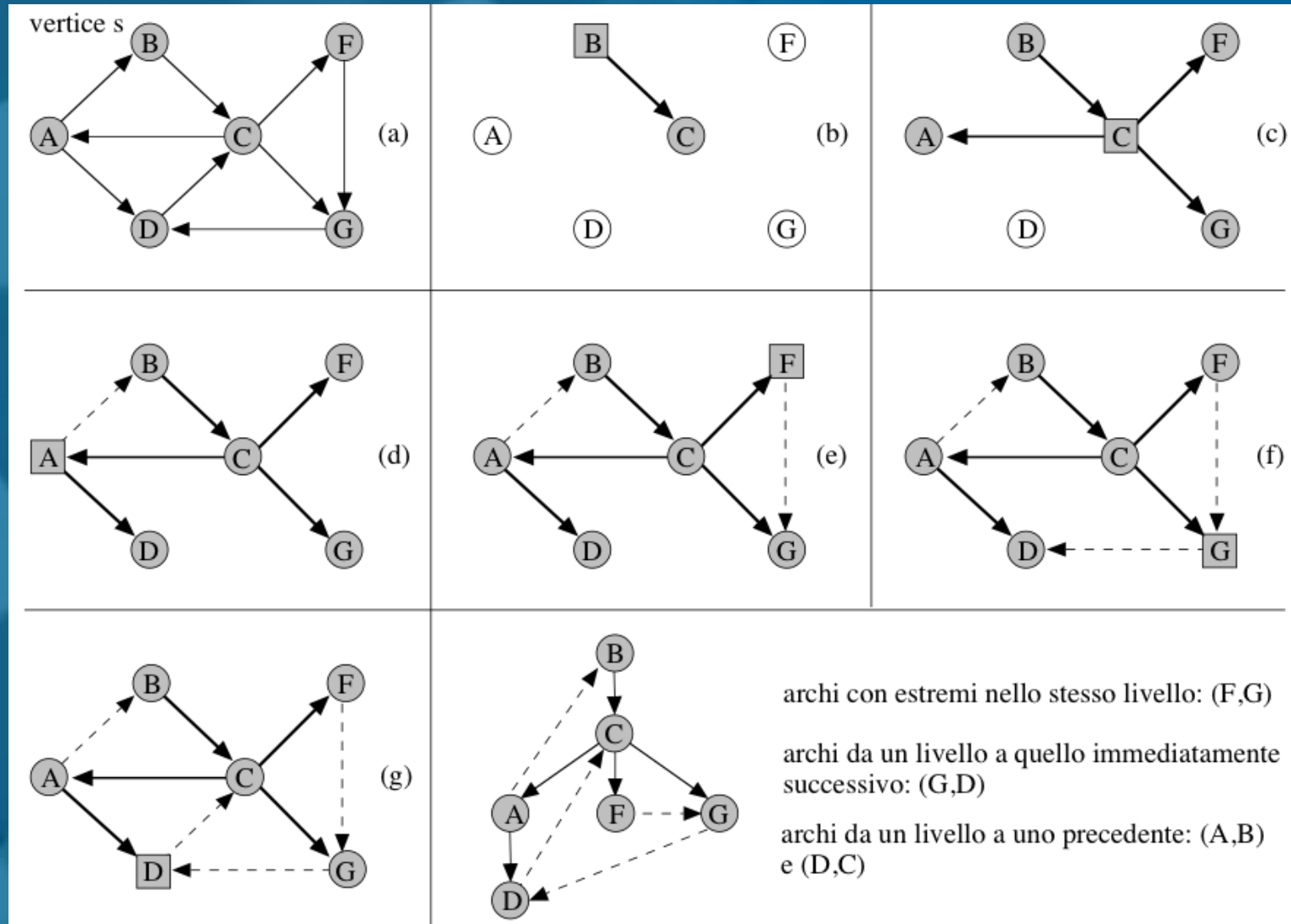


Esempio: grafo non orientato (2/2)



Si noti che nell'albero BFS, i nodi di un dato livello (si ricordi che la radice è a livello 0) sono stati visitati **dopo** tutti i nodi di livello **più piccolo** e **prima** di tutti i nodi di livello **più grande**. L'ordine con cui sono stati visitati i nodi di uno stesso livello dipende dall'ordine in cui li abbiamo scoperti. Nella figura, abbiamo disegnato il BFS in modo che l'ordine di visita di nodi di uno stesso livello va da sinistra a destra

Esempio: grafo orientato



Proprietà dell'albero BFS radicato in s

- Se il grafo è **non orientato**, per ogni arco (u,v) di G si ha:
 - o l'arco (u,v) è un arco di T , e quindi u e v appartengono a livelli consecutivi di T , oppure
 - l'arco (u,v) non è un arco di T ; in tal caso, o i nodi u e v appartengono allo stesso livello di T , oppure a livelli consecutivi (*ne vedete il motivo?*)
- Se il grafo è **orientato**, per ogni arco (u,v) si ha:
 - o l'arco (u,v) è un arco di T , e quindi u è il padre di v nell'albero, oppure
 - l'arco (u,v) non è un arco di T ; in tal caso, o i nodi u e v appartengono allo stesso livello di T , oppure u è a livello immediatamente superiore di v , oppure infine u è su qualche livello arbitrariamente più in basso di quello di v (*ne vedete il motivo?*)
- Dalle osservazioni precedenti, per ogni nodo v , il livello di v nell'albero BFS T è pari alla **distanza** di v dalla sorgente s (sia per grafi orientati che non orientati): infatti, non esistono archi non appartenenti a T che consentono di prendere “scorciatoie”

Visita in profondità

Visita in profondità

procedura visitaDFSRicorsiva(*vertice* v , *albero* T)

1. *marca e visita il vertice* v
2. **for each** (arco (v, w)) **do**
3. **if** (w non è marcato) **then**
4. aggiungi l'arco (v, w) all'albero T
5. visitaDFSRicorsiva(w, T)

algoritmo visitaDFS(*vertice* s) \rightarrow *albero*

6. $T \leftarrow$ albero vuoto
7. visitaDFSRicorsiva(s, T)
8. **return** T

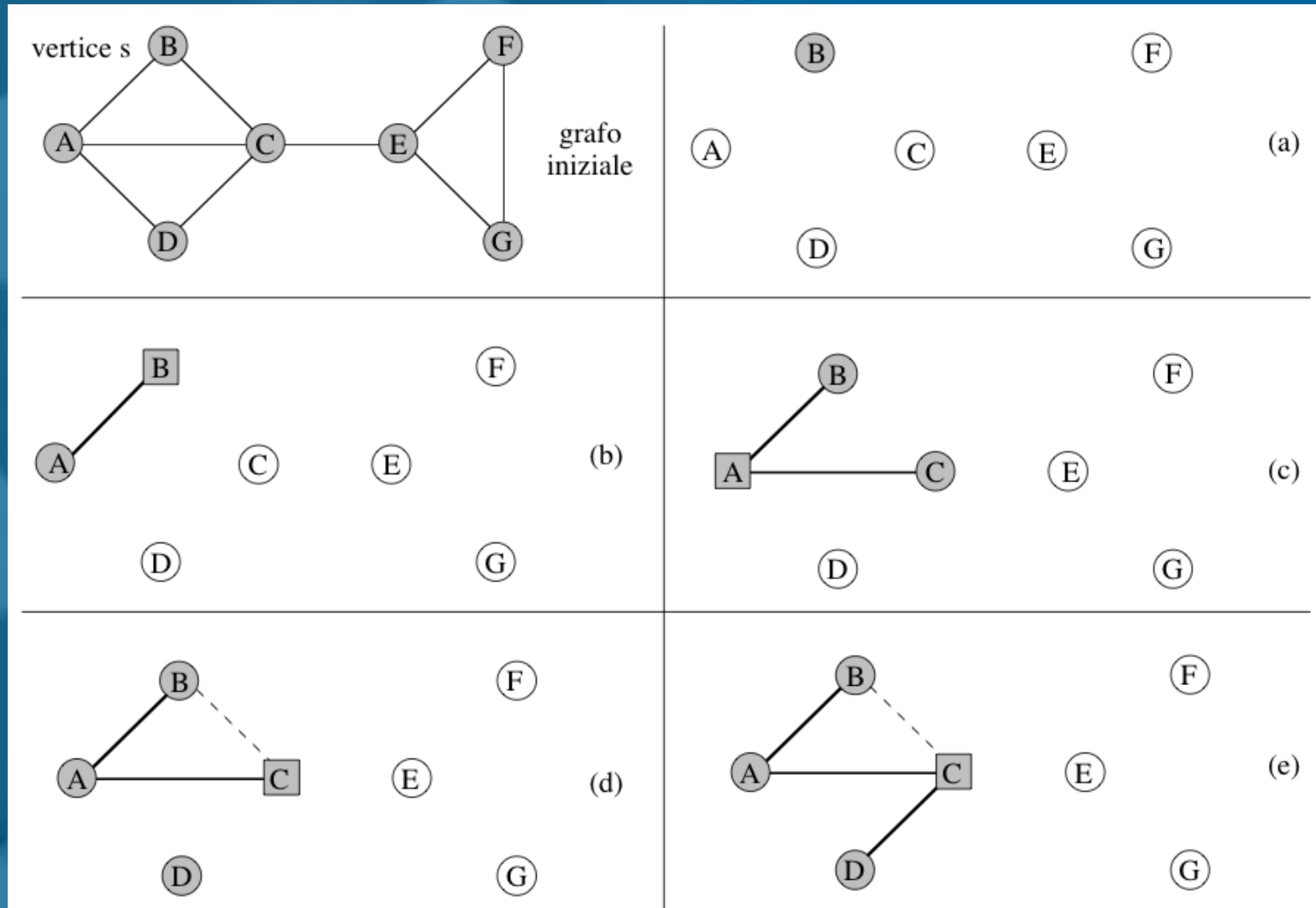
Costo della visita in profondità

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

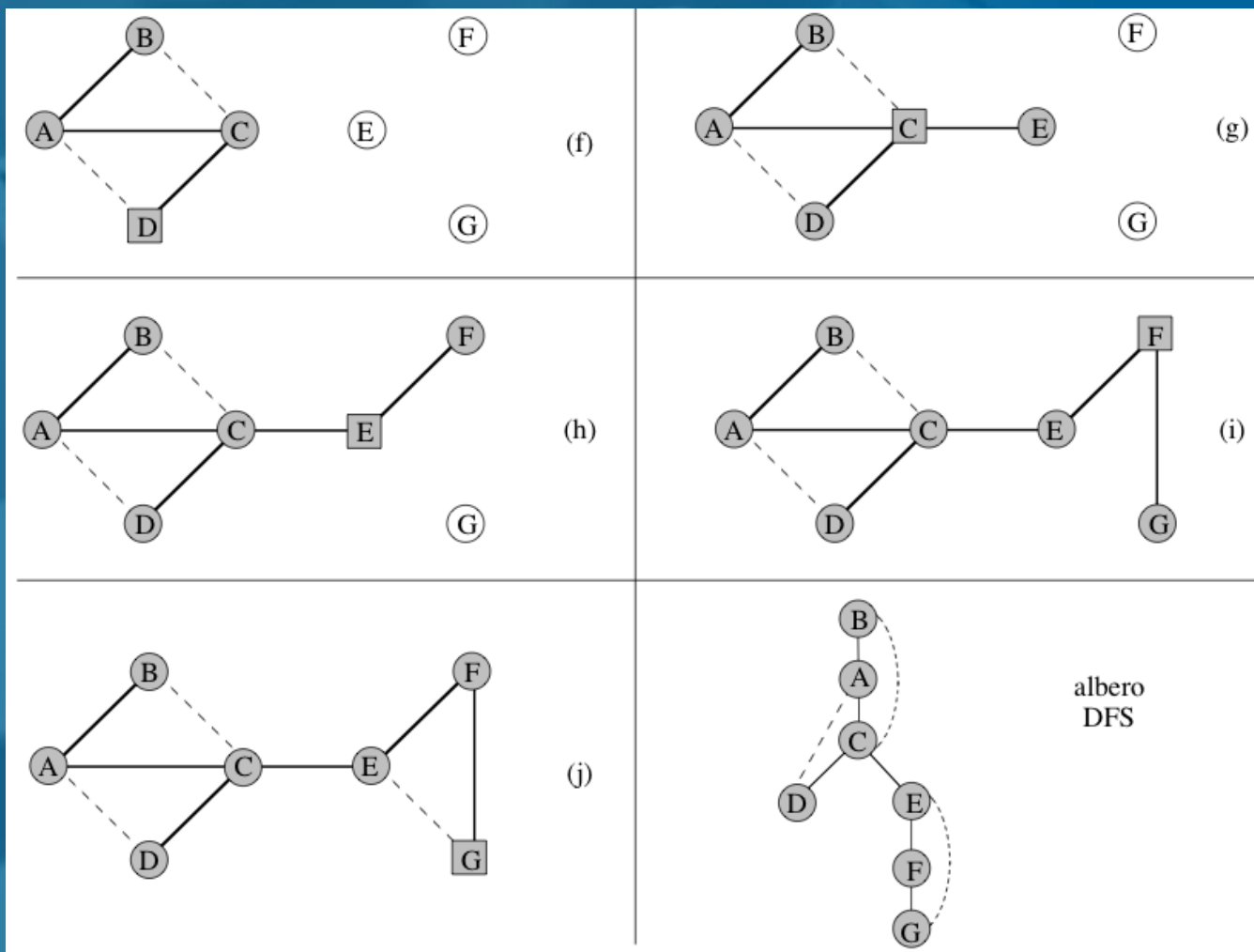
- Liste di adiacenza: $\Theta(m+n)$
- Matrice di adiacenza: $\Theta(n^2)$

⇒ Anche in questo caso, per $m=O(n^2)$ (ovverosia per tutti i grafi **non densi**) la rappresentazione mediante **liste di adiacenza** mi consente di ottenere una DFS **temporalmente più efficiente!**

Esempio: grafo non orientato (1/2)



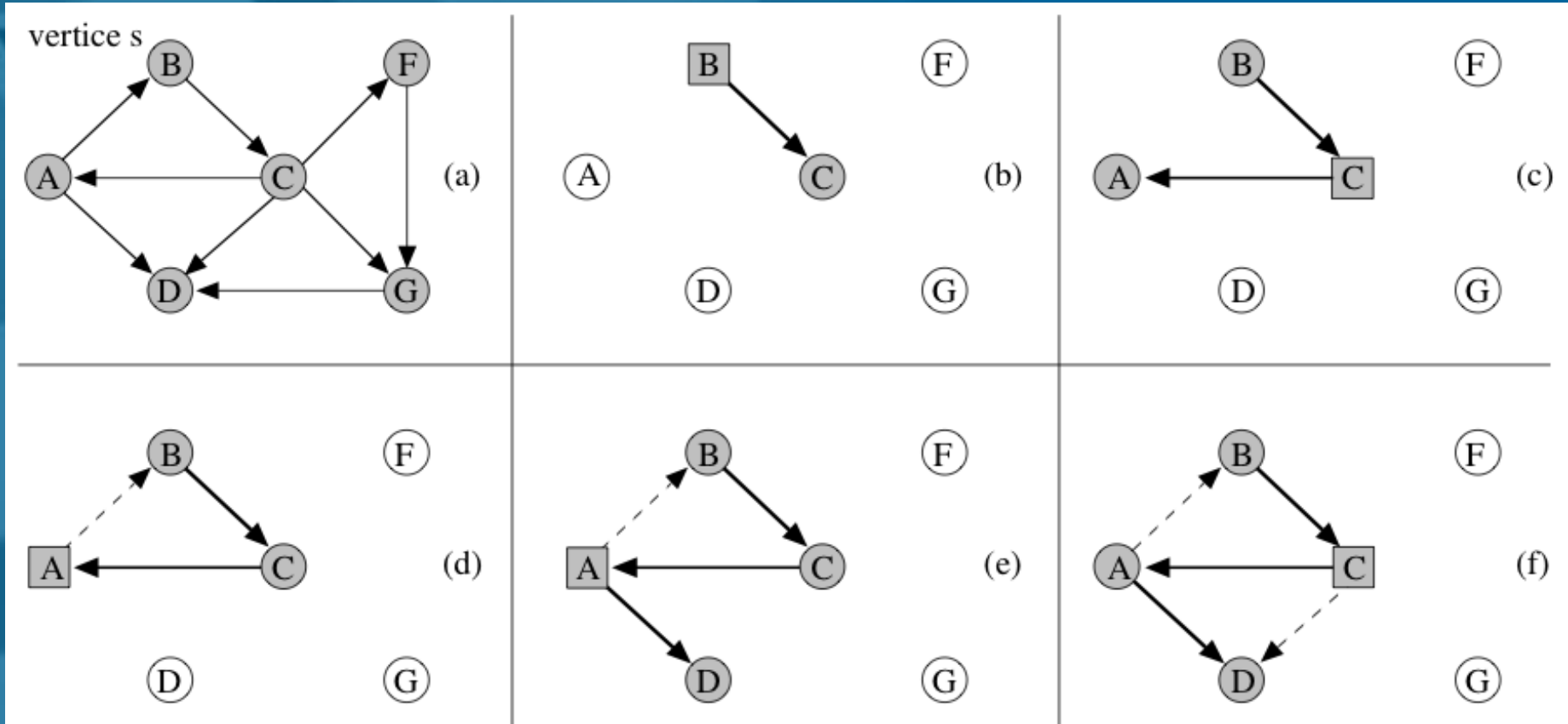
Esempio: grafo non orientato (2/2)



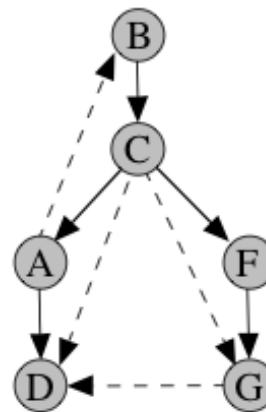
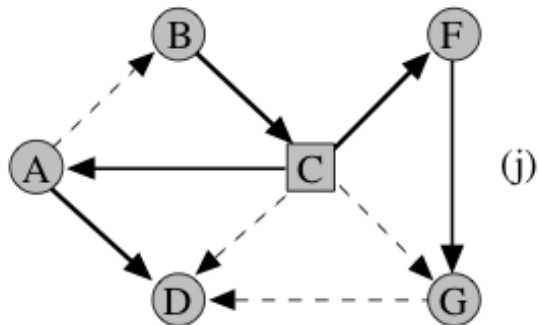
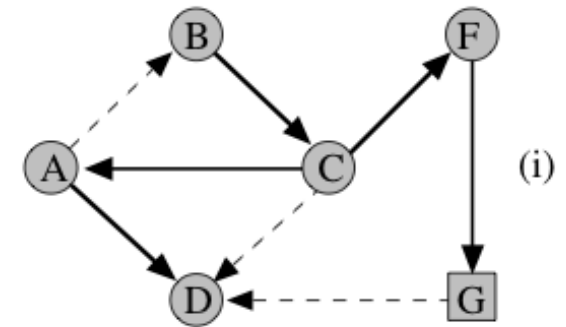
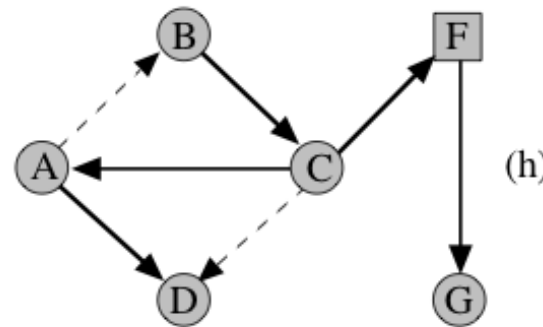
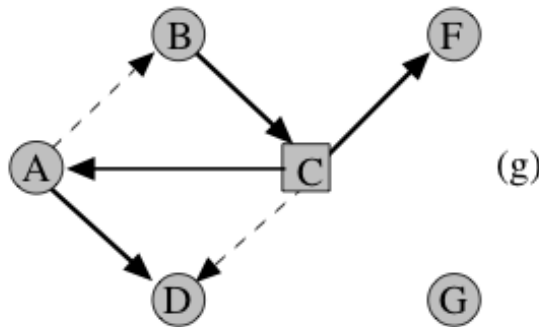
Si noti che nell'albero DFS, un nodo viene visitato sempre **dopo** tutti i suoi **antenati**, e **prima** di tutti i suoi **discendenti**.

Nella figura, abbiamo disegnato il DFS in modo che l'ordine di visita di nodi di uno stesso livello va da sinistra a destra

Esempio: grafo orientato (1/2)



Esempio: grafo orientato (2/2)



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

archi all'indietro: (A,B)

archi trasversali a sinistra: (G,D)

Proprietà dell'albero DFS radicato in s

- Se il grafo è **non orientato**, per ogni arco (u,v) si ha:
 - o l'arco (u,v) è un arco dell'albero DFS T , oppure
 - l'arco (u,v) non è un arco di T ; in tal caso, i nodi u e v sono l'uno discendente/antenato dell'altro in T (ne vedete il motivo?)
- Se il grafo è **orientato**, per ogni arco (u,v) si ha:
 - o l'arco (u,v) è un arco di T , e quindi u è il padre di v in T , oppure
 - l'arco (u,v) non è un arco di T ; in tal caso, i nodi u e v sono l'uno discendente/antenato dell'altro in T , oppure (u,v) è un arco **trasversale a sinistra**, vale a dire il vertice v è in un sottoalbero visitato precedentemente rispetto ad u (ne vedete il motivo?)

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

Approfondimento

Dato un grafo $G=(V,E)$ rappresentato tramite una matrice di adiacenza A , scrivere un algoritmo che scorrendo **una sola volta** A :

1. Verifica se G è **completo**;
2. Trova il **grado** di G ;
3. Costruisce la rappresentazione di G tramite **liste di adiacenza**.

MATRICE - LISTE (A)

COMPLETO \leftarrow TRUE

GRADO-MAX \leftarrow 0

FOR $i \leftarrow 1$ TO n

DO { GRADO- i -ESIMO \leftarrow 0

Adj[i] \leftarrow nil

FOR $j \leftarrow 1$ TO n

DO IF $A[i, j] = 1$

THEN { NEW(p)

GRADO- i -ESIMO \leftarrow GRADO- i -ESIMO + 1

VERTICE[p] \leftarrow j

NEXT[p] \leftarrow Adj[i]

Adj[i] \leftarrow p

ELSE IF $i \neq j$

THEN COMPLETO \leftarrow FALSE

IF GRADO- i -ESIMO > GRADO-MAX

THEN GRADO-MAX \leftarrow GRADO- i -ESIMO

IF COMPLETO = TRUE

THEN STAMPA "G E' COMPLETO"

ELSE STAMPA "G NON E' COMPLETO"

RETURN GRADO-MAX

COMPLESSITA': $\Theta(n^2)$