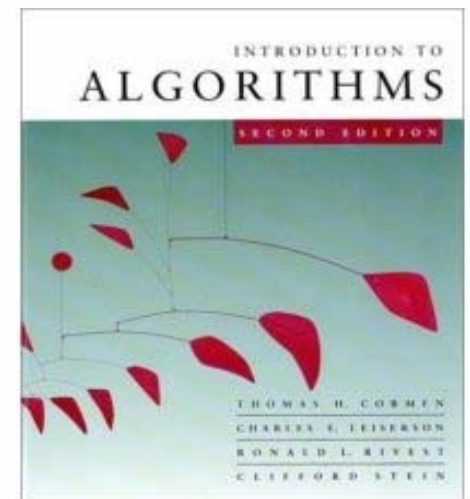


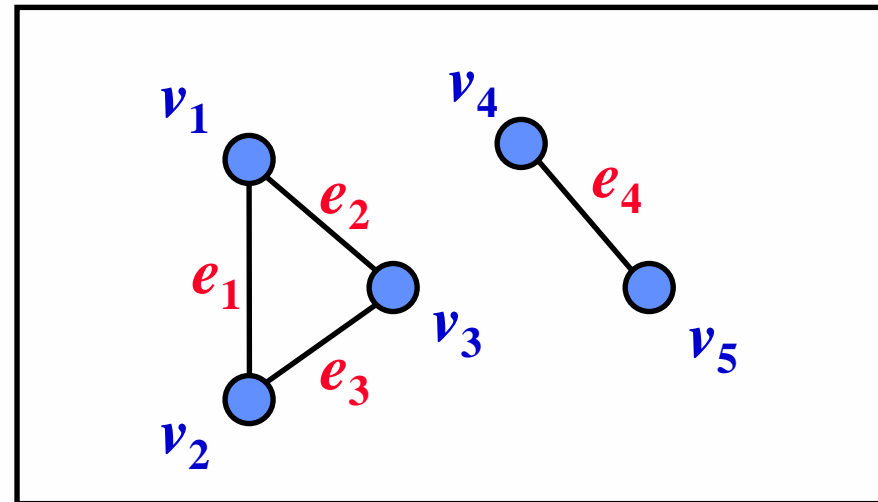
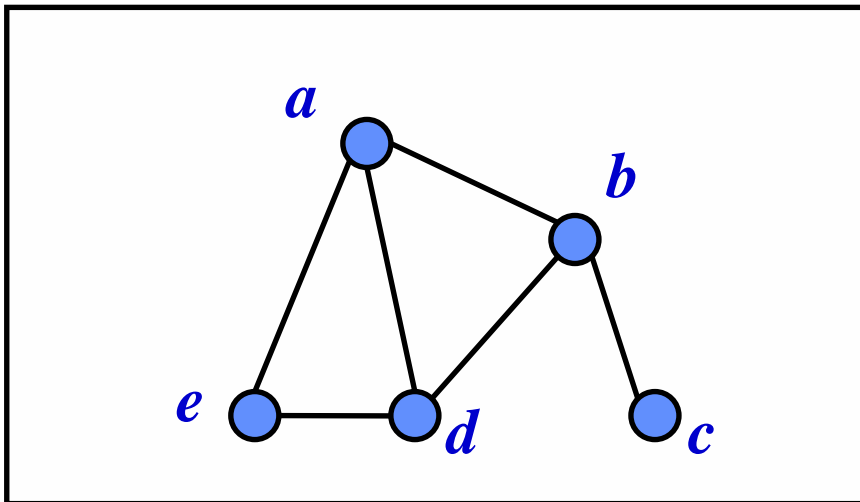
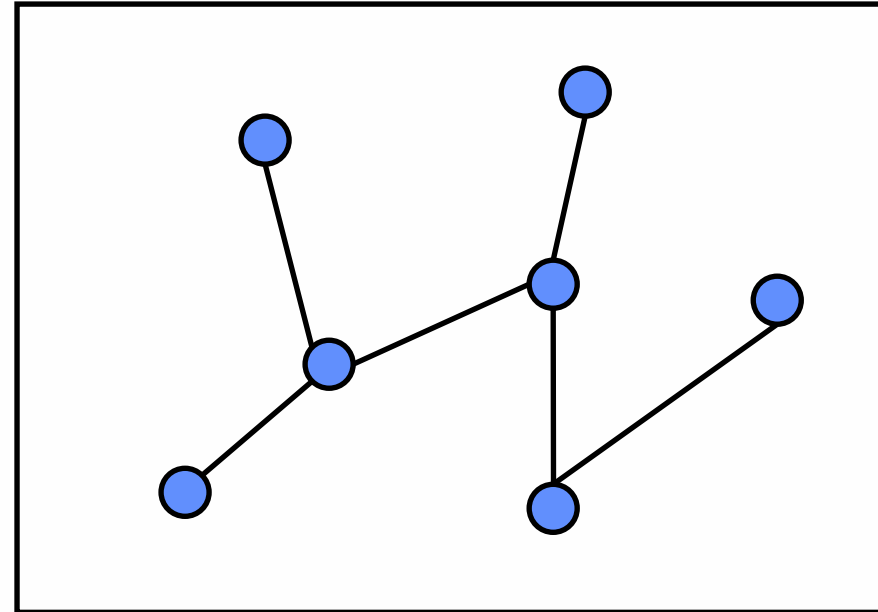
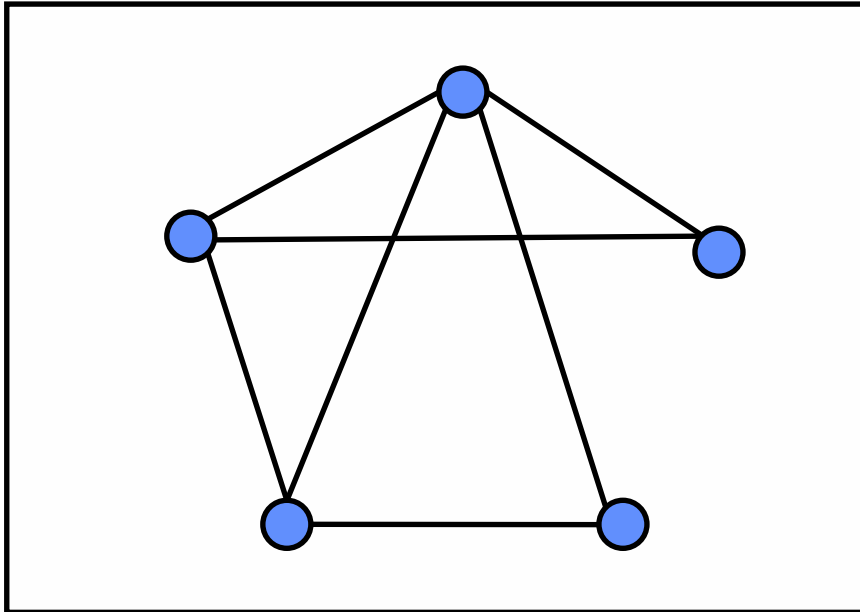


Algoritmi e Strutture Dati

Grafi

- ❑ Questo materiale è tratto dalle trasparenze del corso "Algoritmi e Strutture Dati" del prof. Alberto Montresor dell'Università di Trento. (<http://www.dit.unitn.it/~montreso/asd/index.shtml>)
- ❑ T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein Introduction to Algorithms, Second Edition
- ❑ Queste trasparenze sono disponibili su <http://dei.polimi.it/upload/loiacono>
- ❑ Materiale rilasciato con licenza Creative Commons Attribution-NonCommercial-ShareAlike License (<http://creativecommons.org/licenses/by-nc-sa/2.5/>)

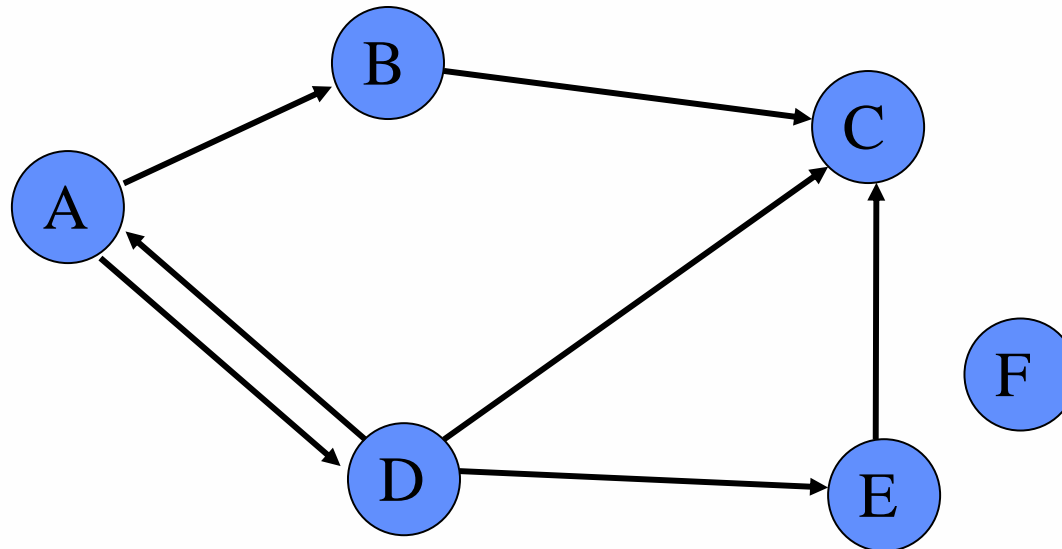




- ❑ Visite
 - ▶ Visite in ampiezza (cammini minimi singola sorgente)
 - ▶ Visite in profondità (ordinamento topologico, componenti fortemente connesse)
- ❑ Cammini minimi
 - ▶ Da singola sorgente
 - ▶ Fra tutte le coppie di vertici
- ❑ Alberi di connessione minimi
- ❑ Problemi di flusso
- ❑

Definizioni

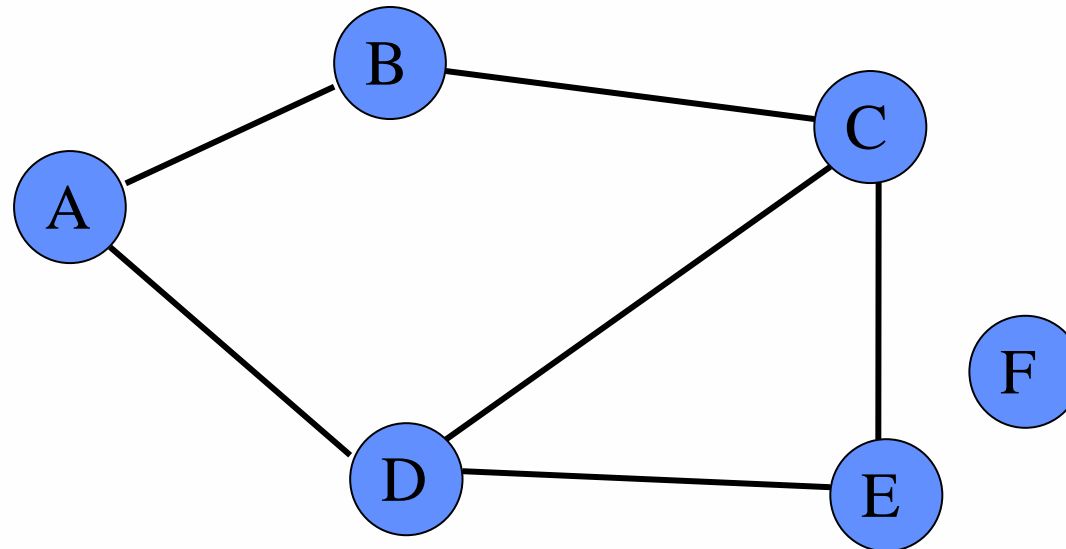
- Un grafo orientato G è definito dalla coppia (V, E) dove:
 - ▶ V è l'insieme finito dei vertici
 - ▶ E è l'insieme degli archi (relazione binaria in V)



$$V = \{A, B, C, D, E, F\}$$

$$E = \{ (A,B), (A,D), (B,C), (D,C), (E,C), (D,E), (D,A) \}$$

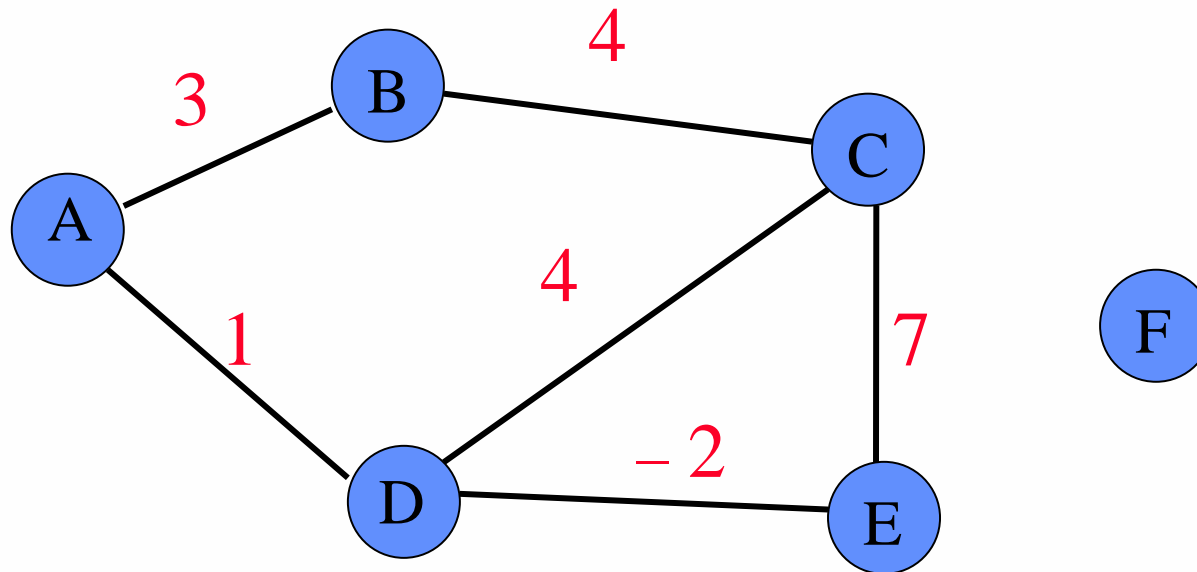
- Un grafo non orientato G è definito dalla coppia (V, E) dove:
 - ▶ V è l'insieme finito dei vertici
 - ▶ E è l'insieme degli archi (coppie ordinate di elementi di V)



$$V = \{A, B, C, D, E, F\}$$

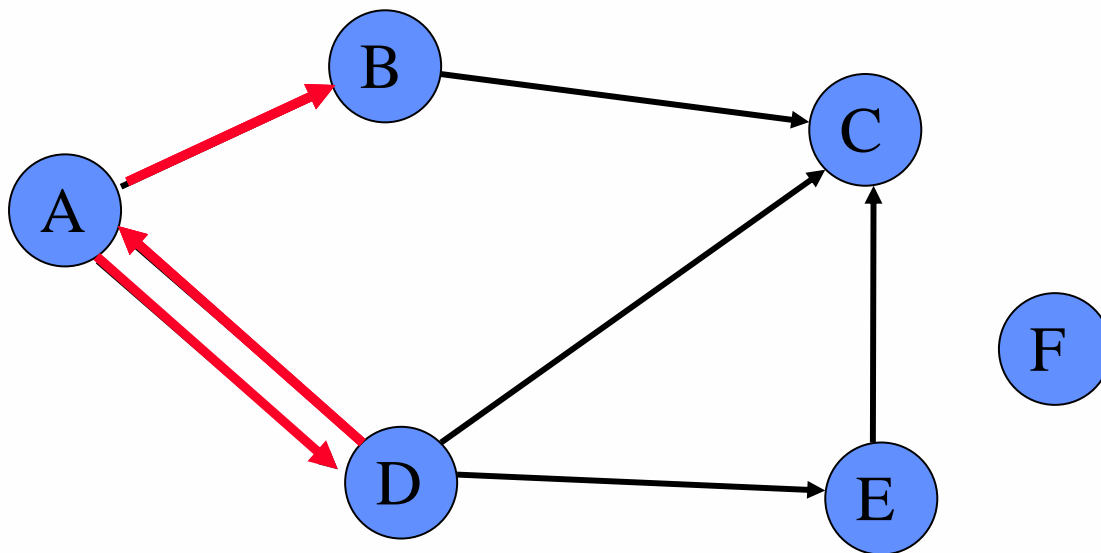
$$E = \{ (A,B), (A,D), (B,C), (C,D), (C,E), (D,E) \}$$

- ❑ In alcuni casi ogni arco ha un *peso* (o *costo*) associato
- ❑ Il costo può essere determinato tramite una funzione di costo $c: E \rightarrow R$, dove R è l'insieme dei numeri reali
- ❑ Quando tra due vertici non esiste un arco, si dice che il costo è infinito



Esempio: $c(C,F) = \infty$

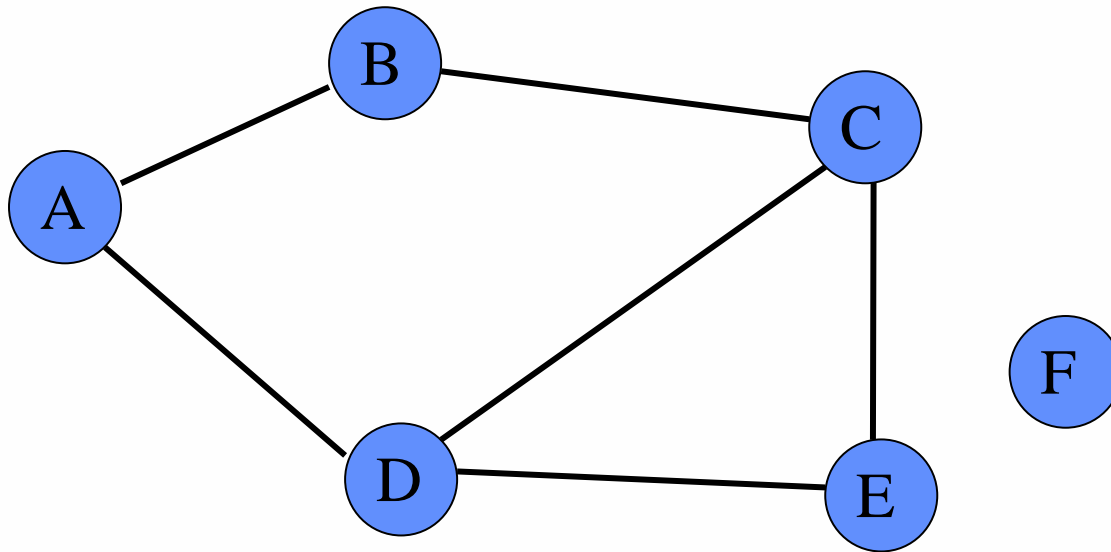
- ❑ In un grafo orientato, un arco (v,w) si dice **incidente** da v in w
- ❑ Un vertice w si dice **adiacente** a v se e solo se $(v, w) \in E$
- ❑ In un grafo non orientato la relazione di adiacenza tra vertici è simmetrica



(A,B) è incidente da A a B
 (A,D) è incidente da A a D
 (D,A) è incidente da D a A

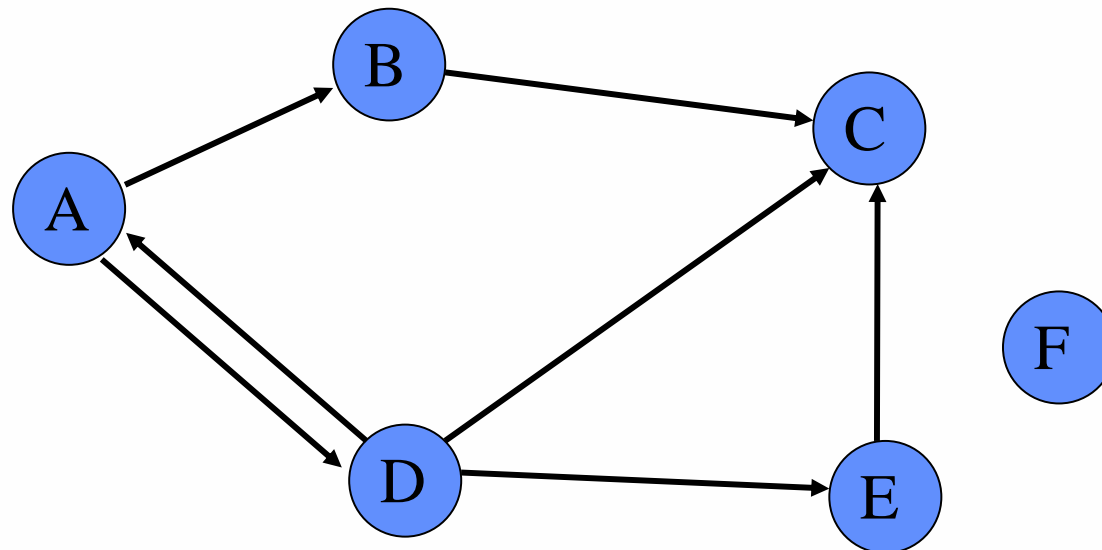
B è adiacente ad A
C è adiacente a B, D, E
A è adiacente a D e viceversa
B non è adiacente a D, C
F non è adiacente ad alcun vertice

- In un grafo **non** orientato, il **grado** di un vertice è il numero di archi che da esso si dipartono



A, B ed E hanno grado 2
C e D hanno grado 3
F ha grado 0

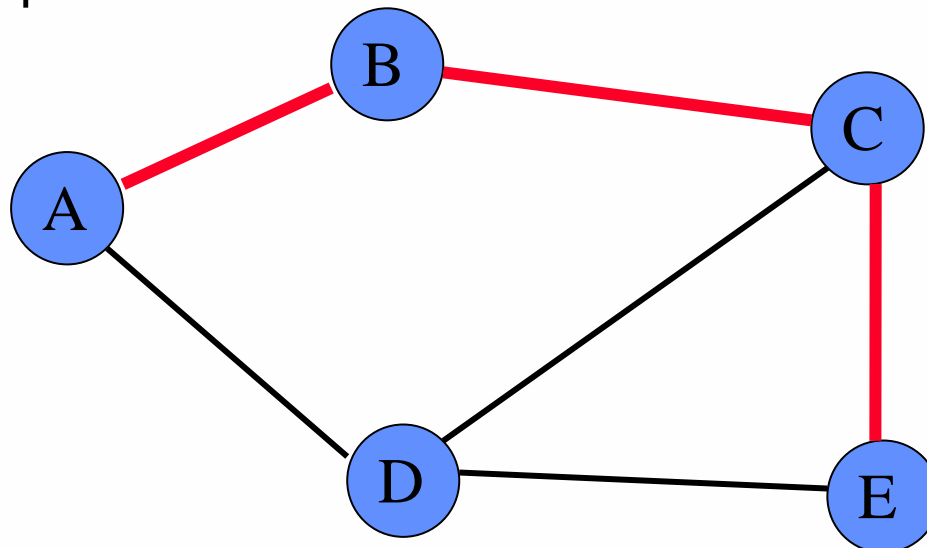
- ❑ In un **grafo orientato**
 - ▶ il **grado entrante** di un vertice è il numero di archi incidenti in esso
 - ▶ il **grado uscente** di un vertice è il numero di archi uscenti da esso
- ❑ In un **grafo orientato** il **grado** di un vertice è la somma del suo grado entrante e del suo grado uscente



A ha *g. u.* 2 e *g. e.* 1
B ha *g. u.* 1 e *g. e.* 1
C ha *g. u.* 0 e *g. e.* 3
D ha *g. u.* 3 e *g. e.* 1

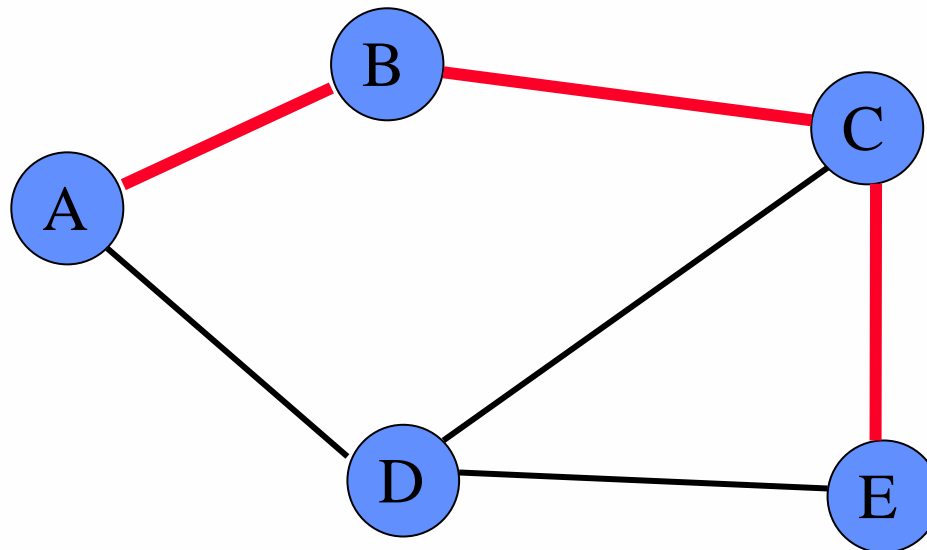
A e *C* hanno *grado* 3
B ha *grado* 2
D ha *grado* 4

- ❑ Un **cammino** nel grafo $G=(V,E)$ è una sequenza di vertici $\langle w_0, w_1, \dots, w_n \rangle$ tale che $(w_i, w_{i+1}) \in E$ per $0 \leq i \leq n-1$
- ❑ Il cammino $\langle w_0, w_1, w_2, \dots, w_n \rangle$ contiene i nodi w_0, w_1, \dots, w_n e gli archi $(w_0, w_1) (w_1, w_2) \dots (w_{n-1}, w_n)$
- ❑ Lunghezza del cammino: # di archi che connettono i vertici nell'ordine della sequenza
- ❑ Esempio:



$\langle A, B, C, E \rangle$ è un cammino nel grafo con lunghezza 3

- ❑ Un cammino si dice **semplice** se tutti i suoi vertici sono distinti (compaiono una sola volta nella sequenza)
- ❑ Esempio:

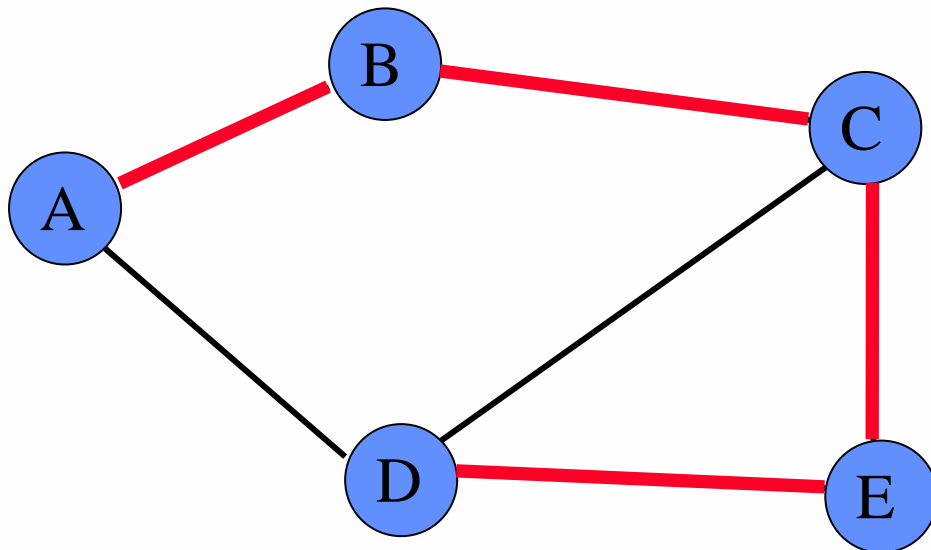


$\langle A, B, C, E \rangle$
è semplice ...

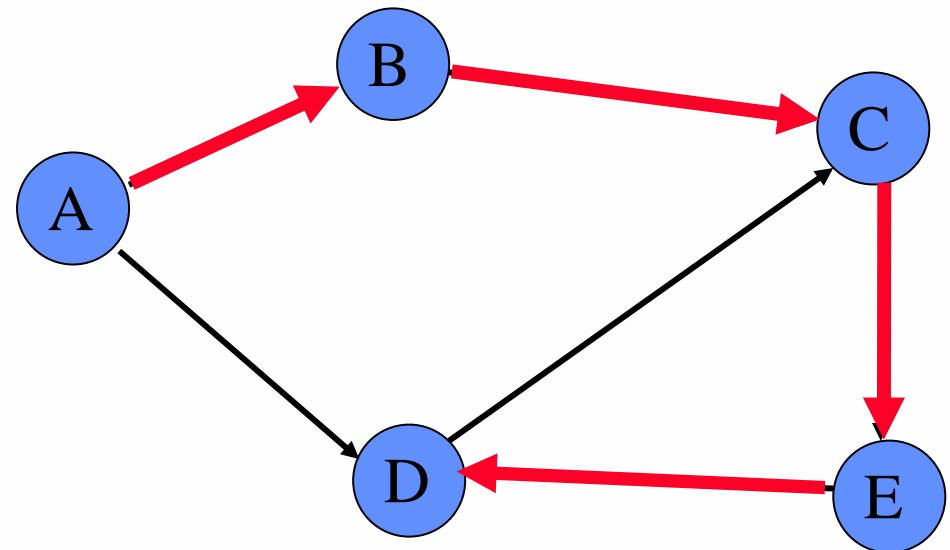
... ma il cammino
 $\langle A, B, C, E, D, C \rangle$
non è semplice,
poiché C è ripetuto

- Se esiste un cammino c tra i vertici v e w , si dice che w è raggiungibile da v tramite c

Esempio: A è raggiungibile da D e viceversa

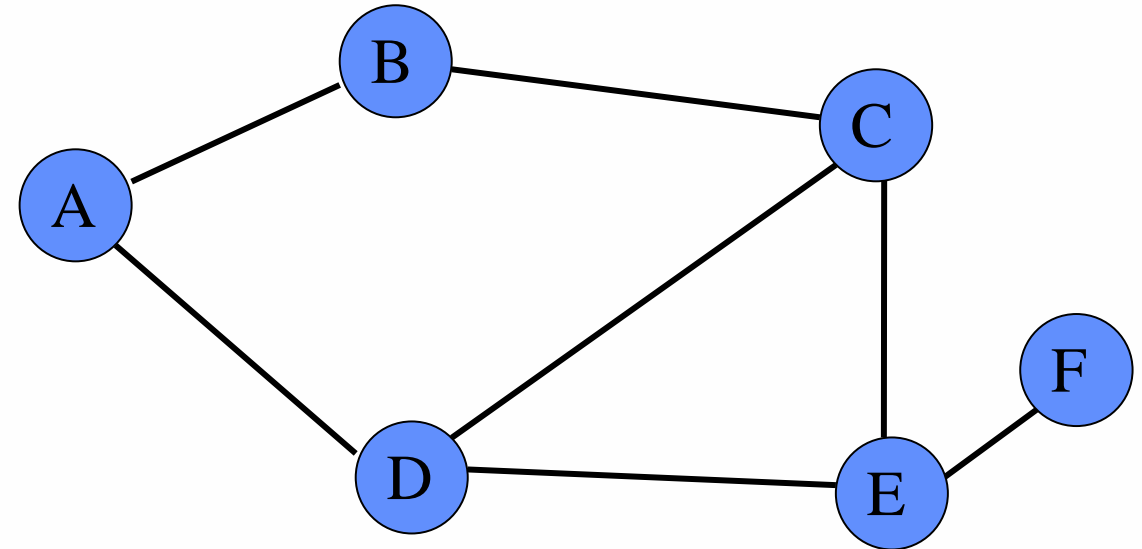
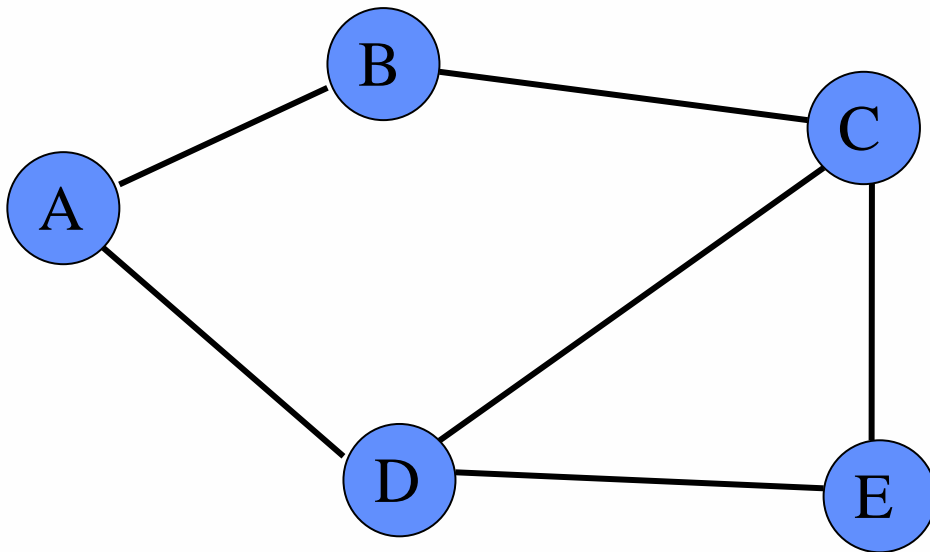


Es.: D è raggiungibile da A ma non viceversa

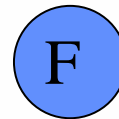


- Se G è un **grafo non orientato**, diciamo che G è **connesso** se esiste un cammino da ogni vertice ad ogni altro vertice.

Questo grafo non orientato
è **connesso**.

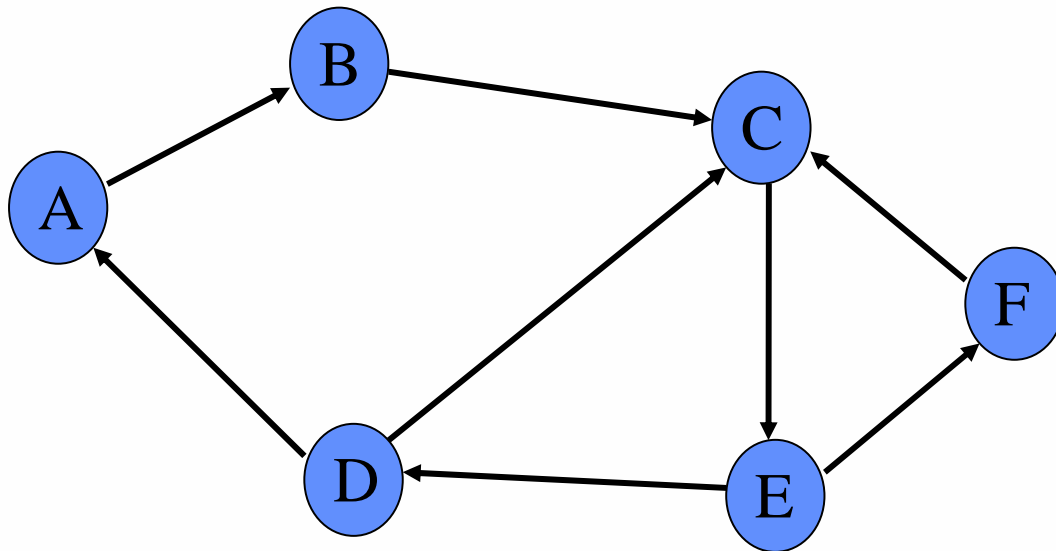


Questo grafo non orientato
non è connesso.

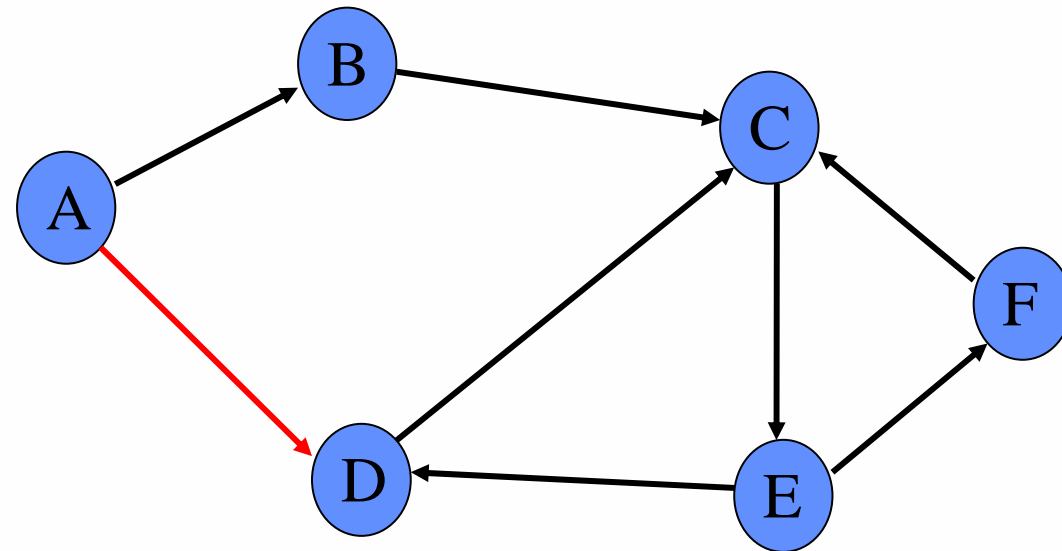


- Se G è un **grafo orientato**, diciamo che G è **fortemente connesso** se esiste un cammino da ogni vertice ad ogni altro vertice.

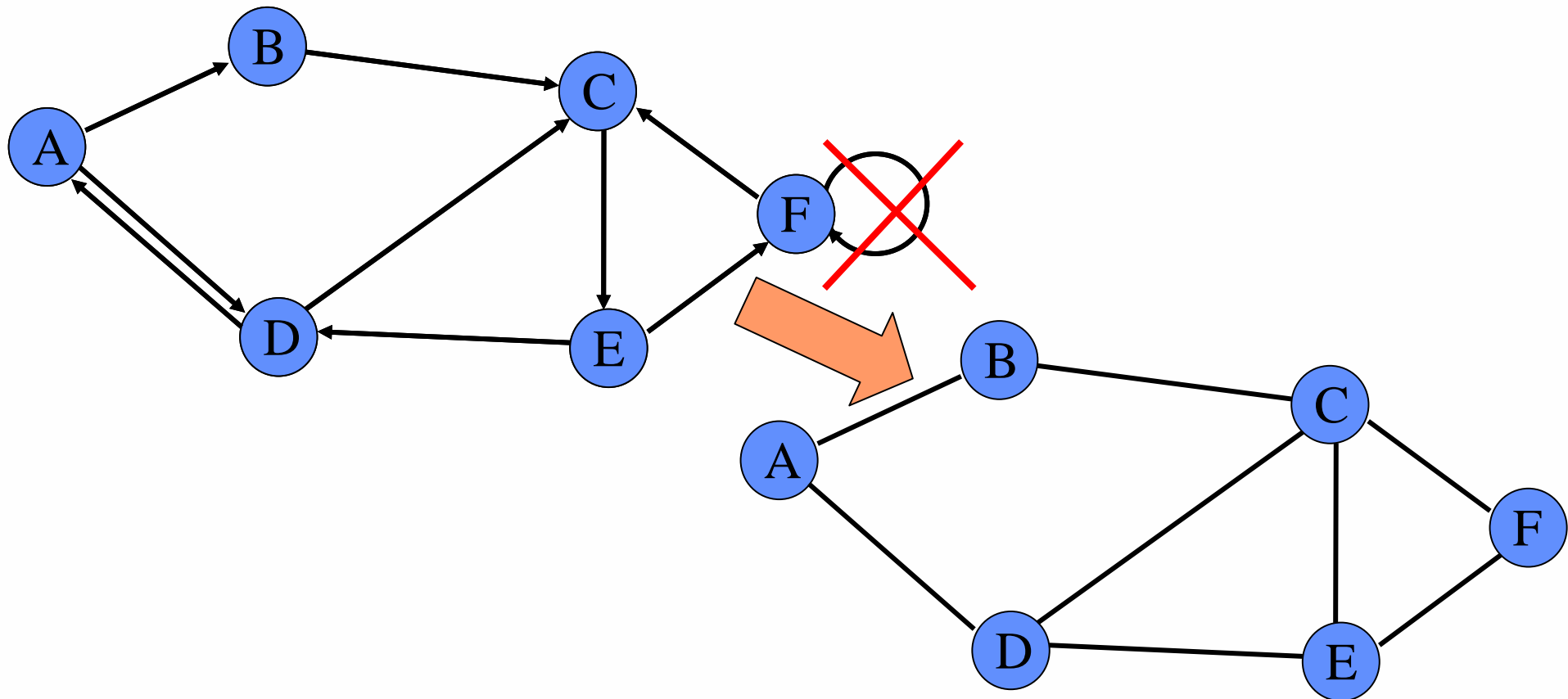
Questo grafo orientato è **fortemente connesso**.



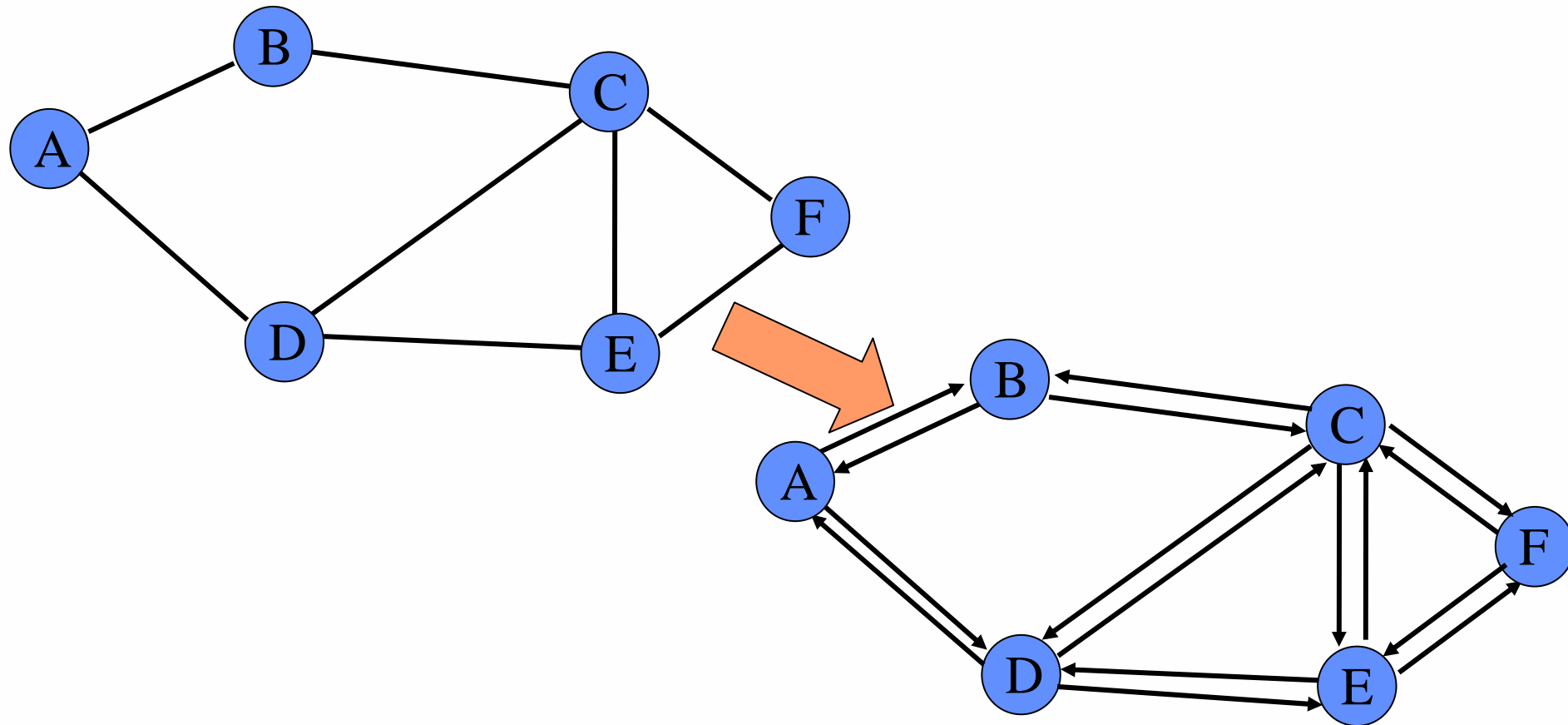
Questo grafo orientato **non è fortemente connesso**; ad es., non esiste cammino da **D** a **A**.



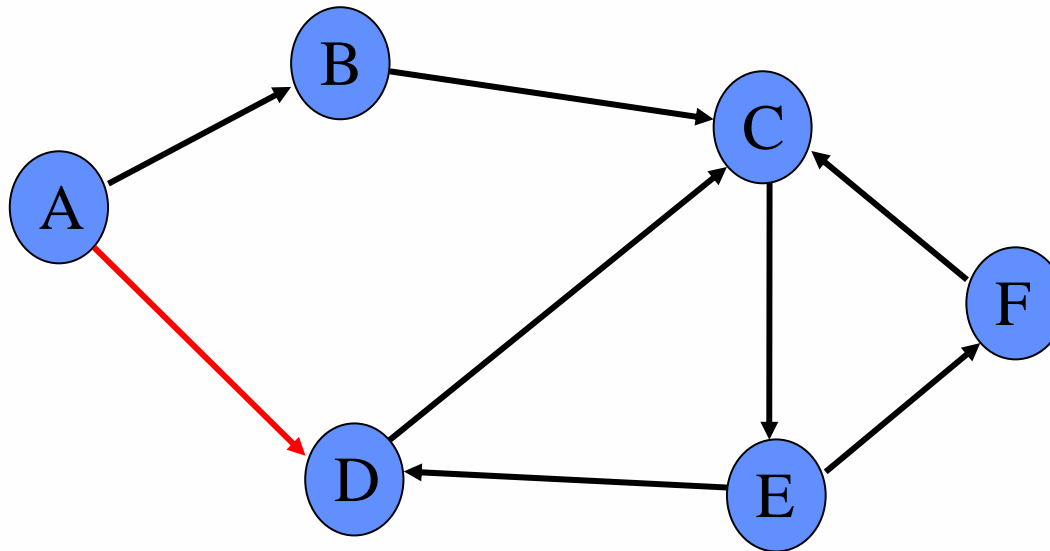
- Se G è un **grafo orientato**, il grafo ottenuto ignorando la direzione degli archi e gli autoanelli è detto il **grafo non orientato sottostante** o anche **versione non orientata** di G .



- Se G è un **grafo non orientato**, il grafo ottenuto inserendo due archi orientati opposti per ogni arco non orientato del grafo è detto la **versione orientata** di G .

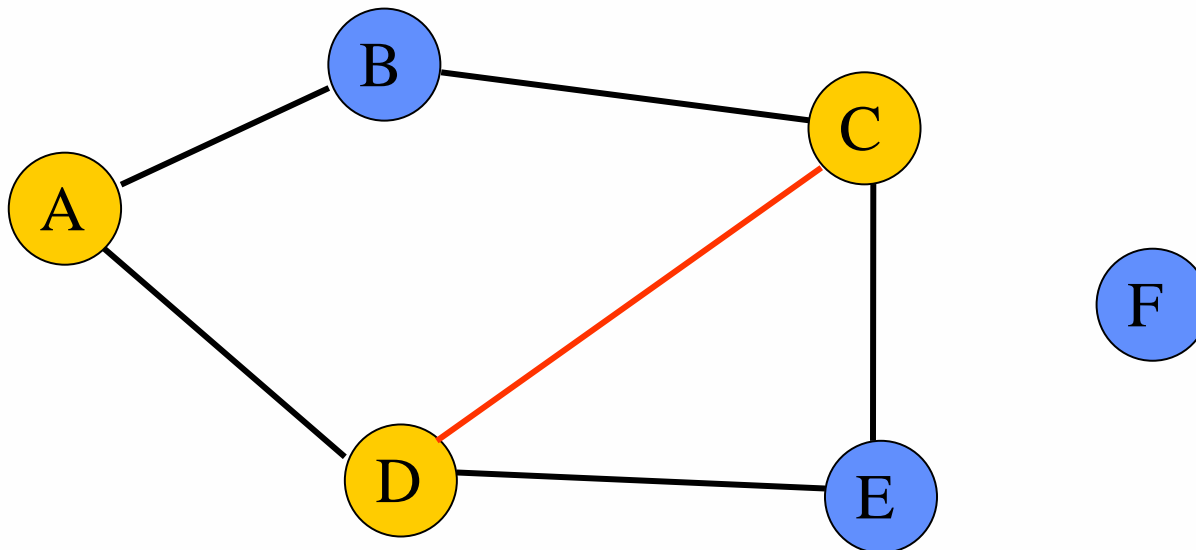


- Se G è un **grafo orientato** che non è fortemente connesso, ma la sua versione non orientata è connessa, diciamo che G è **debolmente connesso**



Questo grafo orientato **non** è fortemente connesso, ma **è debolmente connesso**

- Sia $G = (V, E)$ un grafo. Un *sottografo* di G è un grafo $H = (V^*, E^*)$ tale che $V^* \subseteq V$ e $E^* \subseteq E$

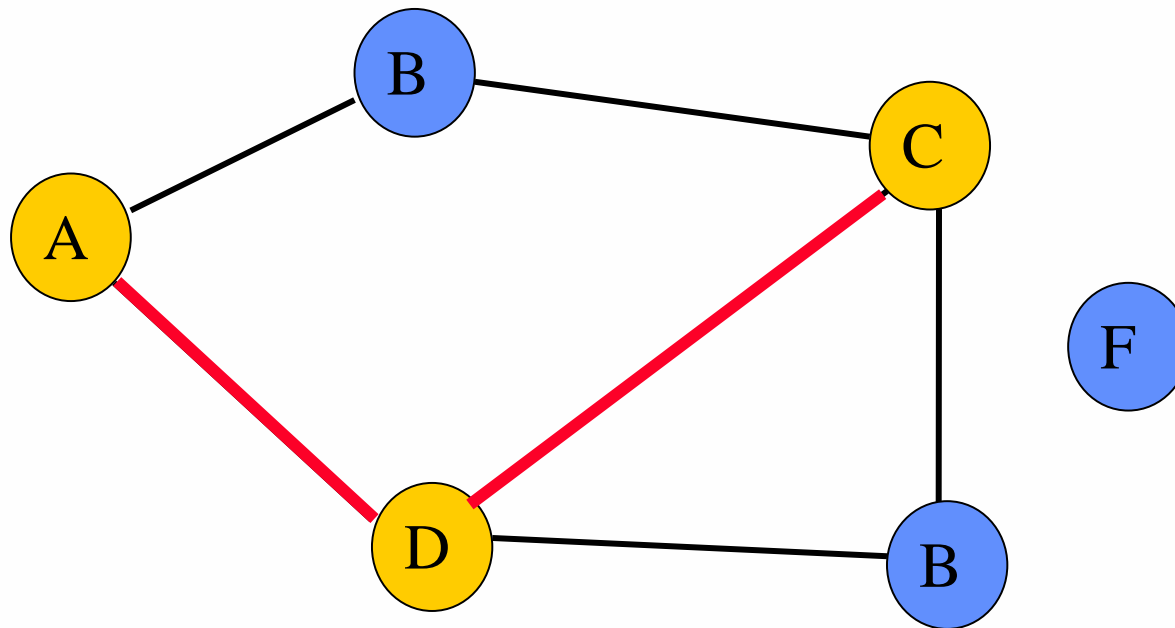


Esempio:

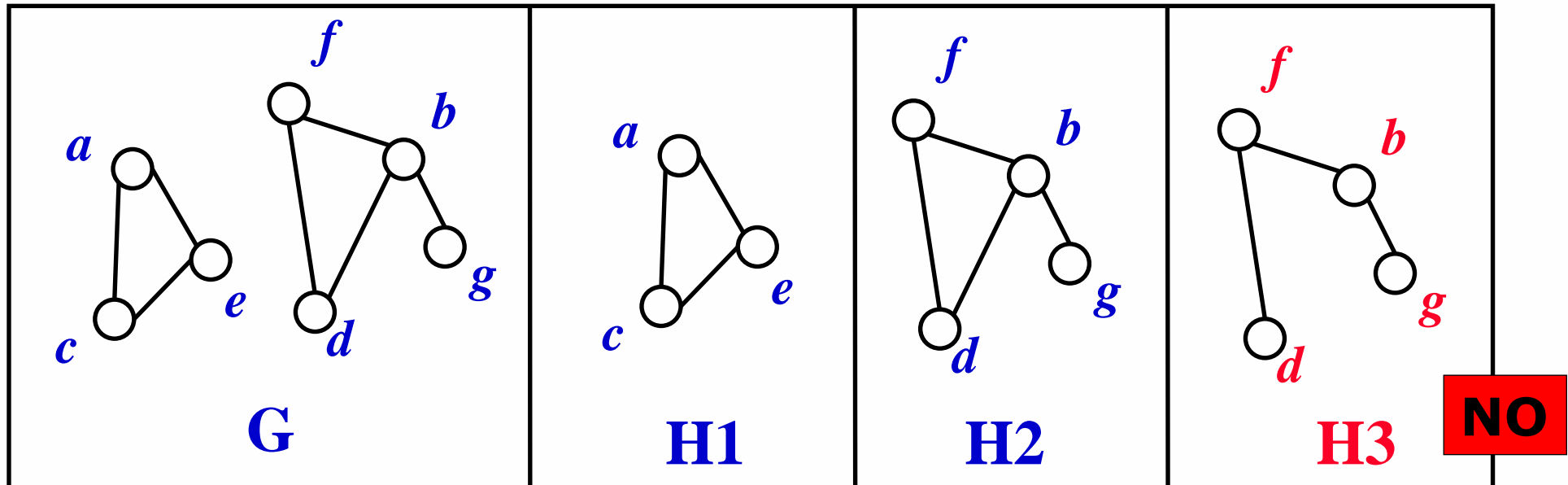
$$V^* = \{A, C, D\},$$

$$E^* = \{(C, D)\}.$$

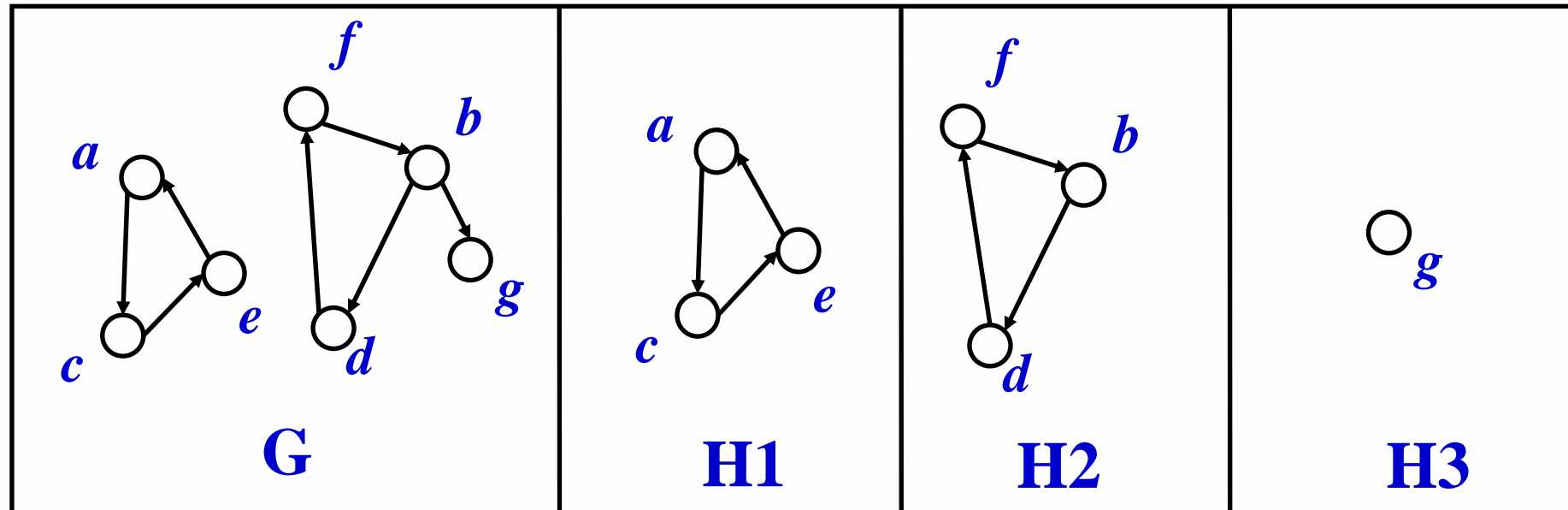
- Sia $G = (V, E)$ un grafo e $V^* \subseteq V$ un insieme di vertici.
- Il *sottografo* di G *indotto* da V^* è il grafo $H = (V^*, E^*)$ tale che $E^* = \{(v, w) \mid (v, w) \in E, \forall v, w \in V^*\}$
- Sia $G = (V, E)$ un grafo. Il *sottografo* $H = (V^*, E^*)$ di G è detto di *supporto* se: $V^* = V$



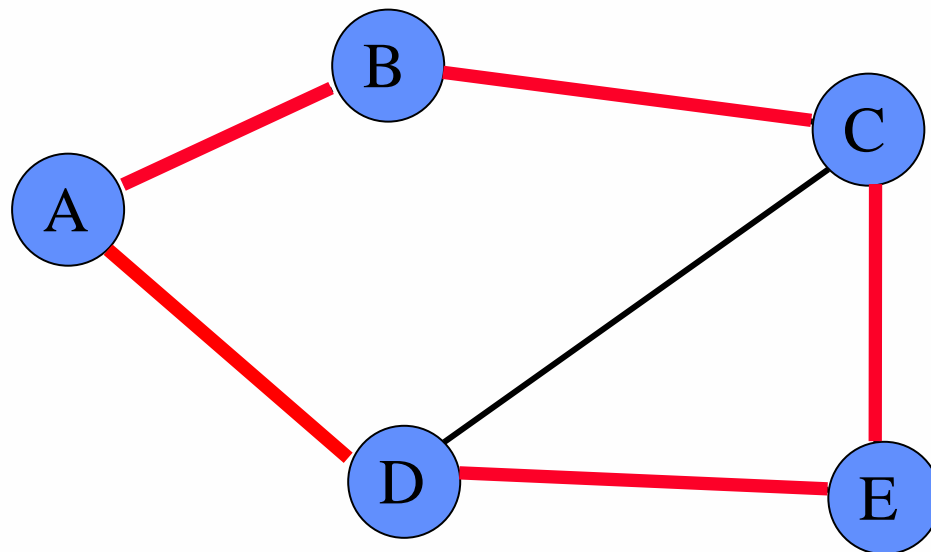
- Si dice **componente connessa** di un grafo non orientato G , un sottografo H di G che sia connesso e **massimale**
- Un sottografo connesso H di G viene detto massimale se si non si possono aggiungere ad H altri vertici o archi in modo che il grafo risultante sia ancora un sottografo connesso di G
- Esempi:



- Si dice **componente fortemente connessa** di un grafo orientato G , un sottografo H di G che sia fortemente connesso e **massimale**
- Esempi:

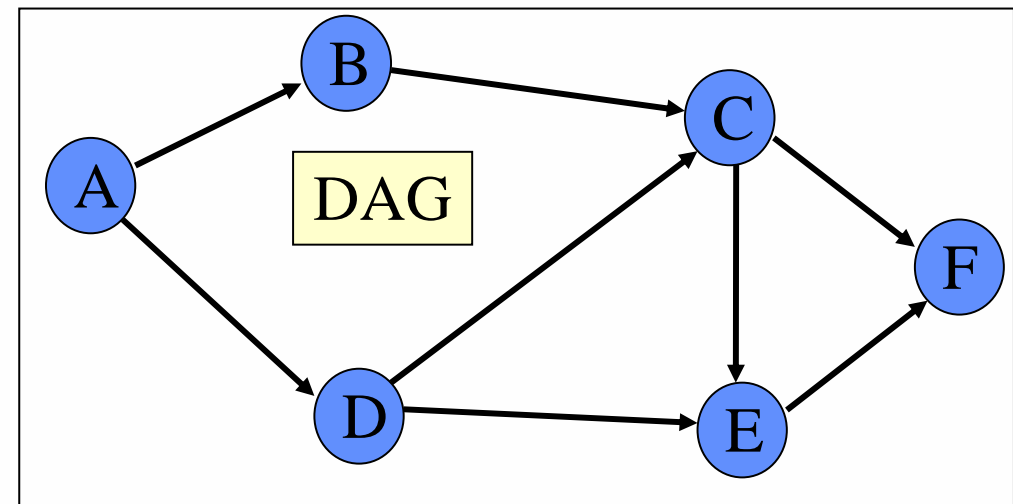
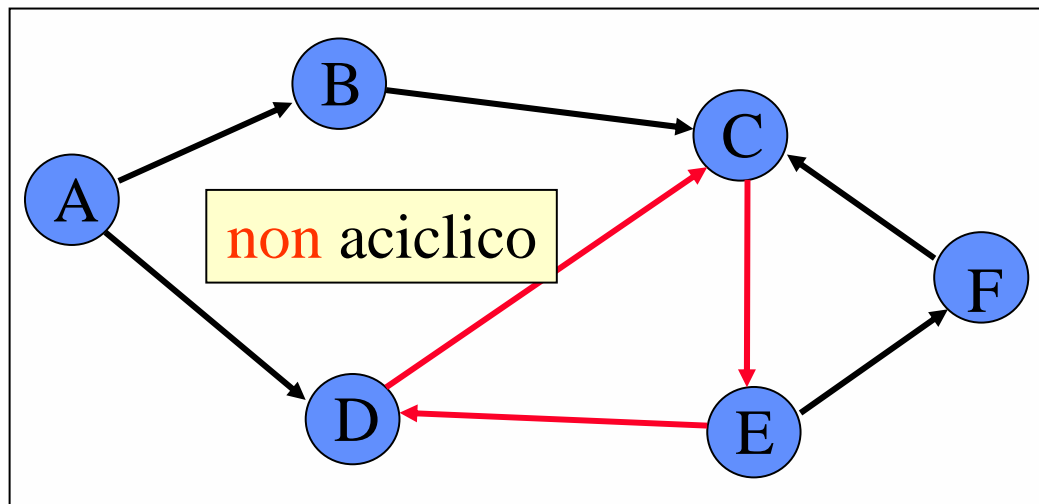
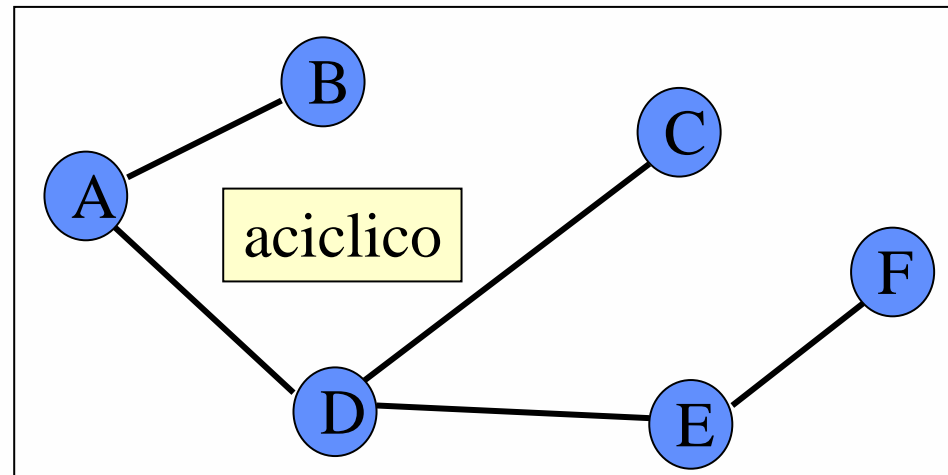


- Un **ciclo** in un grafo orientato è un cammino $\langle w_0, w_1, \dots, w_n \rangle$ di lunghezza almeno 3, tale che $w_0 = w_n$
- Un ciclo è semplice se i nodi w_1, \dots, w_{n-1} sono tutti distinti



Es.: il cammino
 $\langle A, B, C, E, D, A \rangle$
è un *ciclo semplice*

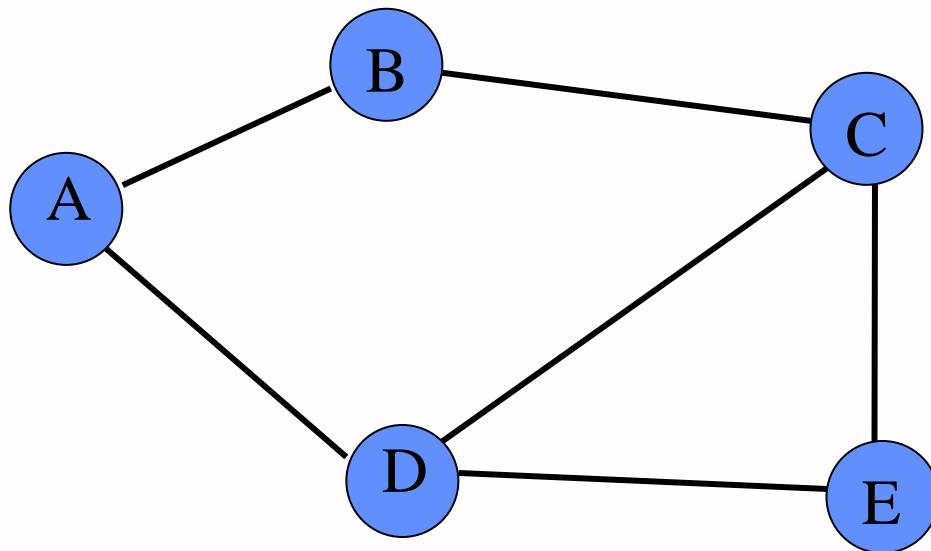
- ❑ Un grafo senza cicli è detto **aciclico**.
- ❑ Un grafo orientato aciclico è chiamato **DAG** (**D**irected **A**cylic **G**raph).
- ❑ Esempi:



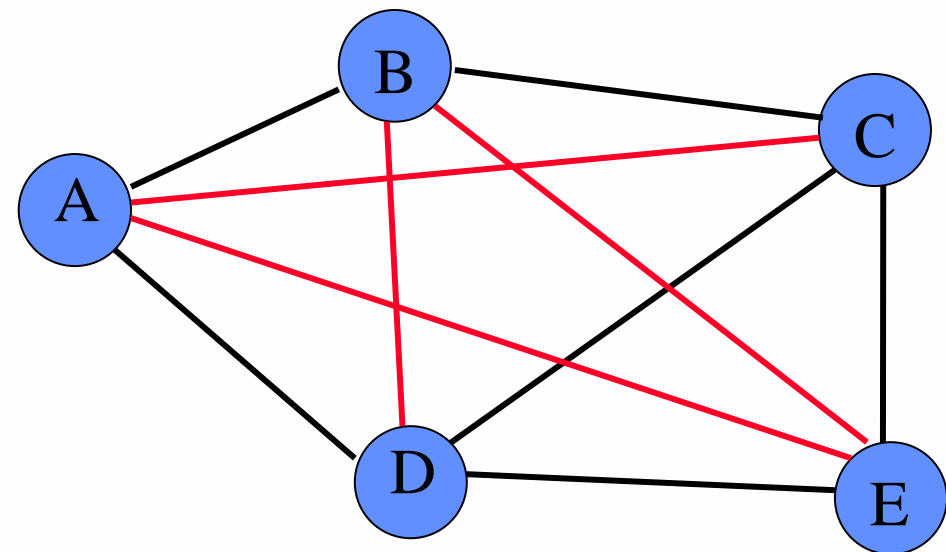
- ❑ Un **grafo completo** è un grafo che ha un arco tra ogni coppia di vertici.
- ❑ Un grafo completo con $n=|V|$ vertici ha un numero di archi pari a:

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

- ❑ Esempi:

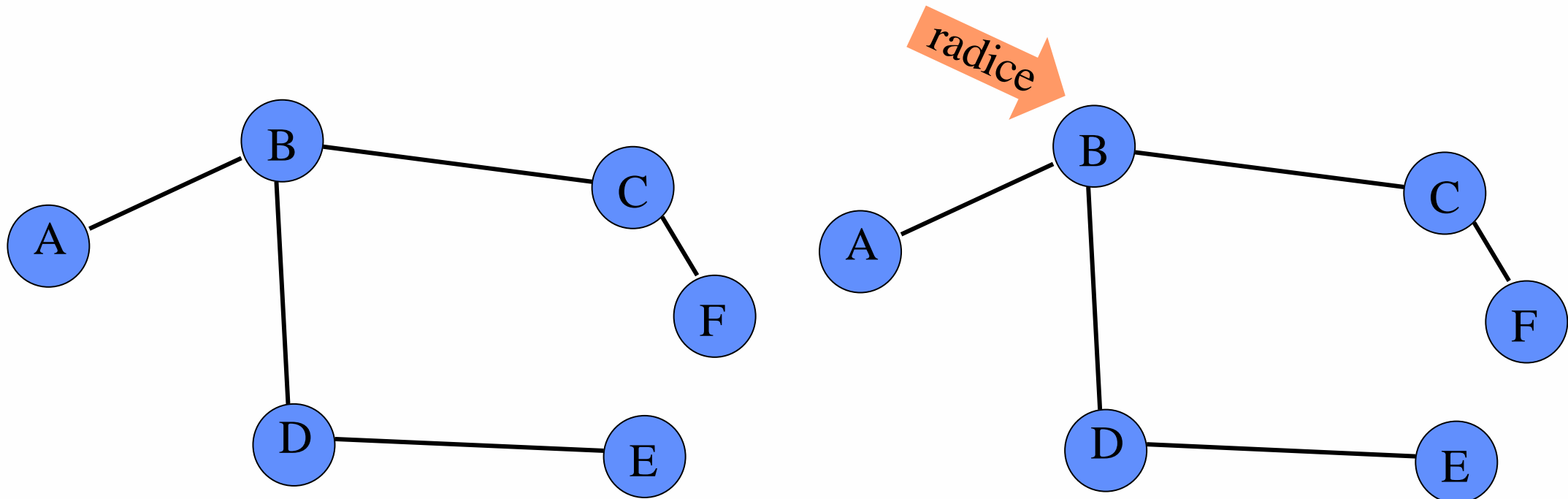


non completo



completo

- ❑ Un **albero libero** è un grafo non orientato connesso, aciclico.
- ❑ Se qualche vertice è detto radice, otteniamo un **albero radicato**.

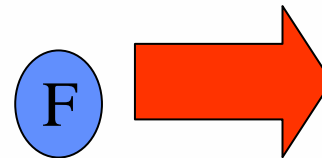
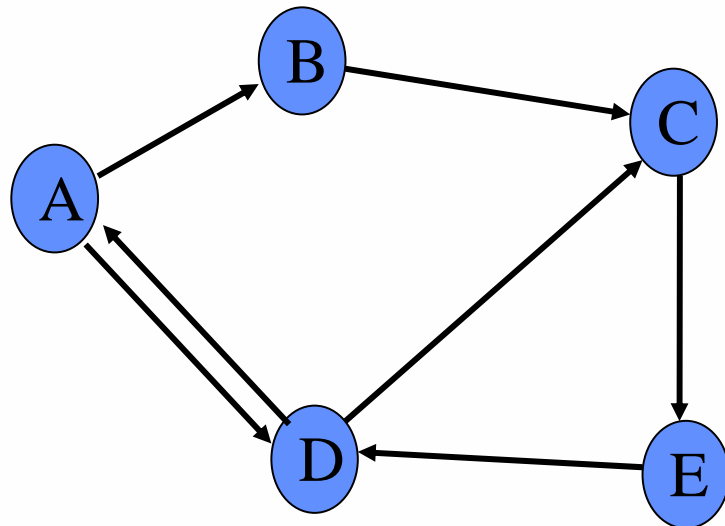


Strutture dati per i grafi

$$M[v, w] = \begin{cases} 1 & \text{se } (v, w) \in E \\ 0 & \text{altrimenti} \end{cases}$$

Memoria: $|V|^2$

□ Esempio:

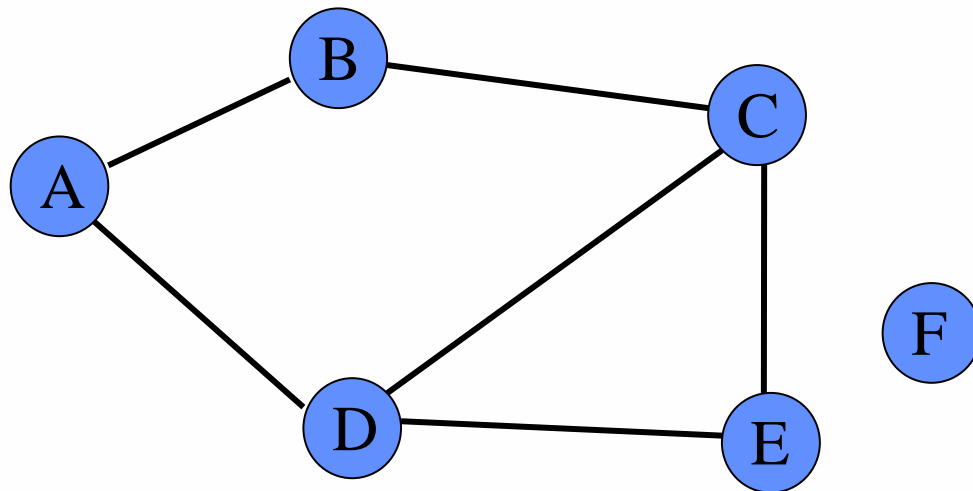


	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	0	1	0
D	1	0	1	0	0	0
E	0	0	0	1	0	0
F	0	0	0	0	0	0

$$M(v, w) = \begin{cases} 1 & \text{se } (v, w) \in E \text{ o } (w, v) \in E \\ 0 & \text{altrimenti} \end{cases}$$

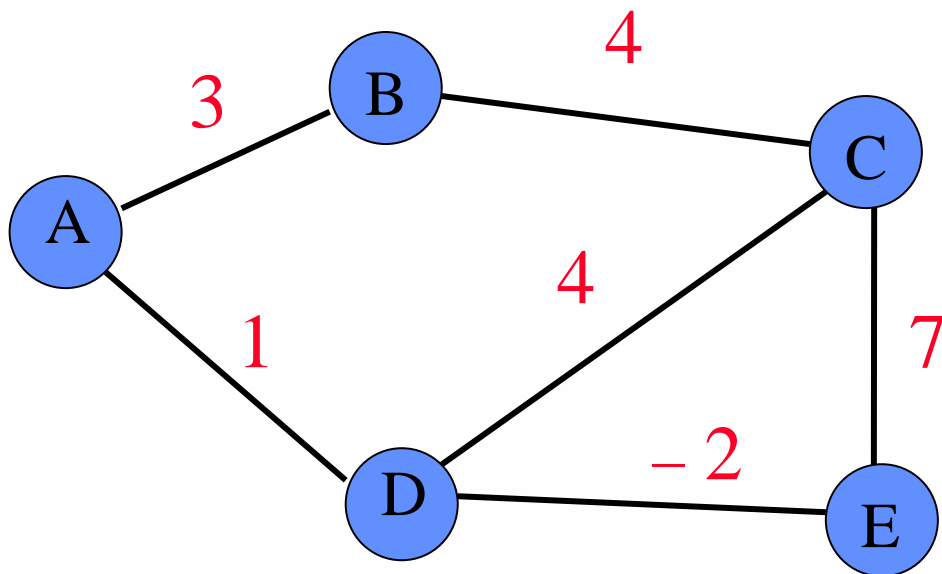
Memoria: $|V|^2$ o $|V|(|V|-1)/2$

□ Esempio:



	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0

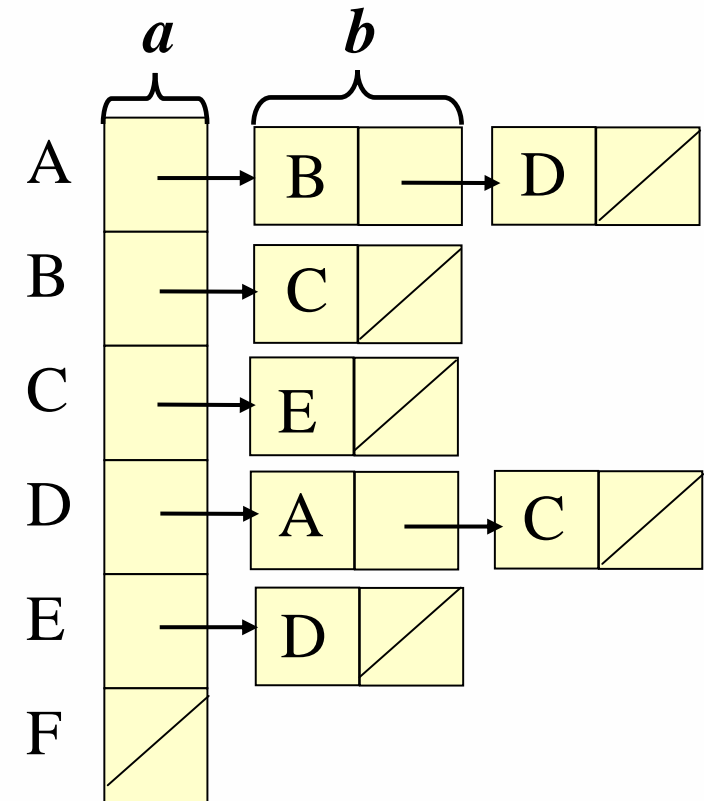
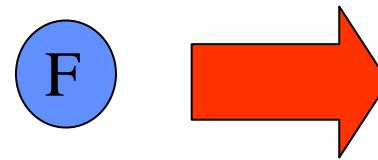
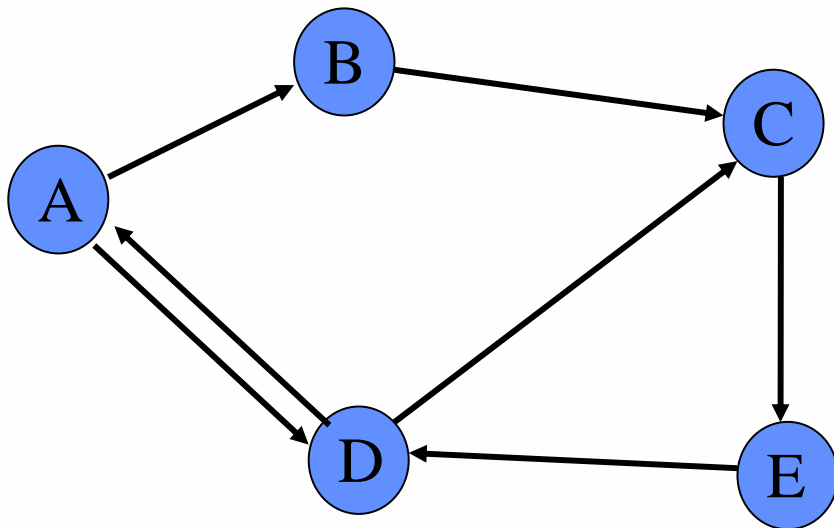
$$M(v, w) = \begin{cases} c(v, w) & \text{se } (v, w) \in E \\ \infty & \text{altrimenti} \end{cases}$$



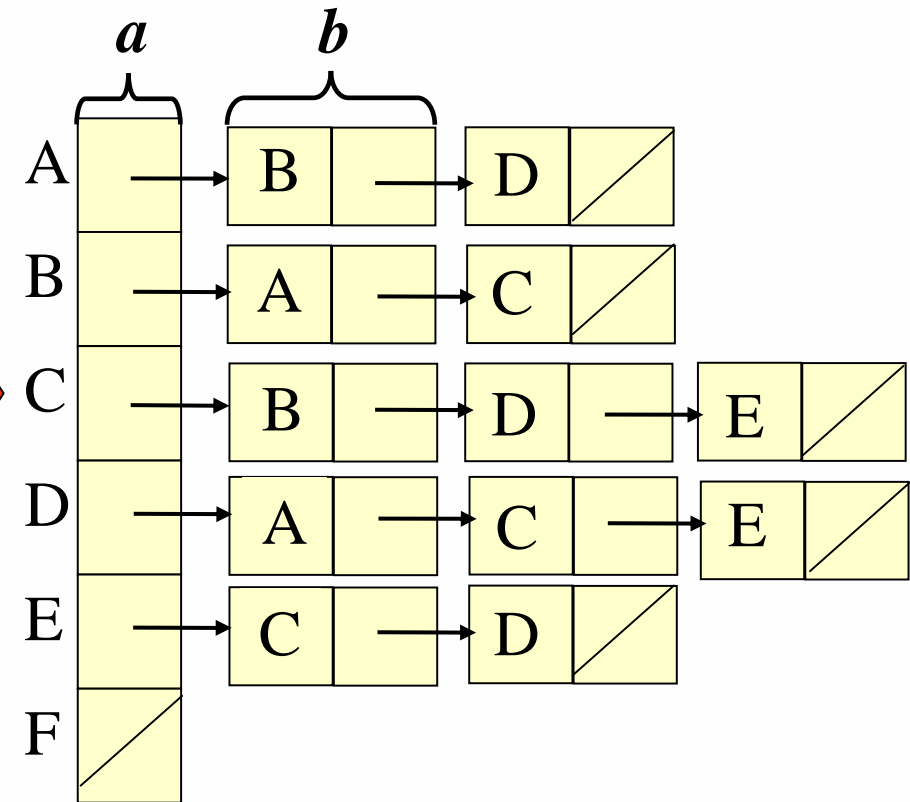
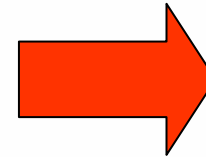
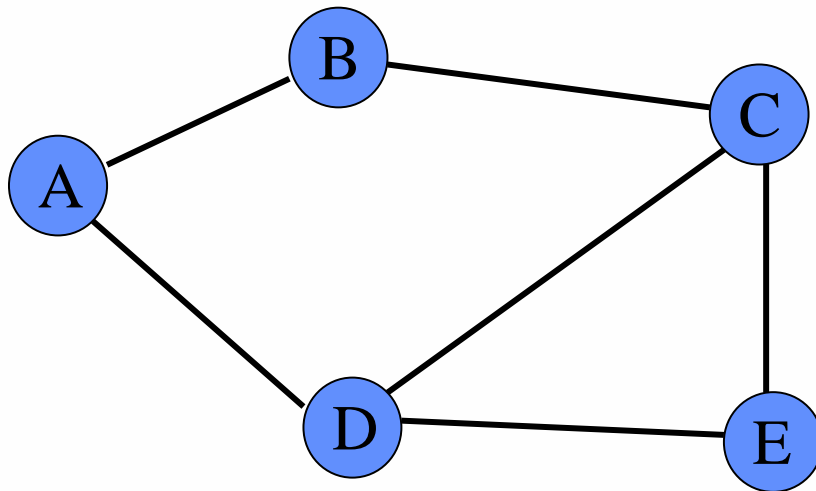
Memoria: $|V|^2$ o $|V|(|V|-1)/2$

	A	B	C	D	E	F
A	*	3	*	1	*	*
B	3	*	4	*	*	*
C	*	4	*	4	7	*
F	1	*	4	*	-2	*
D	*	*	7	-2	*	*
E	*	*	*	*	*	*
F	*	*	*	*	*	*

- ❑ La lista di adiacenza di un vertice v è definita come $\{ w \in V \mid (v,w) \in E \}$
- ❑ Un grafo può essere quindi rappresentato con un array di liste di adiacenze
- ❑ La memoria richiesta è $a|V| + b|E|$ (a e b costanti)
- ❑ Esempio:



- ❑ La lista di adiacenza di un vertice v è definita come $\{ w \in V \mid (v,w) \in E \vee (w,v) \in E \}$
- ❑ Lo stesso arco viene memorizzato in due liste di adiacenze, quindi la memoria richiesta è $a|V| + 2b|E|$ (a e b costanti)
- ❑ Esempio:



- ❑ Ci sono due tipi standard di rappresentazioni di grafi in un computer:
 - ▶ **Matrice di adiacenza**
 - Spazio richiesto $O(V^2)$
 - Verificare se il vertice v è adiacente a w richiede tempo $O(1)$
 - Elencare tutti gli archi costa $O(V^2)$
 - ▶ **Liste di adiacenza**
 - Spazio richiesto $O(V+E)$
 - Verificare se il vertice v è adiacente a w richiede tempo $O(V)$
 - Elencare tutti gli archi costa $O(V+E)$
- ❑ Nota: nelle espressioni, V corrisponde a $|V|$ e E corrisponde a $|E|$

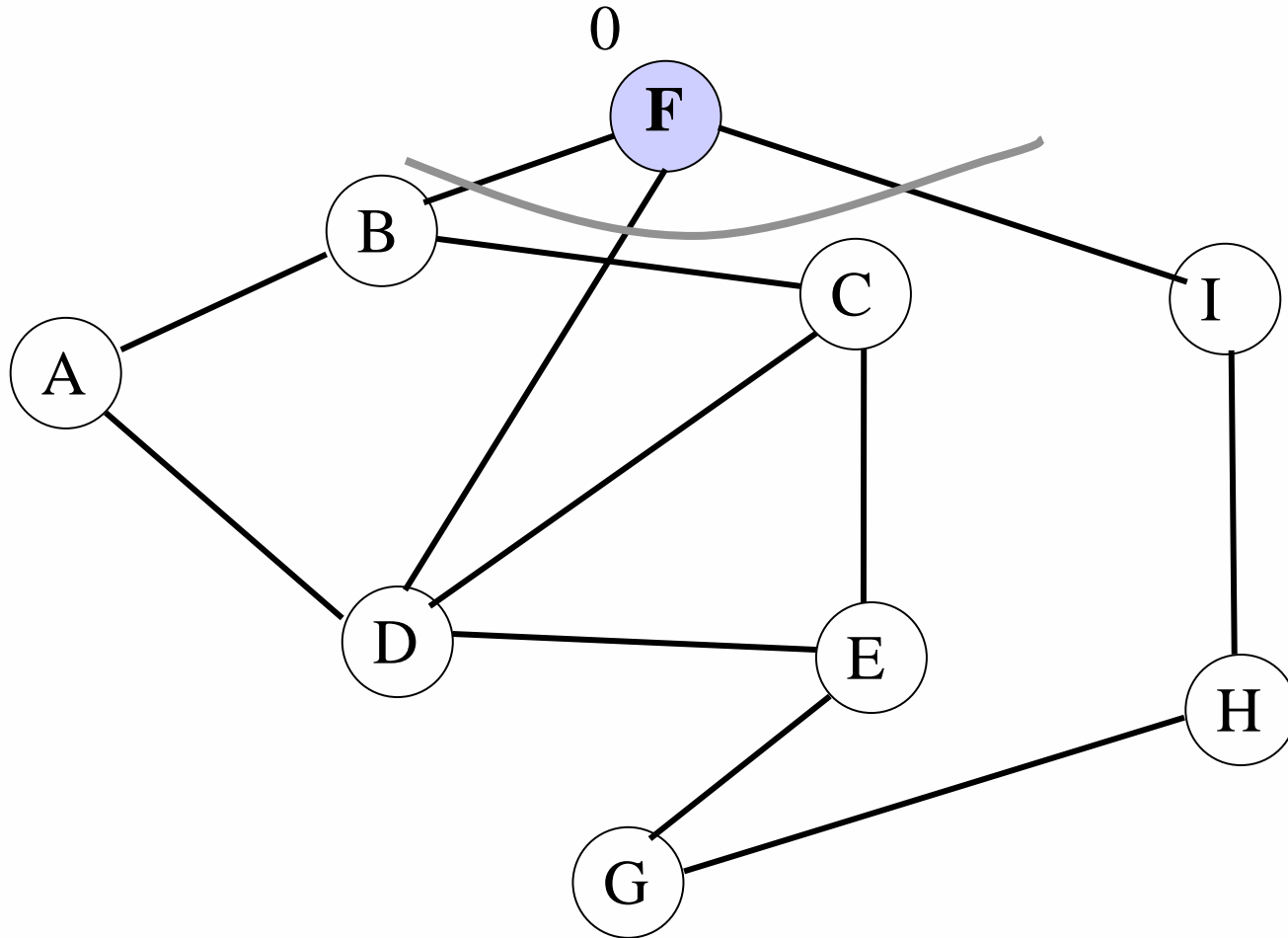
Visita di un grafo

- ❑ Dato un grafo $G = \langle V, E \rangle$ ed un vertice s di V (detto **sorgente**), intendiamo con visita di G a partire dalla sorgente s un attraversamento tale che:
 - ▶ ogni vertice di G raggiungibile da s venga visitato
 - ▶ ogni nodo raggiunto sia visitato **una volta sola**
- ❑ L'operazione di visita di un vertice può comportare diverse operazioni che dipendono dal problema per cui viene effettuata la visita del grafo
- ❑ Generalmente durante la visita ogni vertice viene marcato per evitare di visitarlo nuovamente
- ❑ Spesso viene anche memorizzato l'**albero di visita** che descrive il percorso attraverso cui si può giungere ad un vertice
- ❑ Esistono diversi algoritmi di visita, vedremo i principali:
 - ▶ Visita in ampiezza (breadth-first search)
 - ▶ Visita in profondità (depth-first search)

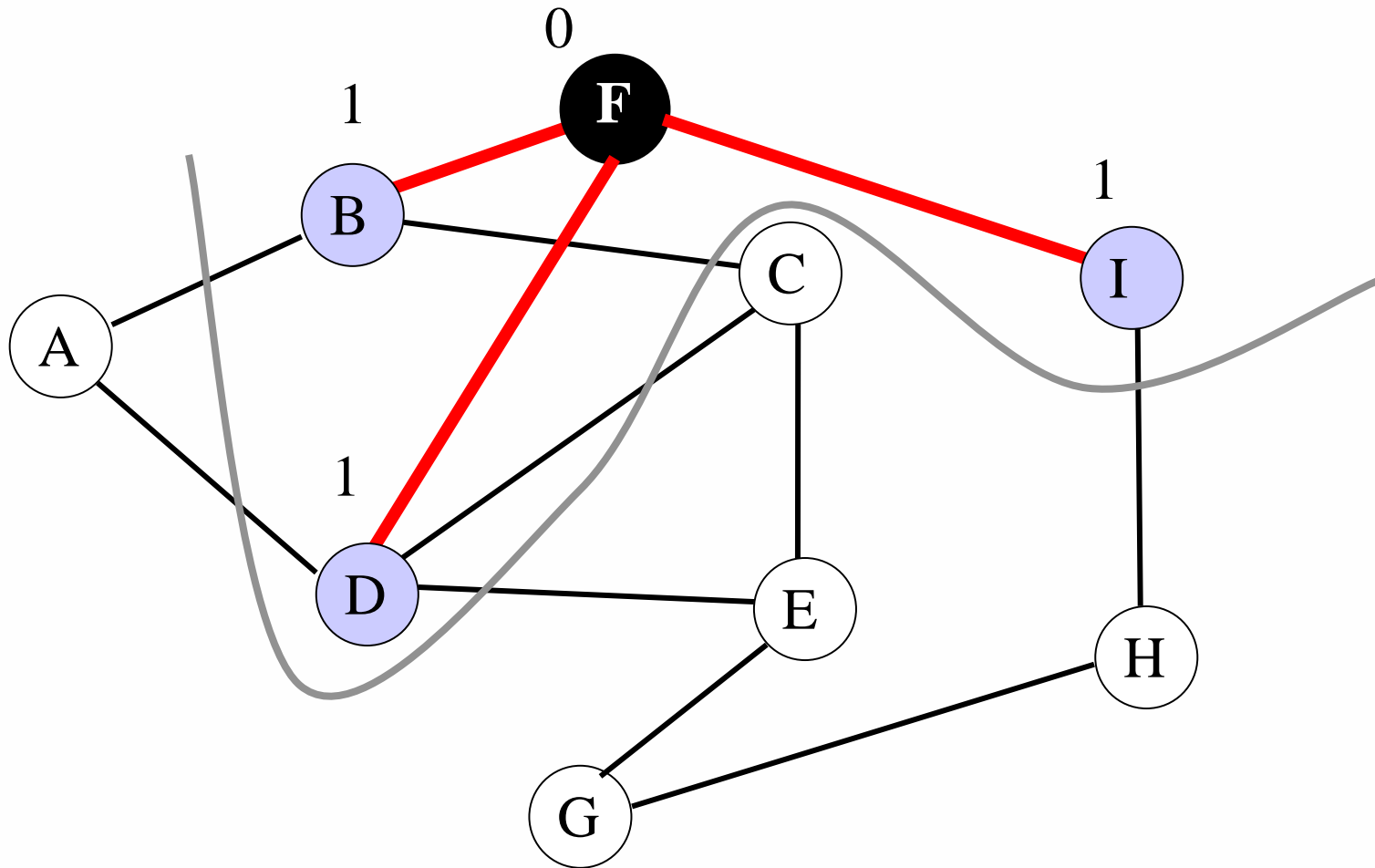
- ❑ Come funziona?
 - ▶ **Visita i vertici a distanze crescenti dalla sorgente:** visita tutti i vertici a distanza k prima di visitare i vertici a distanza $k+1$
- ❑ Output
 - ▶ **Albero BF (breadth-first)** contenente tutti i vertici raggiungibili da s e tale che il cammino da s ad un nodo nell'albero corrisponde al cammino più breve nel grafo
 - ▶ **Distanza minima da s a tutti i vertici raggiungibili** numero di archi attraversati per andare da s ad un vertice

```
BFS(G, s)
  for each v in V do
    mark[v] = white
   $\pi$ [s] = nil
  F =  $\emptyset$ 
  F.enqueue(s)
  mark[s] = gray
  dist[s] = 0
  while F  $\neq$   $\emptyset$  do
    u = F.dequeue()
    "visita il vertice u"
    for each v in adj[u] do
      if (mark[v] == white)
        then
          mark[v] = gray
          dist[v] = dist[u] + 1
          F.enqueue(v)
           $\pi$ [v] := u
    mark[u] = black
```

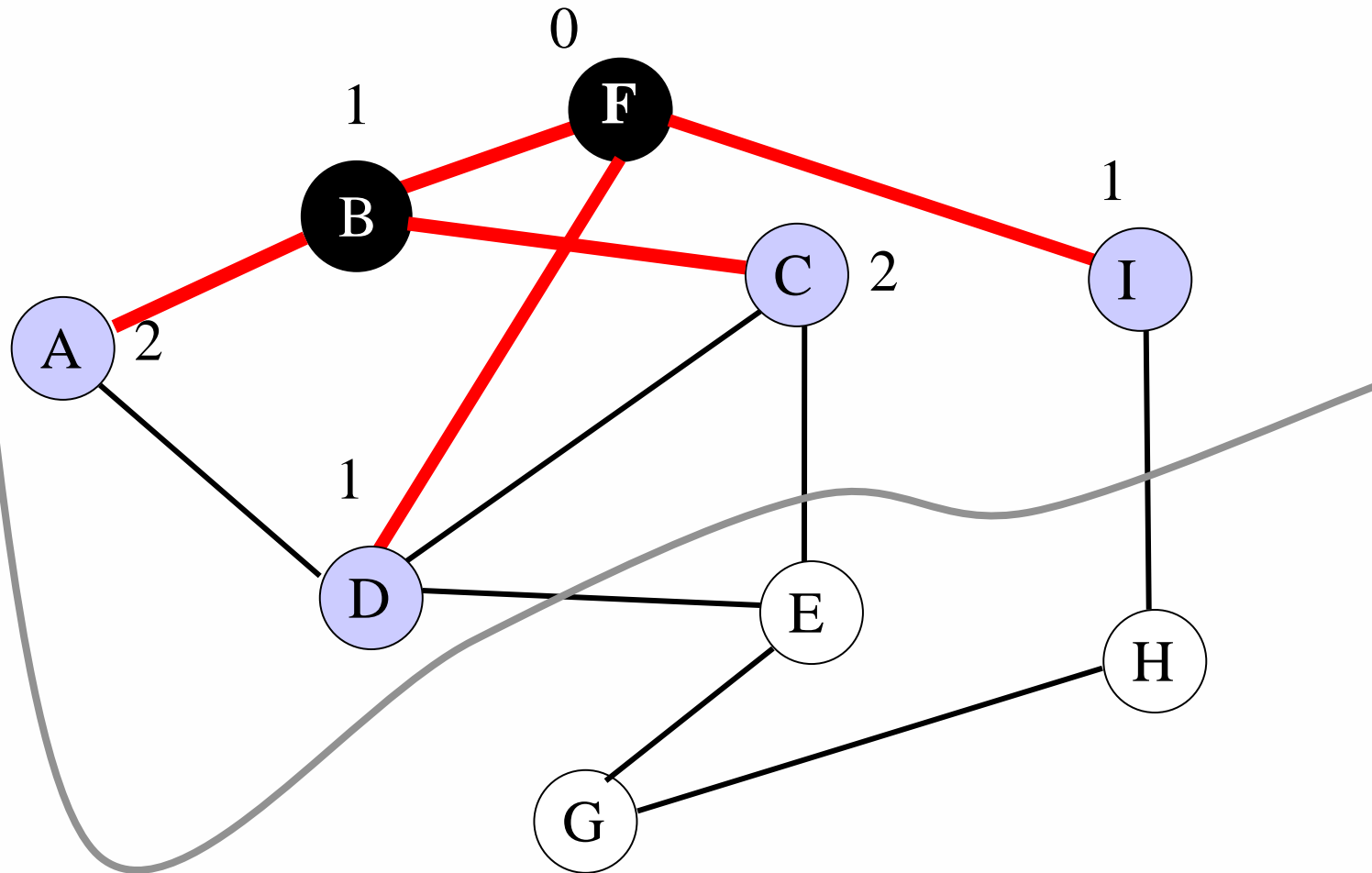
- ❑ *F* è l'insieme dei nodi scoperti, detto anche **frontiera**
- ❑ *mark*[*v*] è la marcatura del nodo *v*
 - ▶ **white**: il nodo non è ancora stato raggiunto
 - ▶ **gray**: il nodo è stato raggiunto ma non è ancora visitato (è in *F*)
 - ▶ **black**: il nodo è stato visitato
- ❑ *dist*[*v*] è la distanza del nodo *v* dal vertice *s*
- ❑ π [*v*] è il predecessore di *v* nell'albero di copertura
- ❑ Costo:
 - ▶ $O(|V| + |E|)$ liste
 - ▶ $O(|V|^2)$ matrice



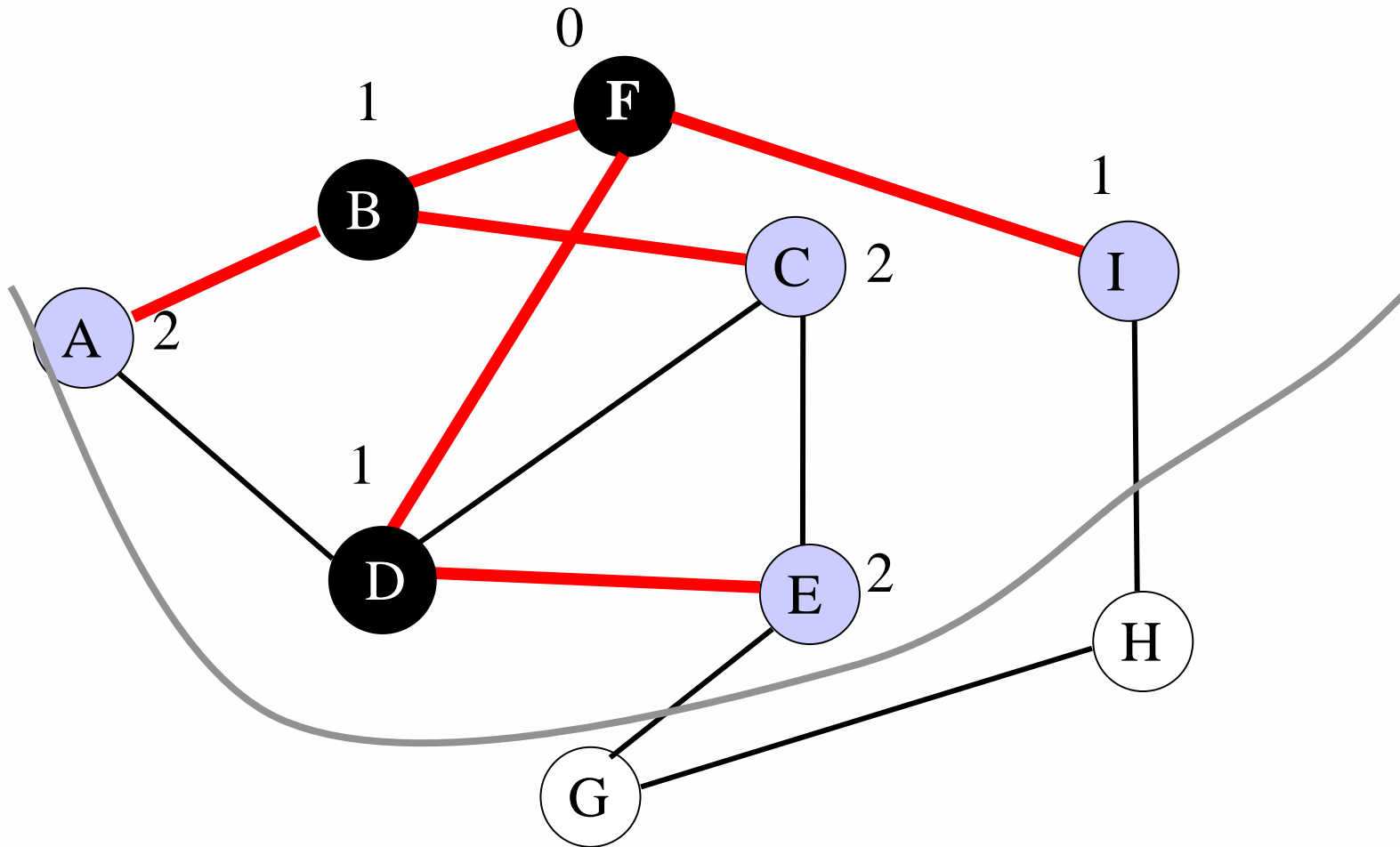
Coda: {F}



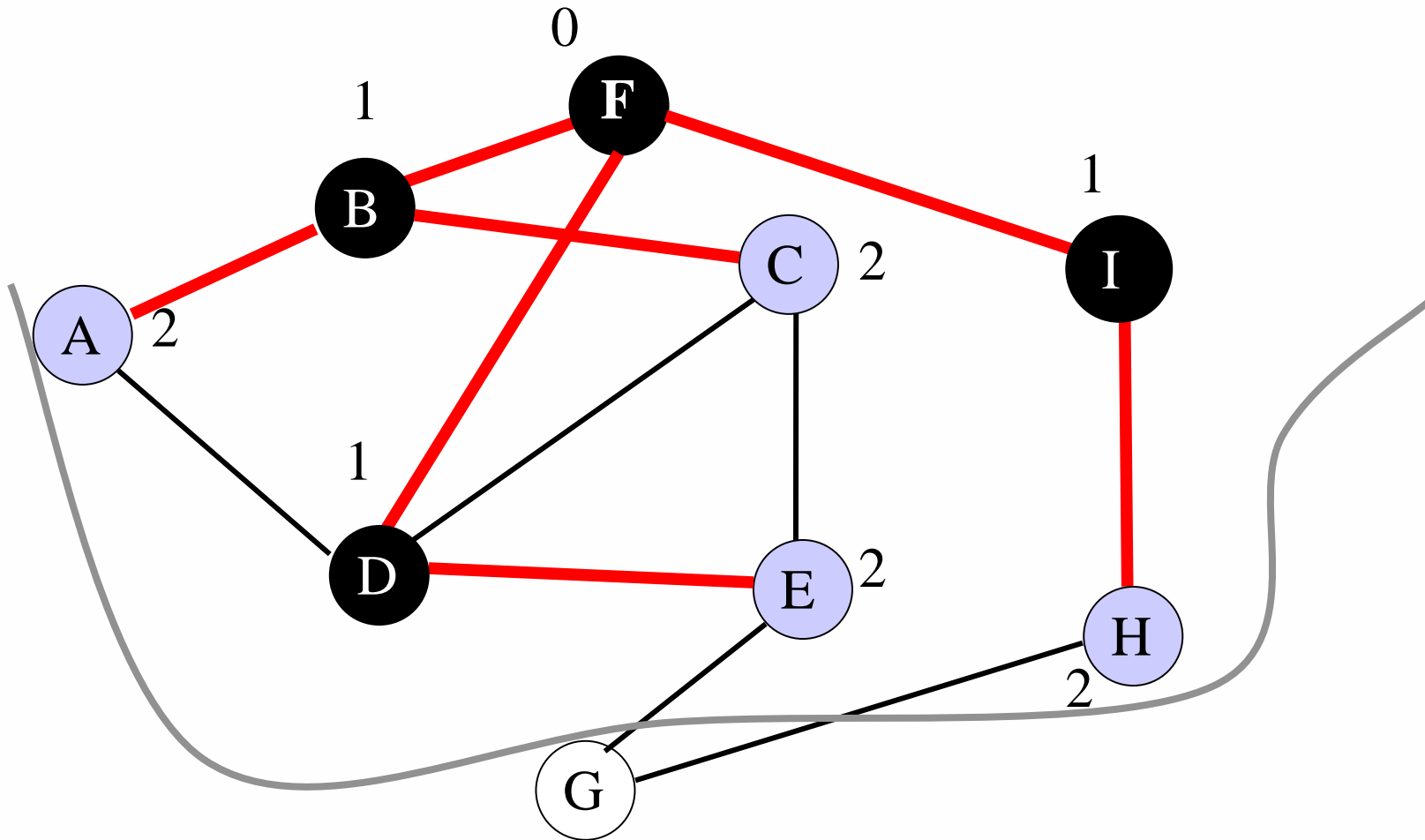
Coda: {B, D, I}



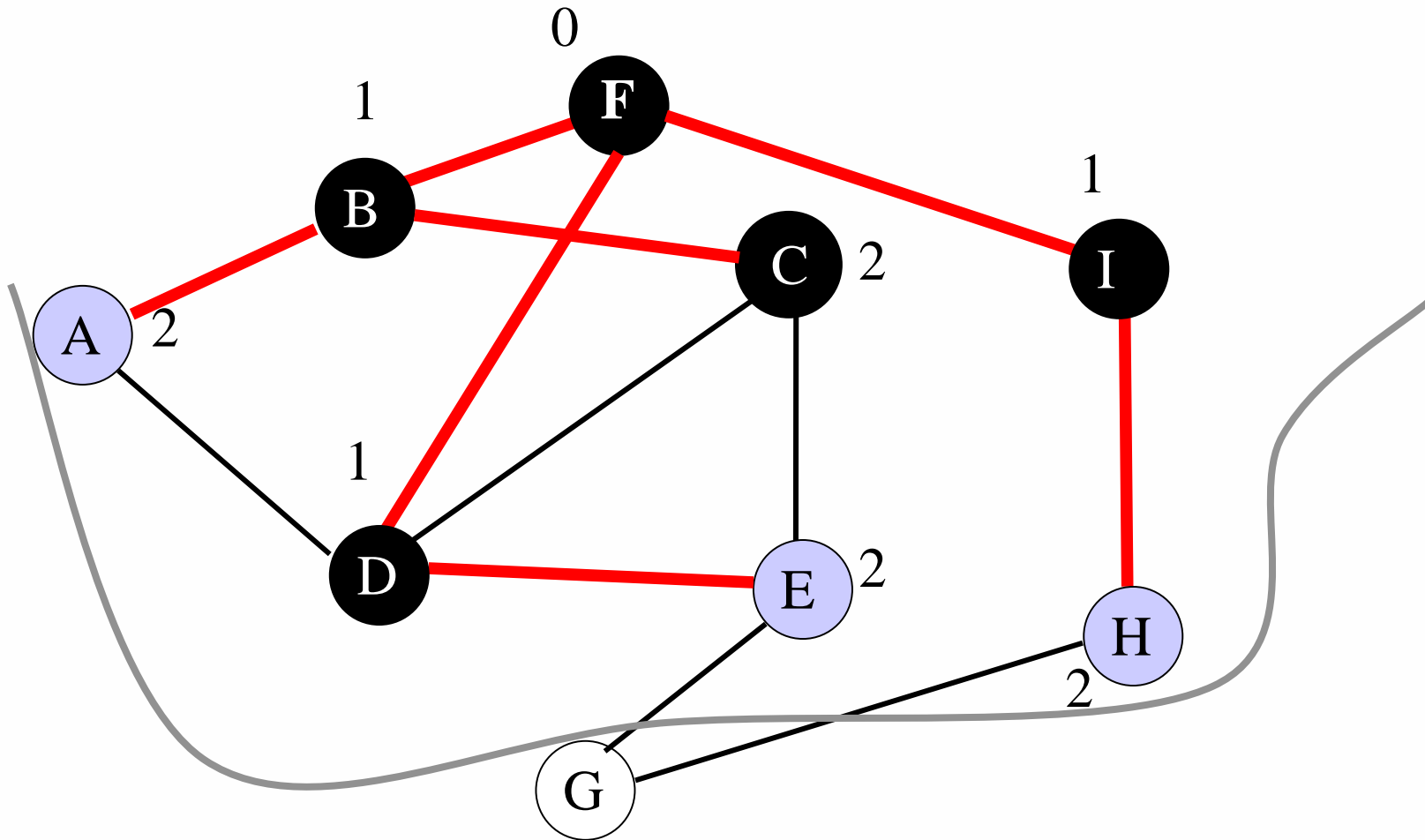
Coda: {D, I, C, A}



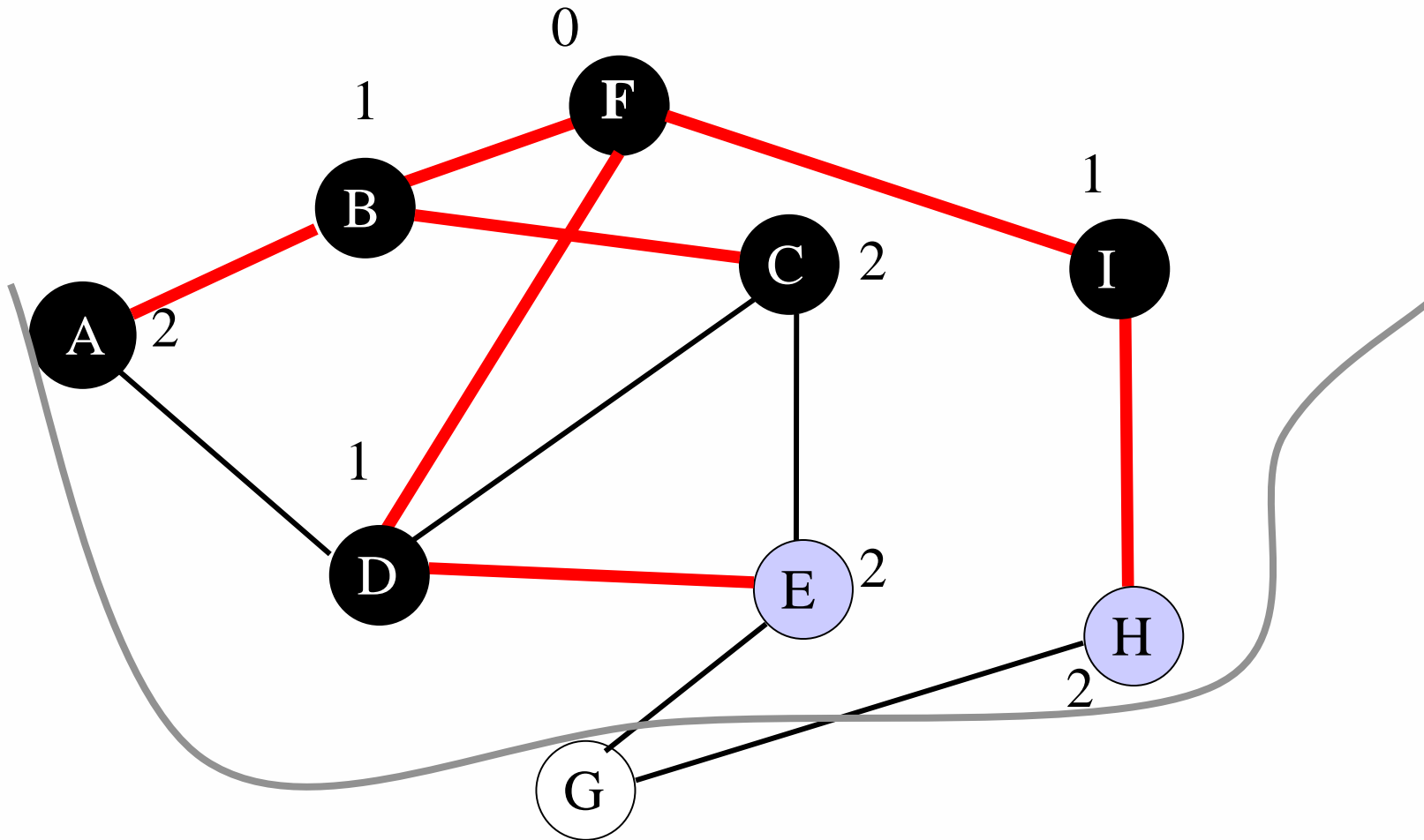
Coda: {I, C, A, E}



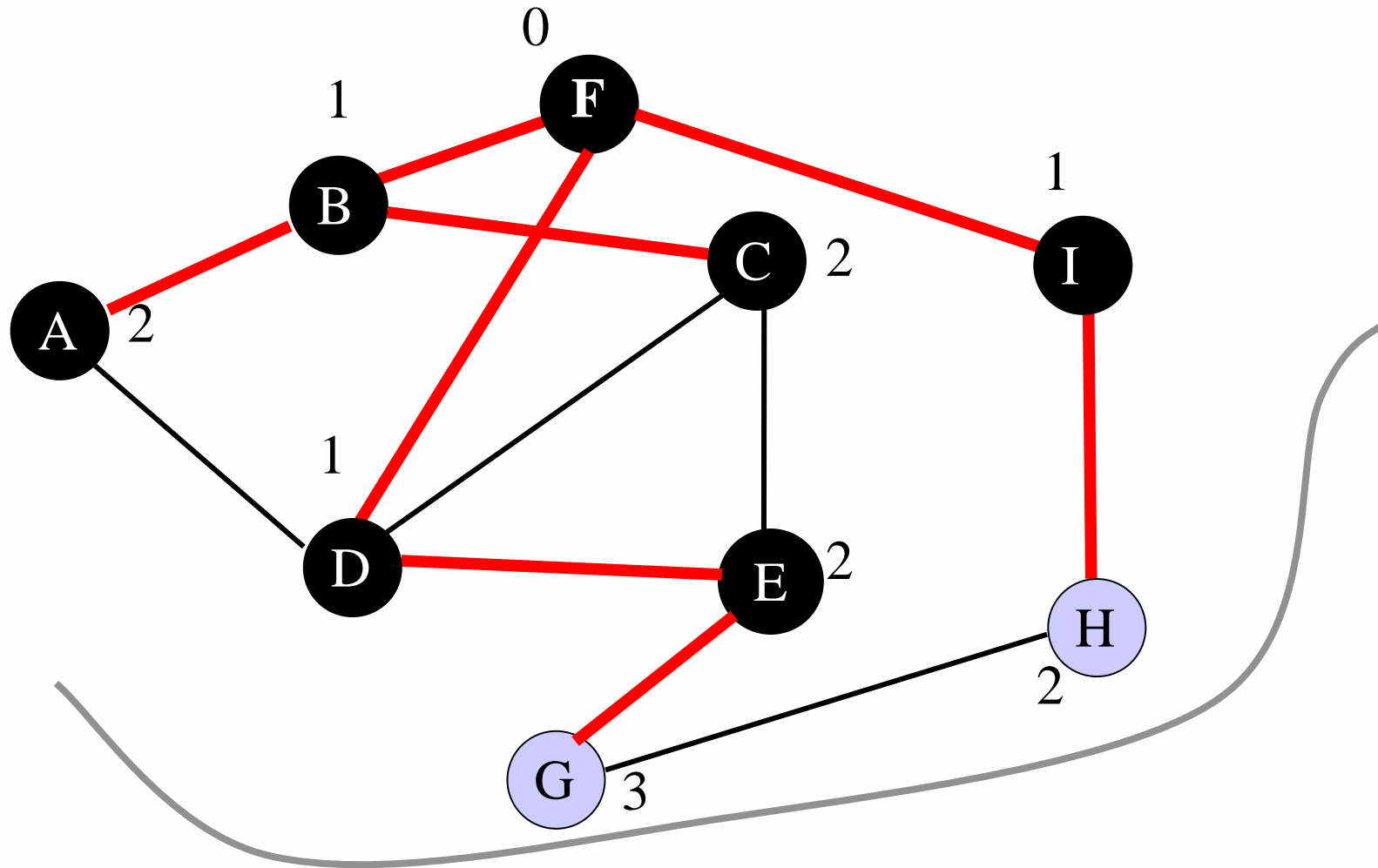
Coda: {C, A, E, H}



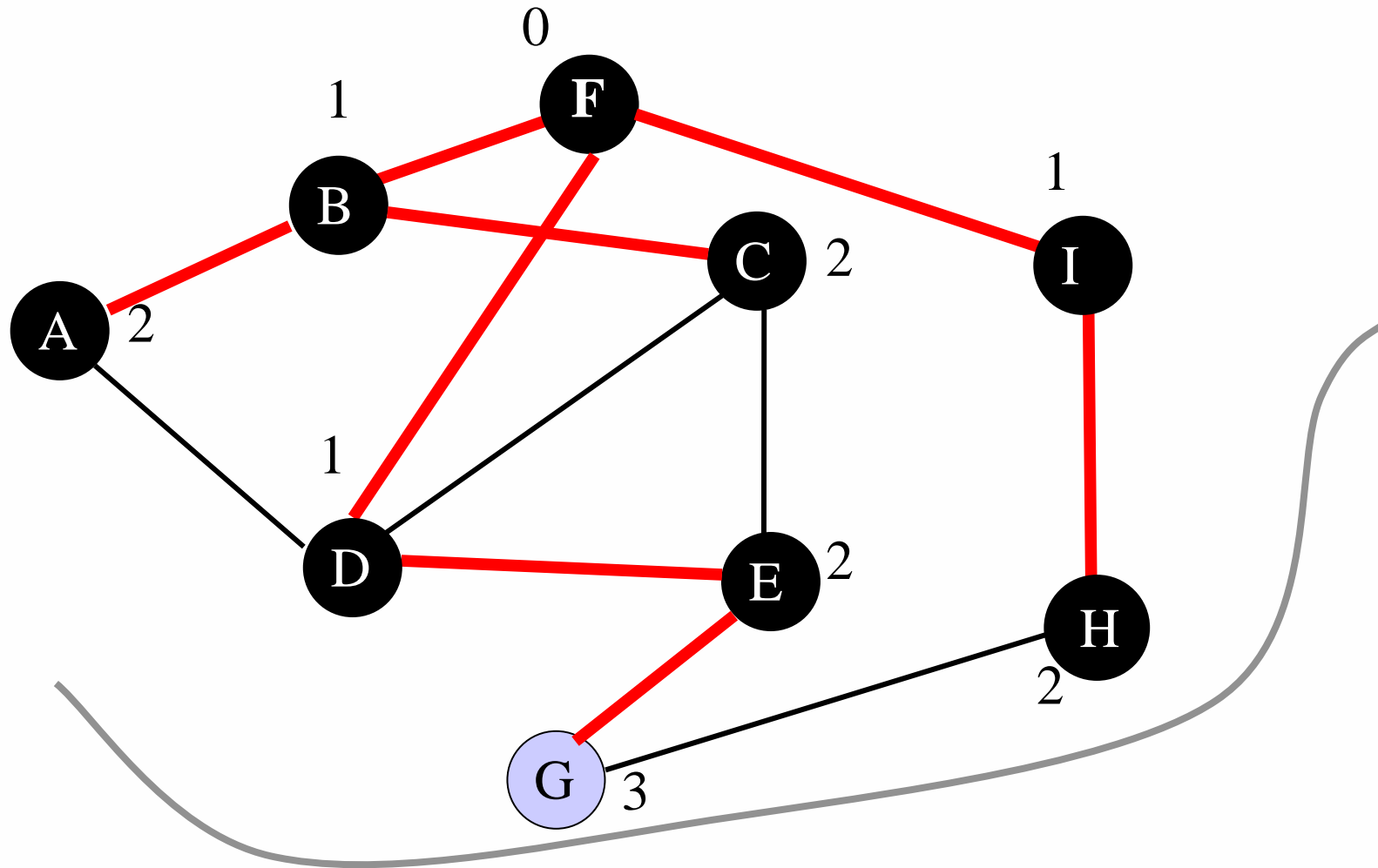
Coda: {A, E, H}



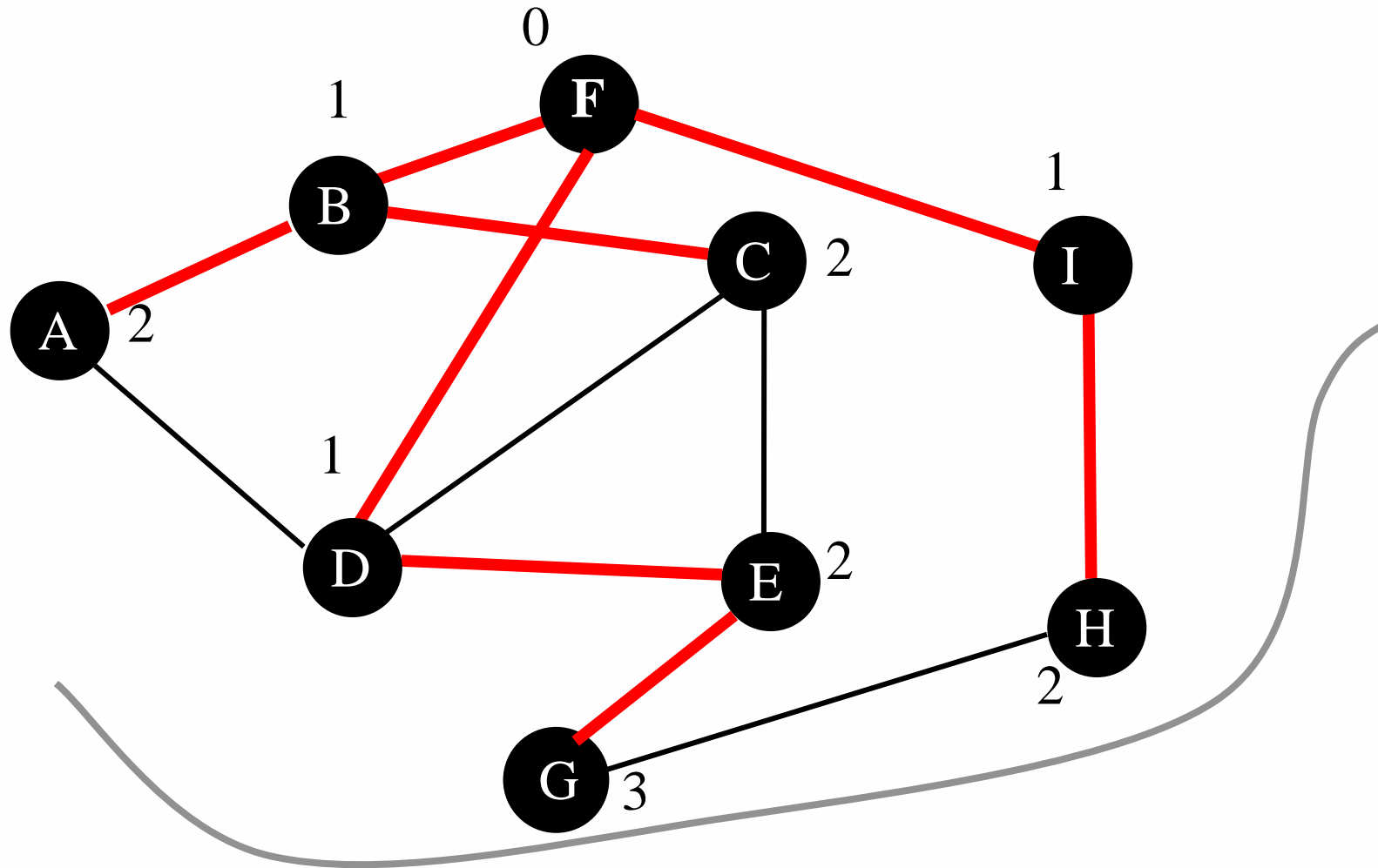
Coda: {E, H}



Coda: {H, G}



Coda: {G}



Coda: { }

- ❑ Come funziona?
 - ▶ **Visita i nodi andando il "più lontano possibile" nel grafo:** prima di concludere la visita di un vertice, viene conclusa la visita di tutti i suoi successori
- ❑ Output:
 - ▶ **Albero DF (depth-first)** contenente tutti i vertici raggiungibili da s seguendo il percorso effettuato dalla visita depth-first
 - ▶ **Istante di tempo in cui ogni vertice viene raggiunto e in cui la visita termina**

DFS (G, s)

```
mark[s] = gray
```

```
time = time+1
```

```
d[s] = time
```

```
for each v in adj[s] do
```

```
  if mark[v] == white
```

```
  then
```

```
     $\pi[v] = s$ 
```

```
    DFS(G, v)
```

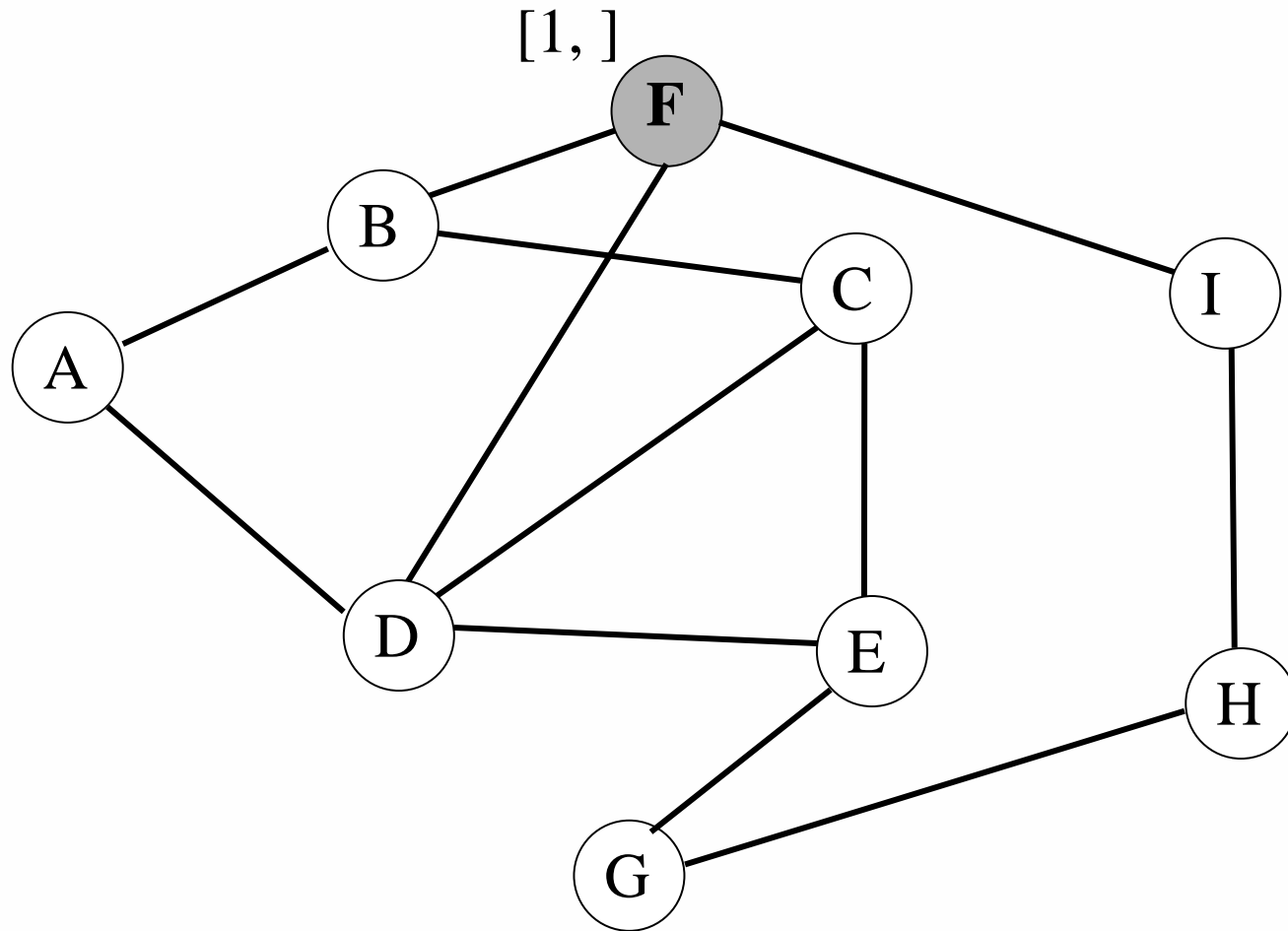
```
  "visita il vertice s"
```

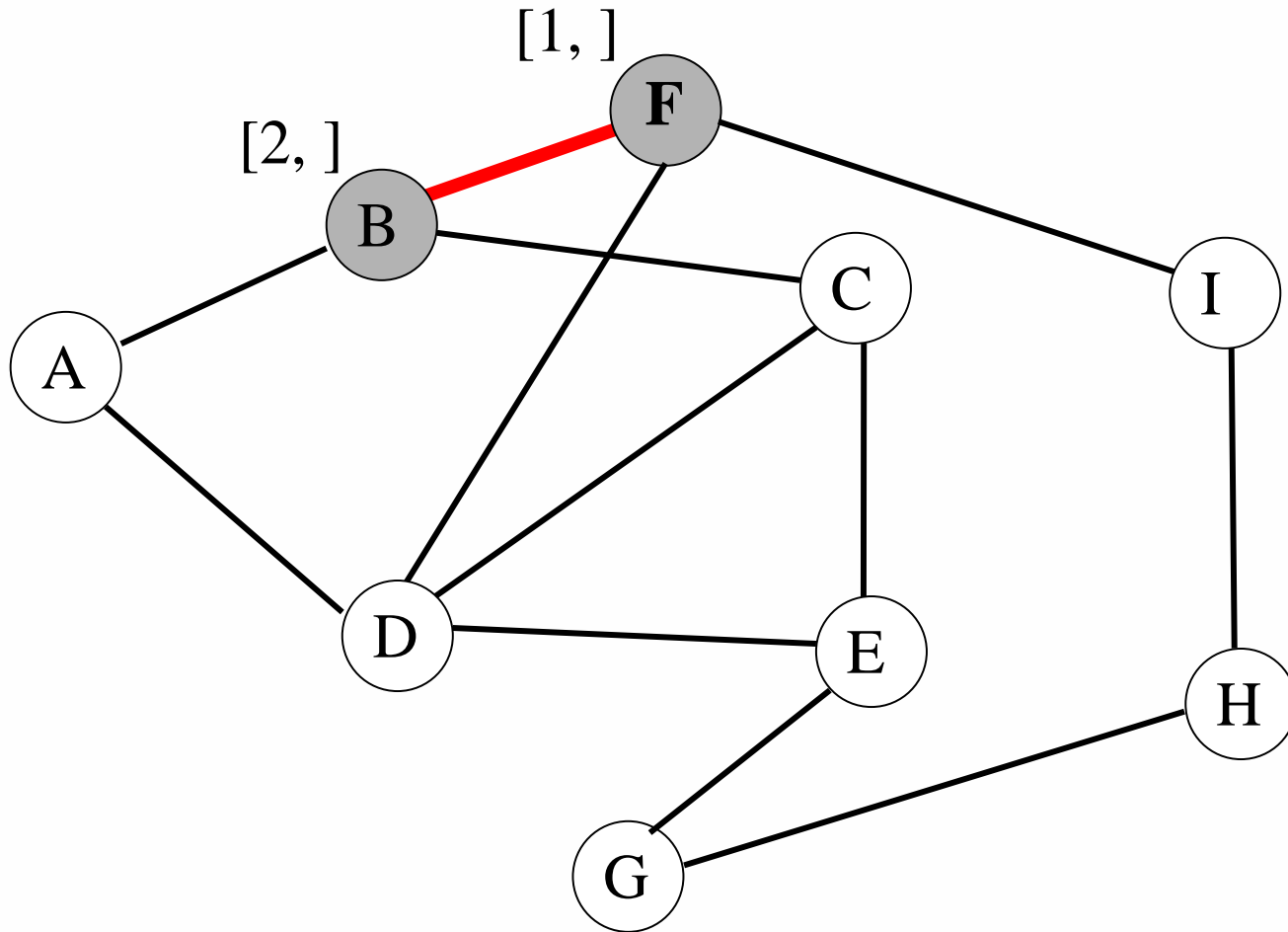
```
time = time+1
```

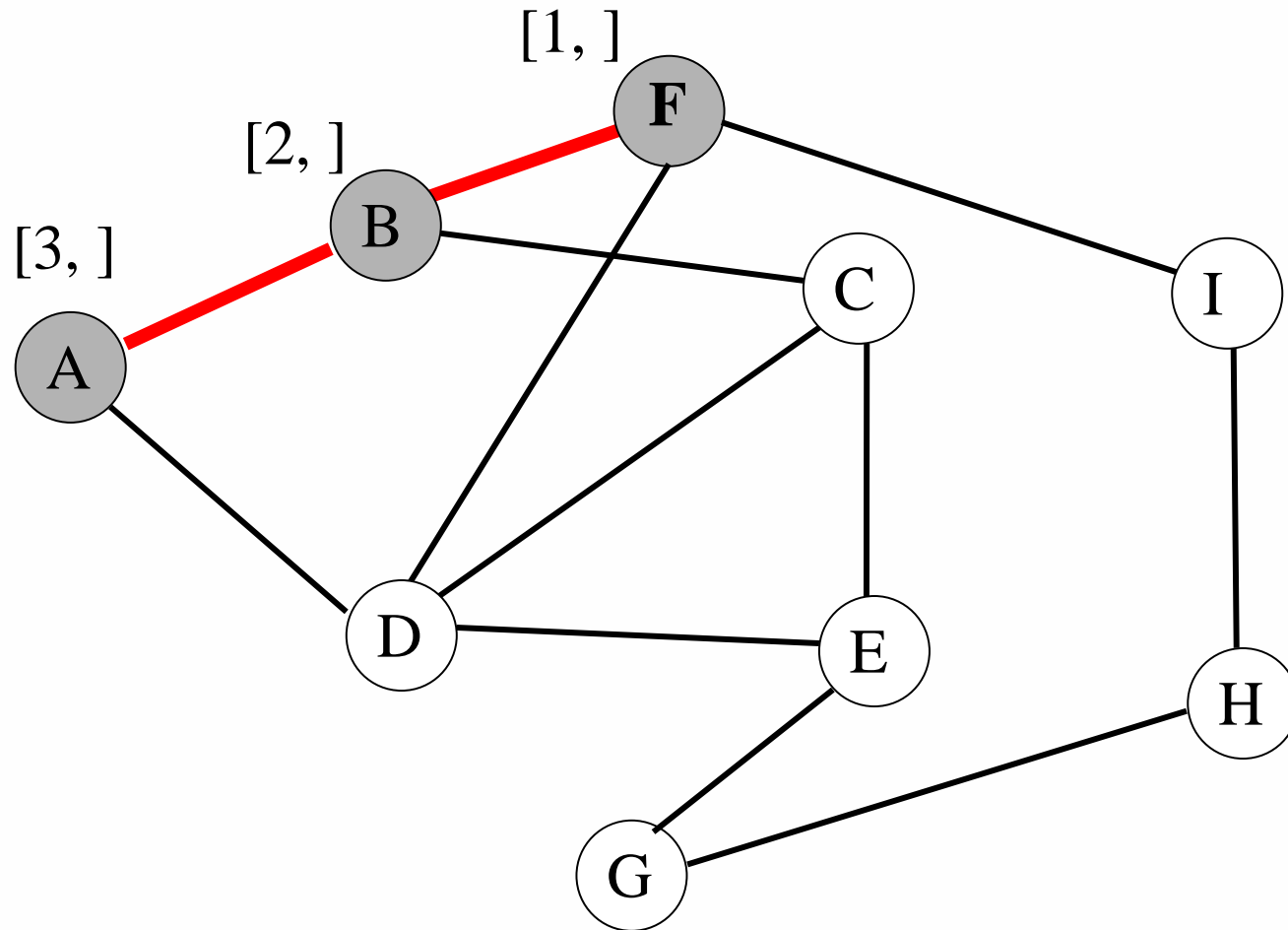
```
f[s] = time
```

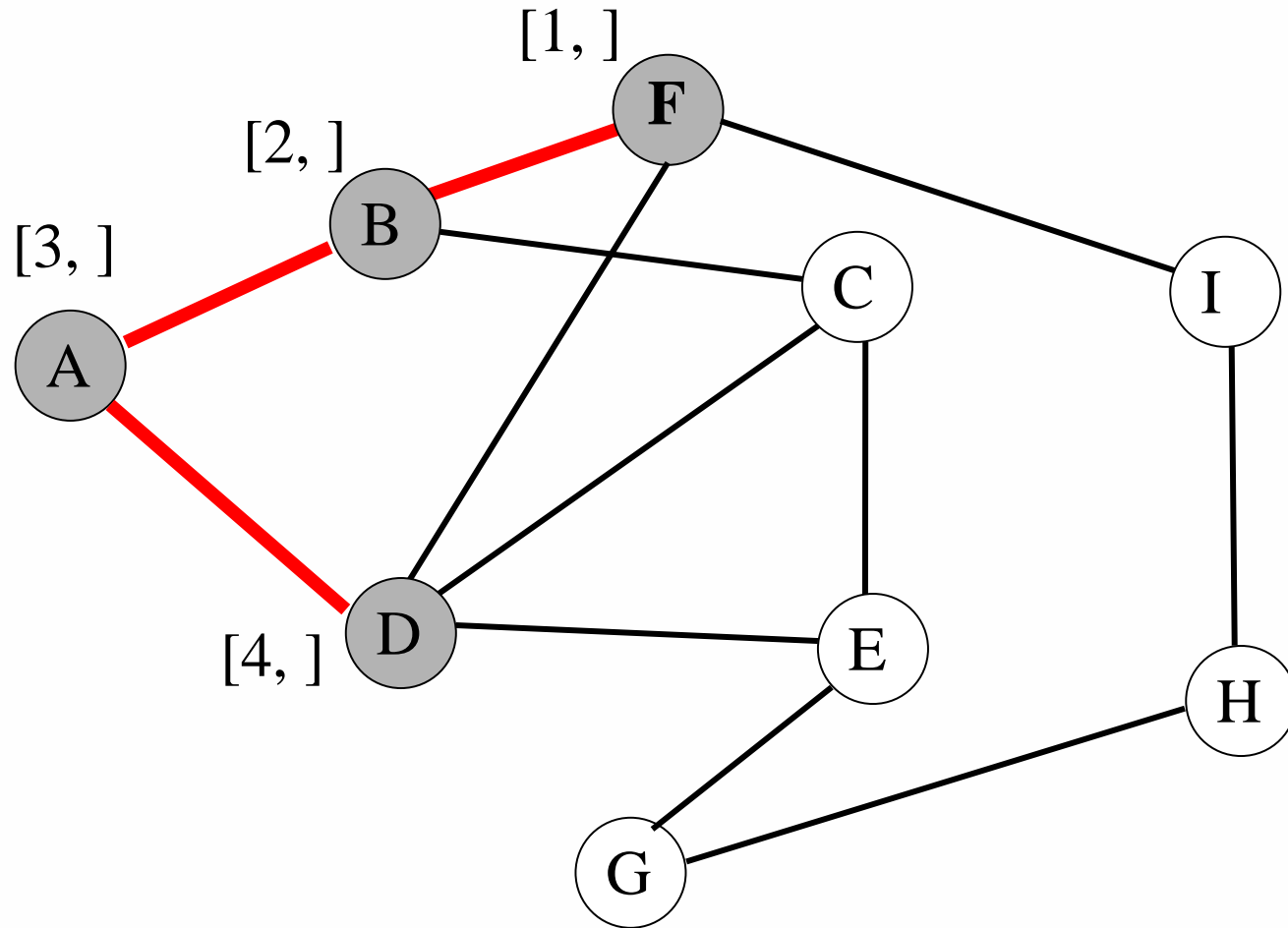
```
mark[s] = black
```

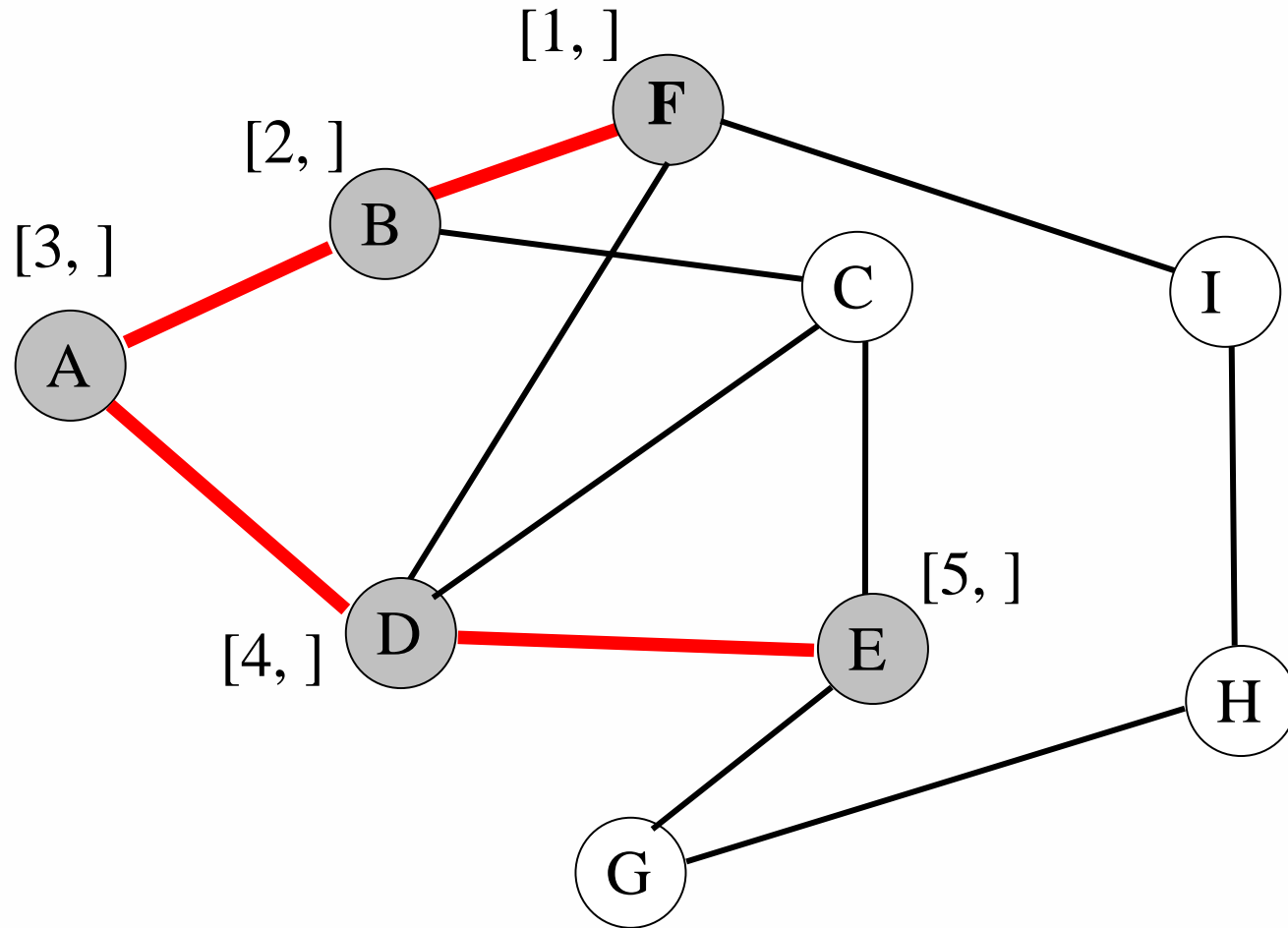
- ❑ $mark[s]$ è la marcatura del nodo s come nel BFS
- ❑ $d[s]$ e $f[s]$ sono rispettivamente l'istante in cui s viene raggiunto e in cui la sua visita termina
- ❑ $\pi[s]$ è il predecessore di s nell'albero di copertura
- ❑ Costo:
 - ▶ $O(|V|+|E|)$ liste
 - ▶ $O(|V|^2)$ matrice

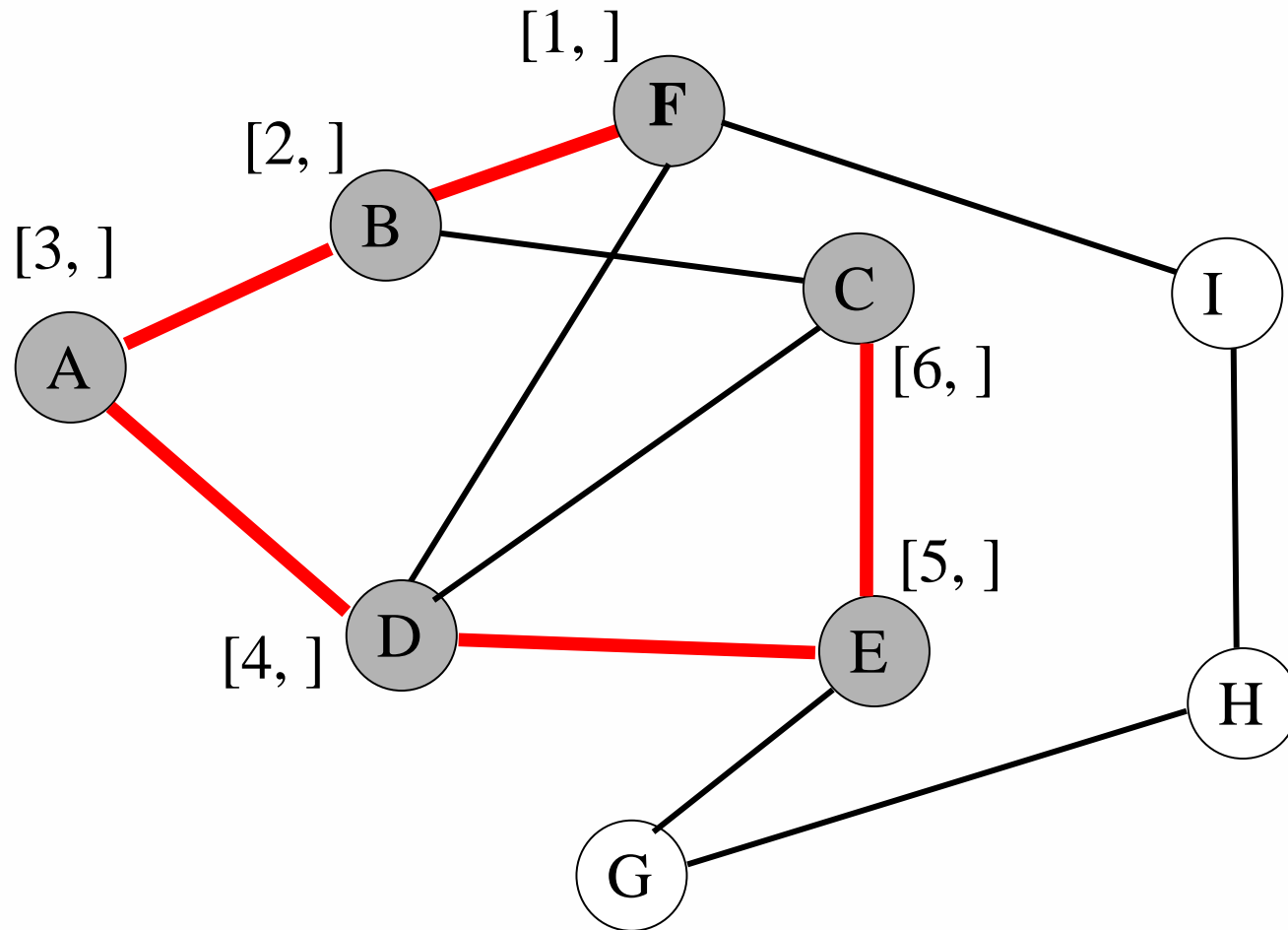


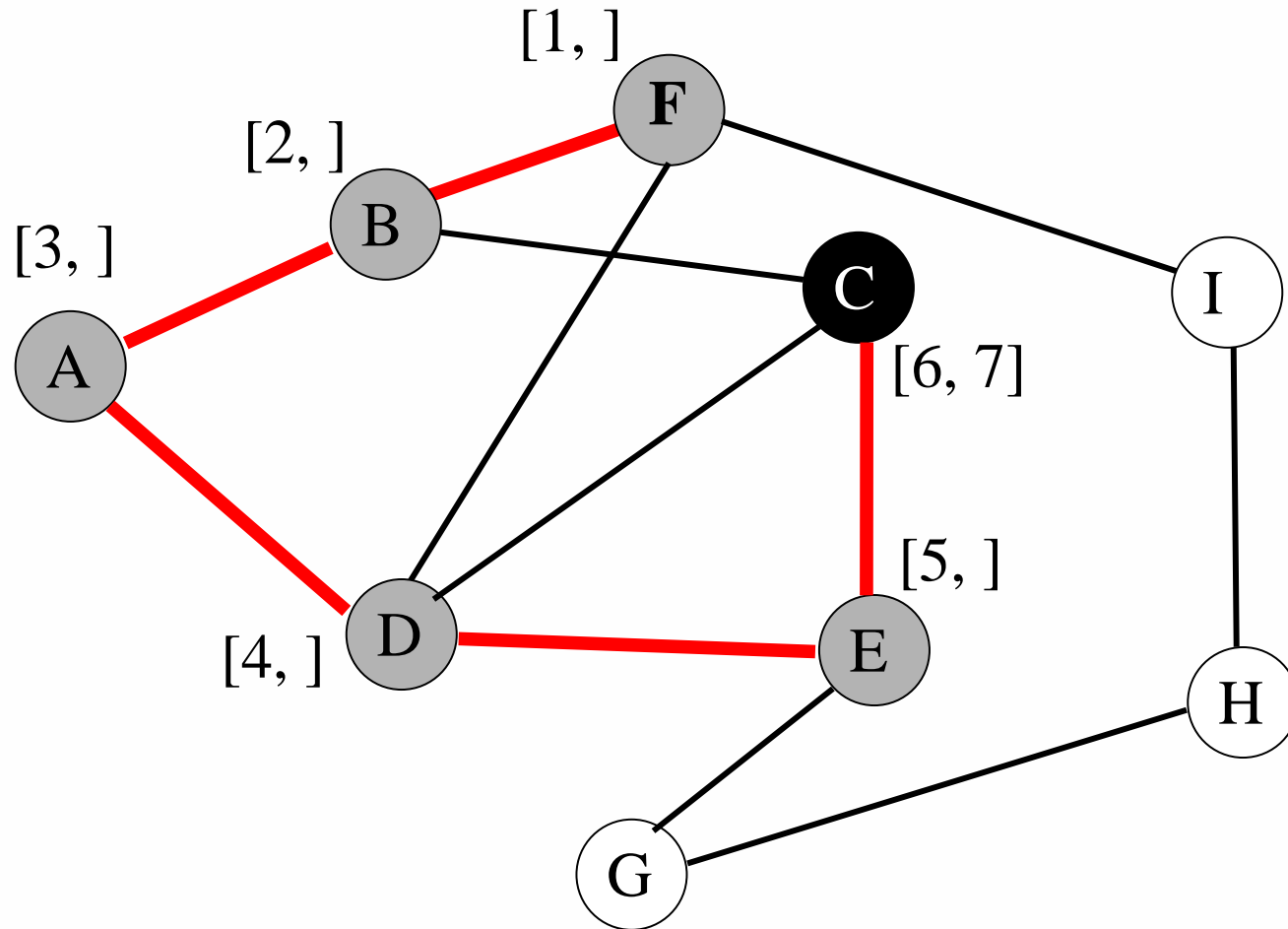


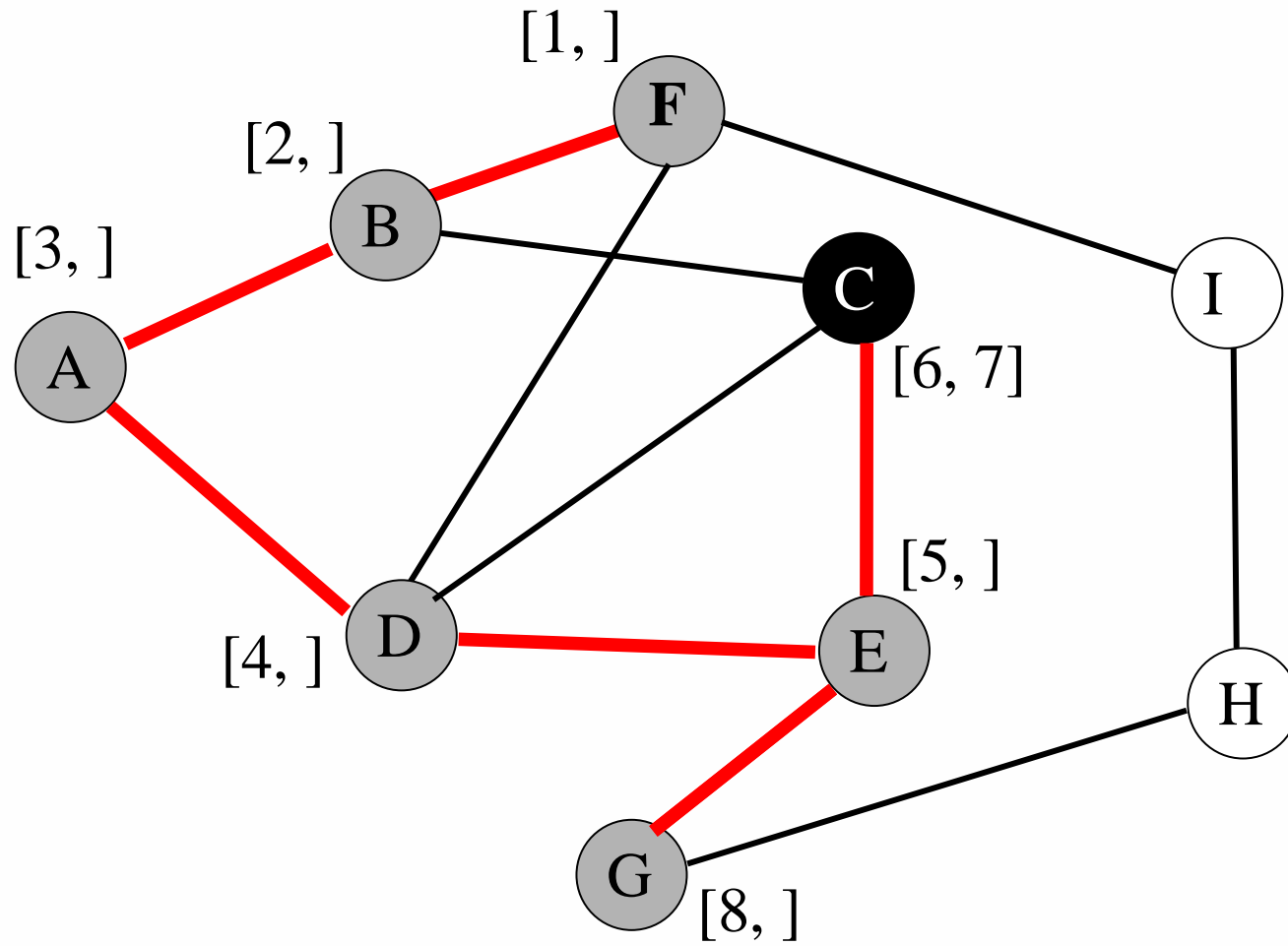


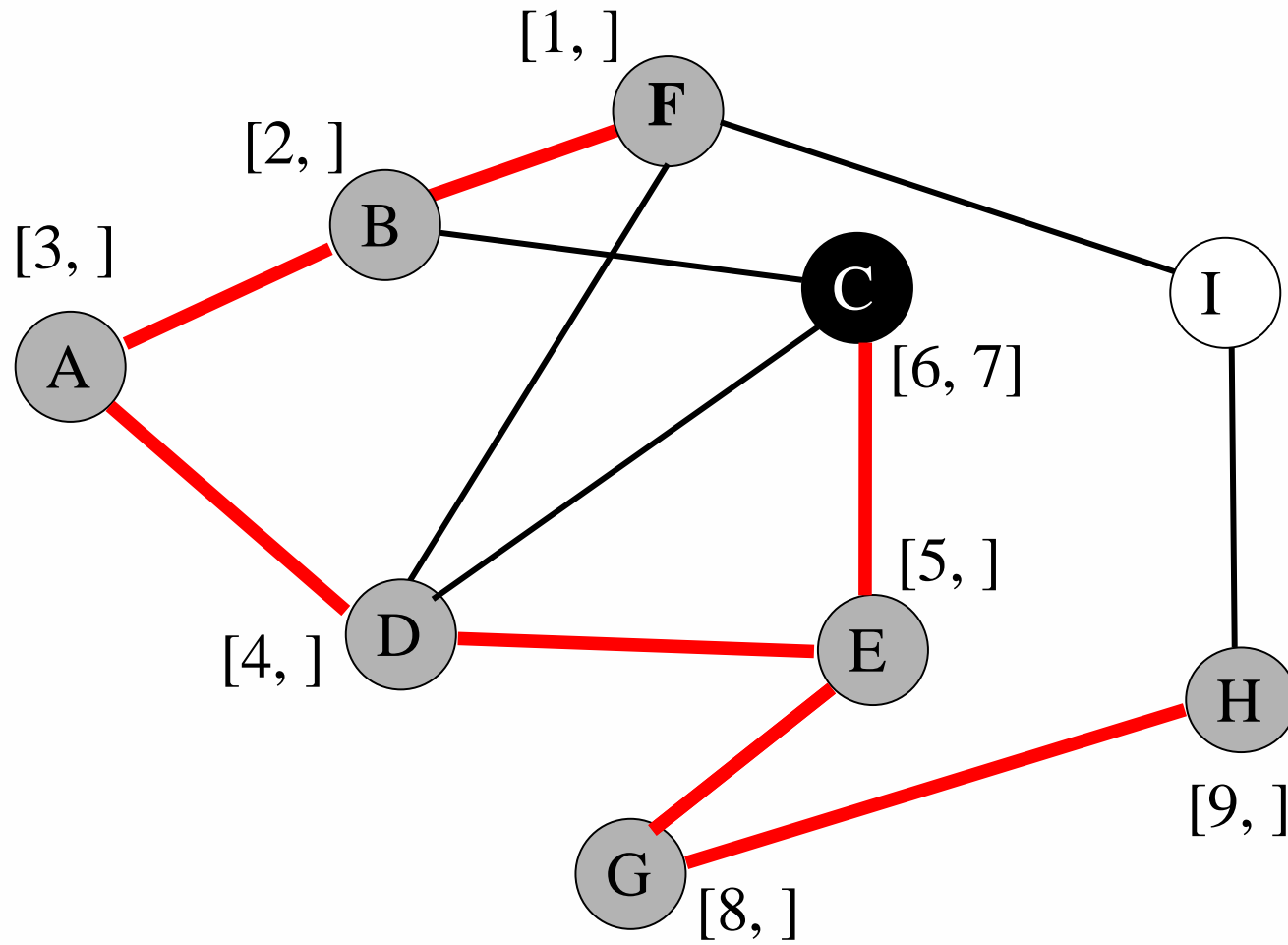


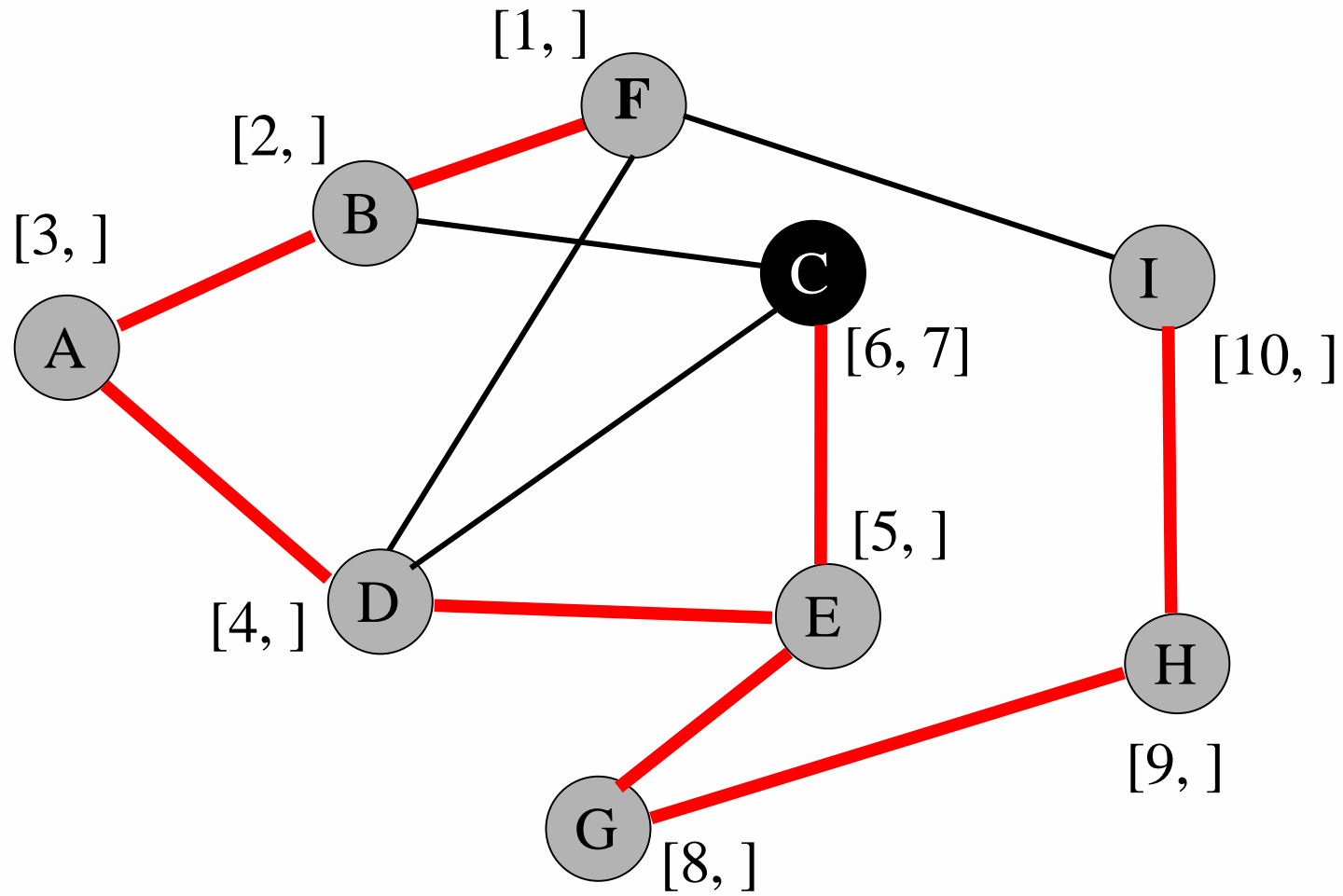


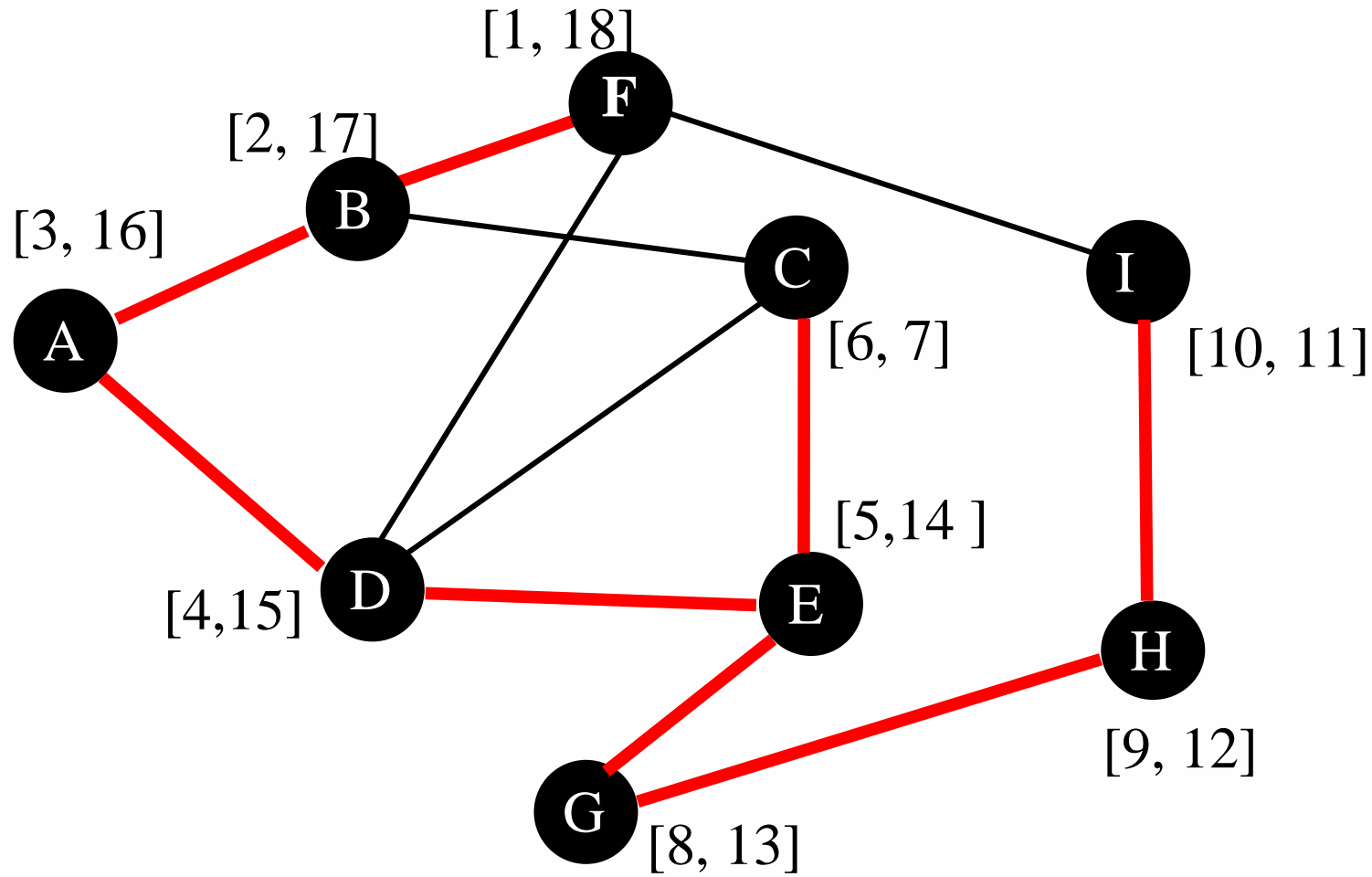












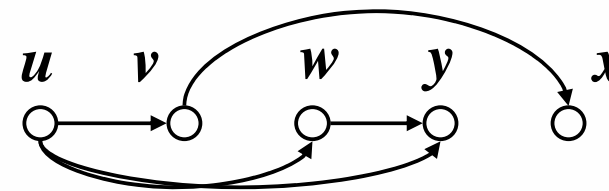
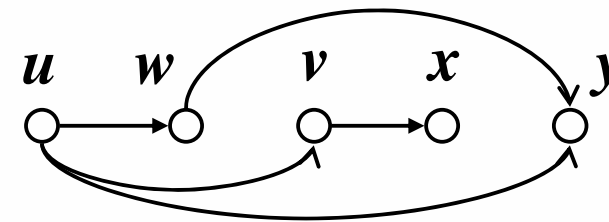
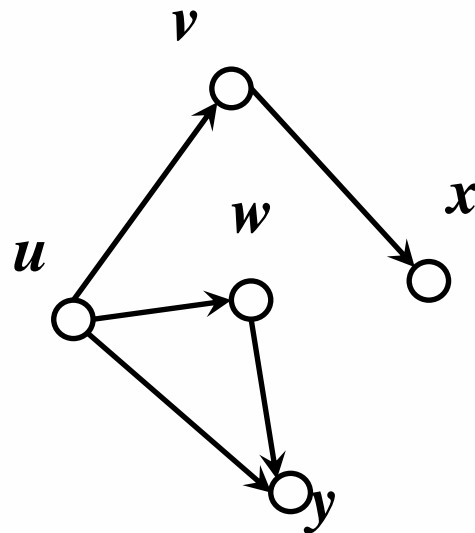
- In qualsiasi visita di profondità di un grafo $G=(V,E)$, per ogni coppia di vertici u,v , una sola delle seguenti condizioni è vera:
 - ▶ Gli intervalli $[d[u], f[u]]$ e $[d[v], f[v]]$ sono disgiunti
 $\Rightarrow u,v$ non sono discendenti l'uno dell'altro nell'albero DF
 - ▶ L'intervallo $[d[u], f[u]]$ è interamente contenuto in $[d[v], f[v]]$
 $\Rightarrow u$ è discendente di v in un albero DF
 - ▶ L'intervallo $[d[v], f[v]]$ è interamente contenuto in $[d[u], f[u]]$
 $\Rightarrow v$ è discendente di u in un albero DF
- In un albero DF di un grafo $G=(V,E)$, il vertice v è un discendente del vertice $u \iff$ al tempo $d[u]$, il vertice v può essere raggiunto da u lungo un cammino che è formato da soli nodi bianchi

Applicazioni

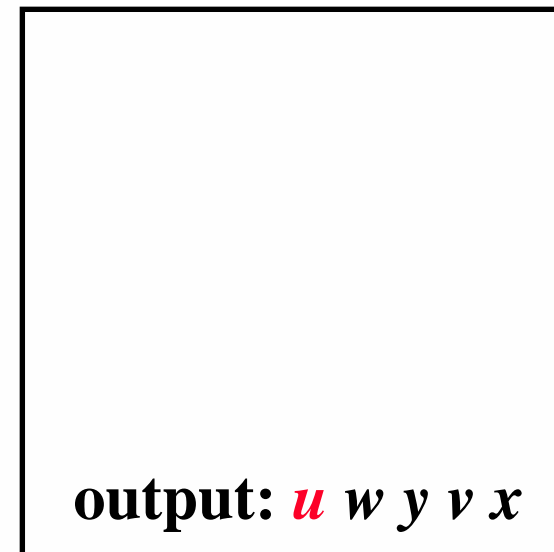
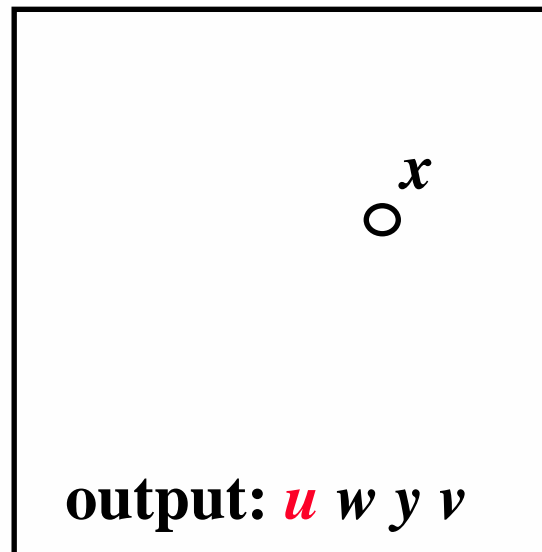
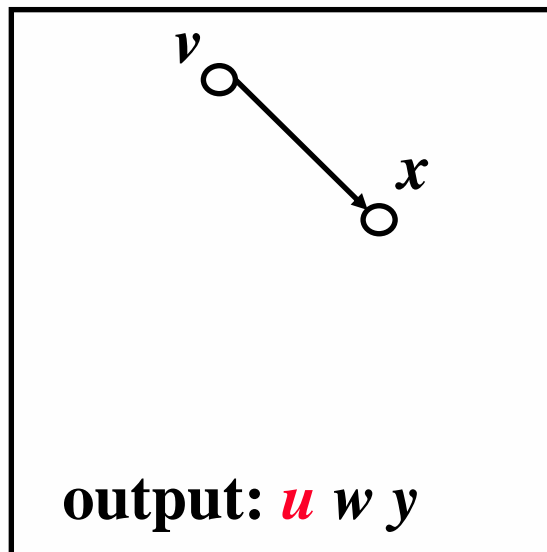
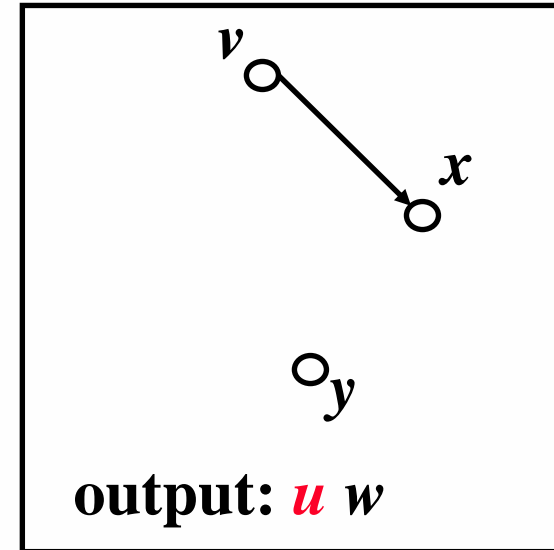
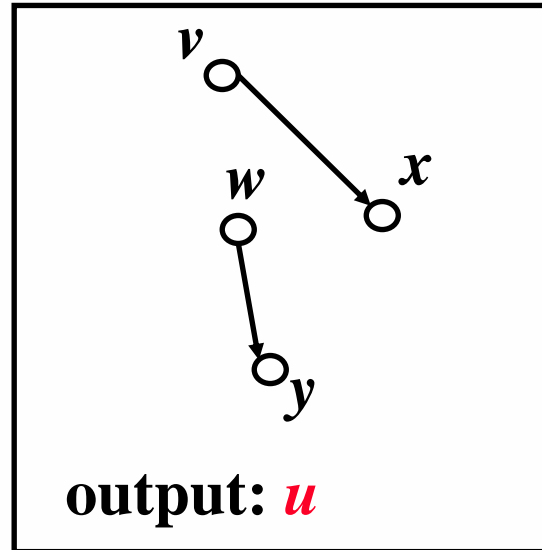
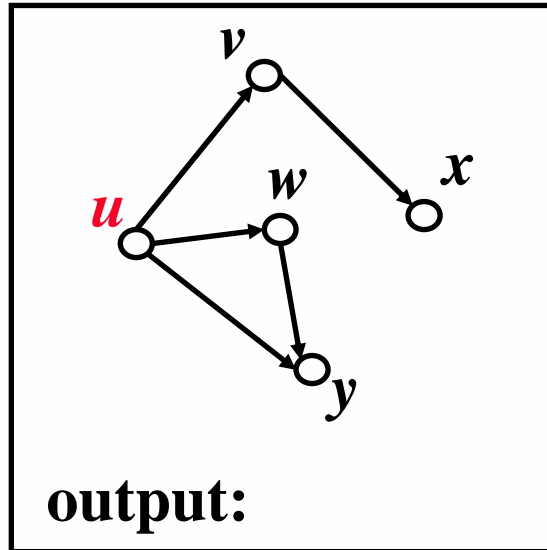
- ❑ La visita BFS può essere utilizzata per ottenere il percorso più breve fra due vertici (numero di archi)

```
print-path(G, s, v)  
  if v = s then  
    print s  
  else if  $\pi(v) = \text{nil}$  then  
    print "no path from s to v"  
  else  
    print-path(G, s,  $\pi(v)$ )  
    print v
```


- Dato un DAG G (direct acyclic graph), un *ordinamento topologico* su G è un ordinamento lineare dei suoi vertici tale per cui:
 - ▶ se G contiene l'arco (u,v) , allora u compare prima di v nell'ordinamento
 - ▶ Per transitività, ne consegue che se v è raggiungibile da u , allora u compare prima di v nell'ordinamento
 - ▶ Nota: possono esserci più ordinamenti topologici



- ❑ Problema:
 - ▶ Fornire un algoritmo che dato un grafo orientato aciclico, ritorni un ordinamento topologico
- ❑ Soluzione diretta
 - ▶ Trovare ogni vertice che non ha alcun arco incidente in ingresso
 - ▶ Stampare questo vertice e rimuoverlo, insieme ai suoi archi
 - ▶ Ripetere la procedura finché tutti vertici risultano rimossi.



- ❑ Problema:
 - ▶ Fornire un algoritmo che dato un grafo orientato aciclico, ritorni un ordinamento topologico
- ❑ Soluzione diretta
 - ▶ Trovare ogni vertice che non ha alcun arco incidente in ingresso
 - ▶ Stampare questo vertice e rimuoverlo, insieme ai suoi archi
 - ▶ Ripetere la procedura finché tutti vertici risultano rimossi.
- ❑ **Basato su DFS**

```
topological-sort(G)
```

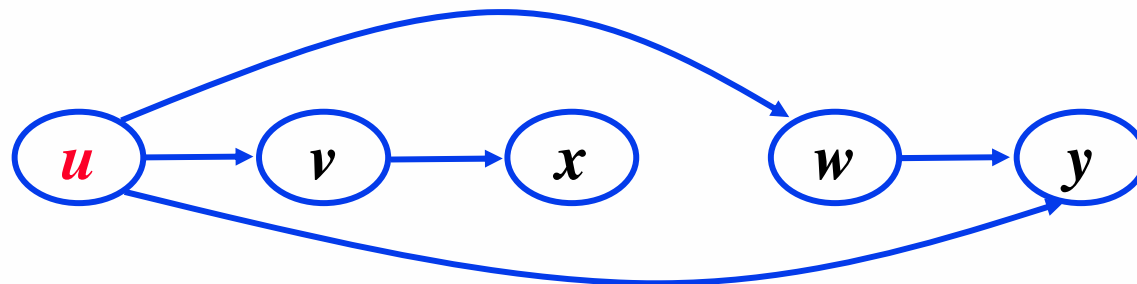
```
for each v in V[G] do
```

```
  if mark[v] == white
```

```
  then DFS(G, v)
```

Nella DFS() l'operazione di visita consiste nell'aggiungere il vertice alla testa di una lista l

```
return l
```



output: $u v x w y$

