Shortest Paths:
Bellman & Ford Algorithm
Dynamic Programming

5 Maggio 2023

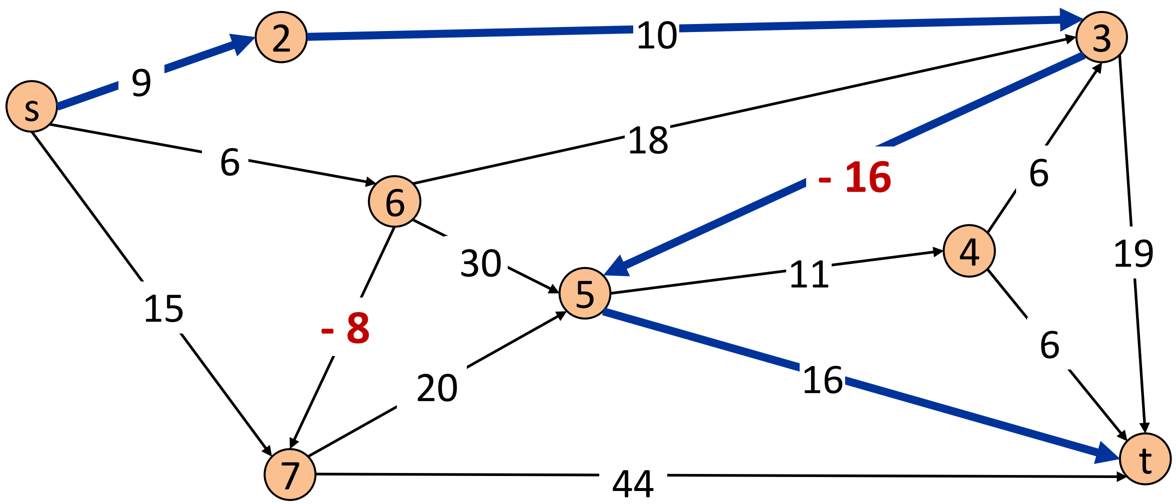
Shortest Paths

Shortest path problem. Given a directed graph G = (V, E), with edge costs c_{vw} find shortest (min cost) path from node s to node t.

allow negative costs

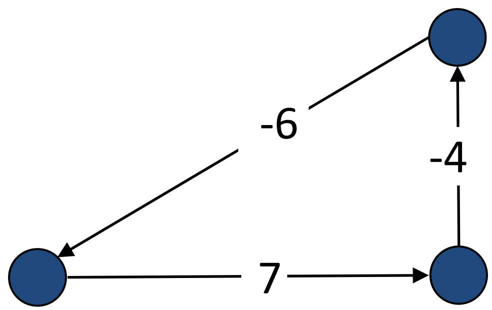
cost of path = sum of edge costs in path
min cost = ∞ if there is no path from s to t

Application. Nodes represent agents in a financial setting and c_{vw} is cost of transaction in which we buy from agent v and sell immediately to w.



Shortest Paths: Negative Cost Cycles

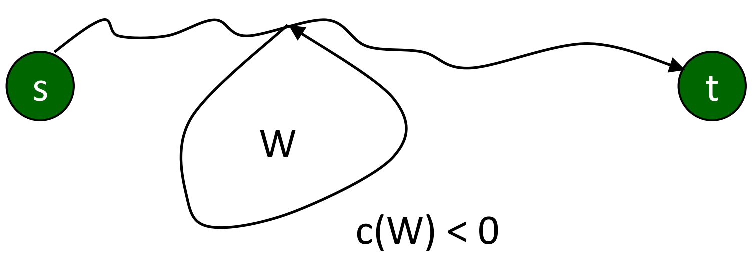
Negative cost cycle.



Observation. If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path.

If no negative cost cycle, there exists a shortest s-t path that is simple,

i.e. of length $\leq$ n$-$1.



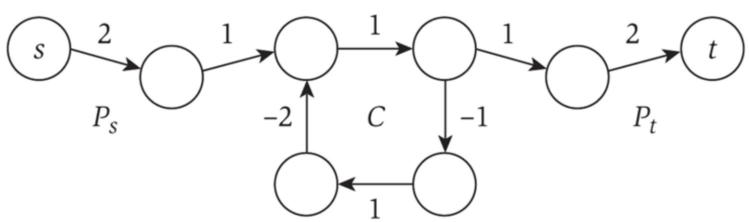
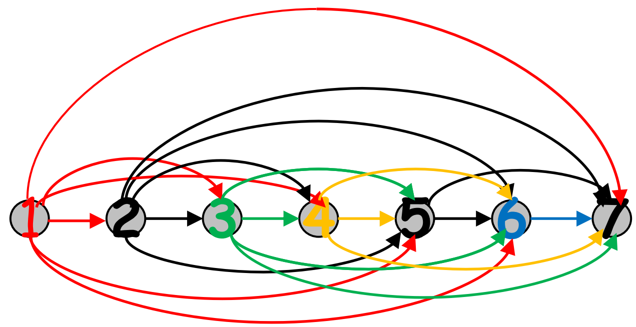


Figure 6.20 In this graph, one can find *s-t* paths of arbitrarily negative cost (by going around the cycle *C* many times).

PROBLEMA dei CAMMINI MINIMI su DAG completo



INPUT: un grafo diretto aciclico G=(V,E) con

V = {1,2,…,n}, E = {(i,j) t.c. i,j in V e i<j }

C(i,j) costo dell'arco (i,j) per ogni (i,j) in E

OUTPUT: un cammino da 1 a n di costo totale minimo

Dynamic Programming: symmetric view

Notation. **OPT(j)** = value of optimal solution to the problem consisting of vertices j, j+1, ..., n.

Case t, for any t=j+1, ..., n:

optimal substructure

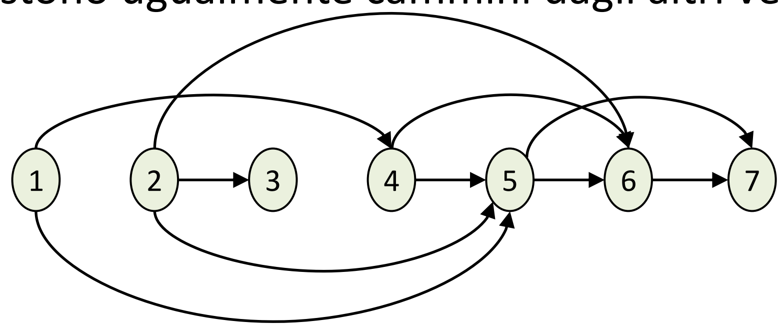
an optimal path OPT selects edge (j, t) and add an optimal solution to problem consisting of vertices t, t+1, …, n

$$OPT(j) = \begin{cases} 0 & \text{if } j = n \\ \min\limits_{t = j+1, \ldots, n} \{C(j,t) + OPT(t)\} & \text{otherwise} \end{cases}$$

DAG non “completo”

Se invece il DAG non fosse “completo”, cioè non fossero necessariamente presenti **tutti** gli archi (v,w) per ogni v, w in V, per qualche v potrebbe non esistere un cammino da v a t.

Nell’esempio: non esistono archi uscenti da 3, e quindi non esistono cammini da 3 a 7. Esistono ugualmente cammini dagli altri vertici a 7.



OPT[v] = min costo di un cammino da v a n,
∞ se non esistono cammini da v ad n

OPT[n] = 0

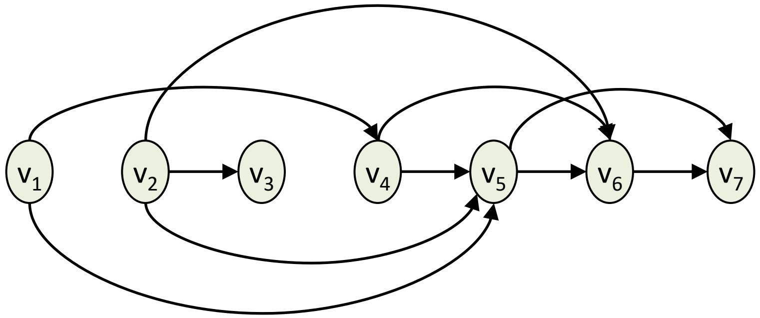
$$OPT[v] = \min \{ \infty, \min_{(v,w)\in E} \{ OPT[w] + C(v,w)\} \} \text{ se } v \neq n$$

Soluzione = OPT[1]

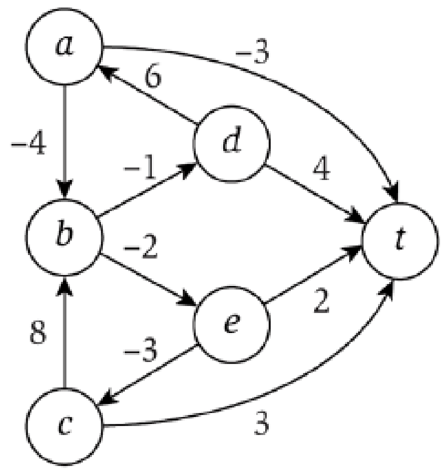
E’ un problema di cammini minimi, ma su DAG, ovvero su grafo diretto con un **ordinamento topologico** quindi, se effettuo i calcoli secondo l’ordinamento dei vertici dal più grande al più piccolo, quando calcolo OPT[v], OPT[w] è già stato calcolato

```
Shortest-Path(G)
M[n]=0
for v=n-1 downto 1
        M[v]= ∞
        foreach (v,w)∈E
        if M[w]+C(v,w)< M[v]
            then M[v]= M[w]+C(v,w)
return M[1]
```

Grafo generico



E' un DAG, ovvero un grafo diretto con un **ordinamento topologico**.



Non è un DAG, ovvero **non** ha un **ordinamento topologico**.
Ho il ciclo c-b-e-c.

Se calcolassi

M[c] = min{ C(c,b)+M[b] , C(c,t)+M[t] }

in che ordine dovrei calcolare M[c], M[b], M[e] per essere sicuri di richiamare sempre valori già calcolati?

Bellman e Ford usano un altro ordine, l'ordine di lunghezza dei cammini

Shortest Paths: Dynamic Programming

Def. Per i, intero positivo, v, vertice

OPT(i, v) = costo minimo di un cammino da v a t che usa al più i archi (ovvero di lunghezza minore o uguale a i);

∞ se non esiste un cammino da v a t.

Caso 1: se esiste un cammino da v a t

Caso 2: se non esiste un cammino da v in t,OPT(i,v)= ∞.

Shortest Paths: Dynamic Programming

Def. i, intero positivo, v, vertice

OPT(i, v) = costo minimo di un cammino da v a t che usa al più i archi (ovvero di lunghezza minore o uguale a i),

∞ se non esiste un cammino da v a t.

Caso 1: se esiste un cammino da v a t, il cammino ottimale P avrà almeno un arco; allora se (v,w) è il suo primo arco, P è composto da (v,w) seguito da un cammino ottimale da w a t con al più i-1 archi

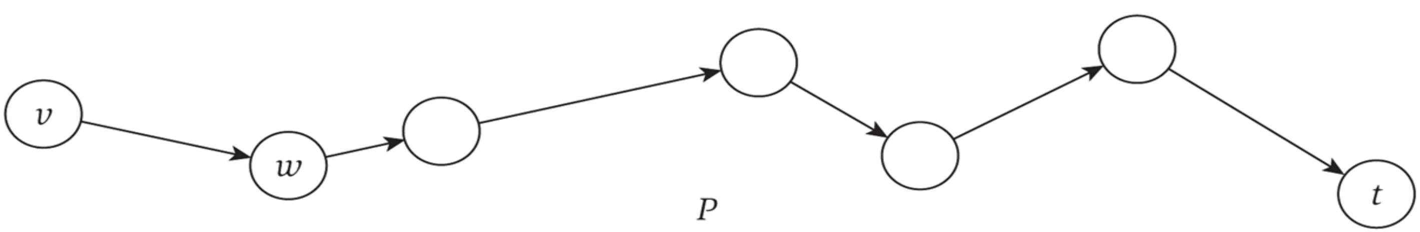


Figure 6.22 The minimum-cost path P from v to t using at most i edges.

Shortest Paths: Dynamic Programming

Def. i, intero positivo, v, vertice

OPT(i, v) = costo minimo di un cammino da v a t che usa al più i archi (ovvero di lunghezza minore o uguale a i),

∞ se non esiste un cammino da v a t.

Caso 2: se non esiste un cammino v-t, OPT(i,v)= ∞.

Ma come possiamo ottenerlo da valori precedenti? Un modo è definendolo così:

OPT(i,v) = OPT(i-1,v), essendo OPT(0,v)= ∞.

Infine, da osservazioni precedenti, se non ci sono cicli di costo negativo, allora

OPT(n-1, v) = costo minimo di un cammino da v a t.

Shortest Paths: Dynamic Programming

Def. OPT(i, v) = costo minimo di un cammino da v a t che usa al più i archi, (ovvero di lunghezza minore o uguale a i), ∞ se non esiste un cammino da v a t.

Caso 1: se esiste un cammino da v a t, il cammino ottimale P avrà almeno un arco;

allora se (v,w) è il suo primo arco, P è composto da (v,w) seguito da un cammino ottimale da w a t con al più i-1 archi

Caso 2: se non esiste un cammino da v a t,OPT(i,v)= ∞.

OPT(I, v) = OPT(i-1,v), essendo OPT(0,v)= ∞.

Infine, da osservazioni precedenti, se non ci sono cicli di costo negativo, allora
OPT(n-1, v) = costo minimo di un cammino da v a t.

$$OPT(i, v) = \begin{cases} 0 & \text{se } v = t \\ \min\left\{ OPT(i-1, v), \underbrace{\min_{(v,w)\in E} \{ OPT(i-1, w) + c_{vw} \}}_{\text{Se esistono (v,w) in E}} \right\} & \text{altrimenti} \\ \infty & \text{se } i = 0, v \neq t \end{cases}$$

Shortest Paths: Implementation

$$OPT(i, v) = \begin{cases} 0 & \text{se } v = t \\ \min\left\{ OPT(i-1, v), \ \min\limits_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \} \right\} & \text{altrimenti} \\ \infty & \text{se } i = 0, v \neq t \end{cases}$$

Se esistono (v,w) in E

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] ← ∞
   M[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
         M[i, v] ← M[i-1, v]
         foreach edge (v, w) ∈ E
         M[i, v] ← min { M[i, v], M[i-1, w] + c_vw }
}
```

Shortest Paths: Implementation

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] ← ∞
   M[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
         M[i, v] ← M[i-1, v]
         foreach edge (v, w) ∈ E
         M[i, v] ← min { M[i, v], M[i-1, w] + c_vw }
}
```

Cosa succederebbe invertendo i 2 for?

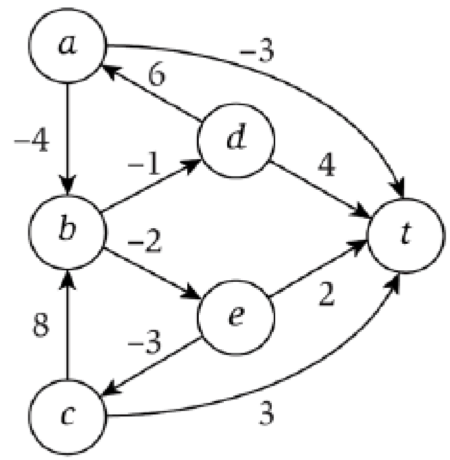
Analysis. $\Theta(n^2)$ space, $\Theta(mn)$ time

(la somma del numero di archi uscenti da v, per ogni v, è m)

Finding the shortest paths.

Maintain a "successor" for each table entry.

Shortest Paths: Esempio



(a)

$$OPT(i,v) = \begin{cases} 0 & \text{se } v = t \\ \min\left\{ OPT(i-1,\ v),\ \min\limits_{(v,w)\in E}\{ OPT(i-1,\ w) + c_{vw} \} \right\} & \text{altrimenti} \\ \infty & \text{se } i = 0, v \neq t \end{cases}$$

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	−3	−3	−4	−6	−6
b	∞	∞	0	−2	−2	−2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

(b)

Figure 6.23 For the directed graph in (a), the Shortest-Path Algorithm constructs the dynamic programming table in (b).

$OPT(5,a) = \min(\ OPT(4,a),\ \min(OPT(4,t)+c_{at}, OPT(4,b)+c_{ab}\ )$
$= \min(\ -6,\ \min(0-3,-2-4)\ )$

$OPT(4,a) = \min(\ OPT(3,a),\ \min(OPT(3,t)+c_{at}, OPT(3,b)+c_{ab}\ )$ a-b
$= \min(\ -4,\ \min(0-3,-2-4)\ )$

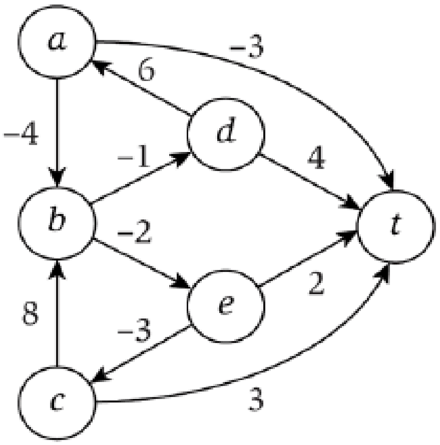
$OPT(3,b) = \min(\ OPT(2,b),\ \min(OPT(2,e)+c_{be}, OPT(2,d)+c_{bd}\ )$ b-e
$= \min(\ 0,\ \min(0-2,3-1)\ )$

$OPT(2,e) = \min(\ OPT(1,e),\ \min(OPT(1,c)+c_{ec}, OPT(1,t)+c_{et}\ )$ e-c
$= \min(\ 2,\ \min(3-3,0+2)\ )$

$OPT(1,c) = \min(OPT(0,c),\ \min(OPT(0,b)+c_{cb},\ OPT(0,t)+c_{ct}\ )$ c-t
$= \min(\ \infty,\ \min(\infty+8,0+3)\ )$

a - b - e - c - t
-4 -2 -3 +3 = -6

Shortest Paths: Esempio



(a)

Trovare il cammino più corto.
Qual'è l'informazione che possiamo ricordare?

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	−3	−3	−4	−6	−6
b	∞	∞	0	−2	−2	−2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

(b)

Figure 6.23 For the directed graph in (a), the Shortest-Path Algorithm constructs the dynamic programming table in (b).

$$OPT(5,a) = \min(\ OPT(4,a),\ \min(OPT(4,t)+c_{at}, OPT(4,b)+c_{ab}\ )$$
$$= \min(\ -6,\ \min(0-3,-2-4)\ )$$

$$OPT(4,a) = \min(\ OPT(3,a),\ \min(OPT(3,t)+c_{at}, OPT(3,b)+c_{ab}\ )$$
$$= \min(\ -4,\ \min(0-3,-2-4)\ )$$

a-b

$$OPT(3,b) = \min(\ OPT(2,b),\ \min(OPT(2,e)+c_{be}, OPT(2,d)+c_{bd}\ )$$
$$= \min(\ 0,\ \min(0-2,3-1)\ )$$

b-e

$$OPT(2,e) = \min(\ OPT(1,e),\ \min(OPT(1,c)+c_{ec}, OPT(1,t)+c_{et}\ )$$
$$= \min(\ 2,\ \min(3-3,0+2)\ )$$

e-c

$$OPT(1,c) = \min(OPT(0,c), \min(OPT(0,b)+c_{cb}, OPT(0,t)+c_{ct}\ )$$
$$= \min(\ \infty,\ \min(\infty+8,0+3)\ )$$

c-t

a - b - e - c - t

-4 -2 -3 +3 = -6

Sia G=(V,E) un grafo con V={1,2,3,4,5,6,7} e E={(1,2), (1,3), (1,4), (2,5), (2,6), (3,6), (4,5), (5,7), (6,7)}. L'albero della visita in profondità DFS del grafo G=(V,E), a partire dal vertice 1, in cui i vertici adiacenti sono analizzati in ordine crescente, ha profondità

(svolto)

A. 3 B. 4 C. 5 D. Nessuna delle risposte precedenti

Si consideri il grafo (non orientato) G=(V,E) con V={1,2,3,4,5,6} e E={(1,2), (1,3), (2,3), (4,5), (5,6)}.

(svolto)

A. G è un grafo connesso
B. G è un grafo fortemente connesso
C. G non è un grafo connesso e ha due componenti connesse
D. Nessuna delle risposte precedenti

In un grafo connesso con n vertici ed m archi:

A. $\log m = \Theta(\log n)$

B. $\log m = O(\log n)$, ma non $\log m = \Theta(\log n)$

C. $\log n = O(\log m)$, ma non $\log n = \Theta(\log m)$

D. Nessuna delle risposte precedenti

Sia $G=(V,E)$ un grafo con $V=\{1,2,3,4,5,6,7\}$ e $E=\{(1,2), (1,3), (1,4), (2,5), (2,6), (3,6), (4,5), (5,7), (6,7)\}$. L'albero della visita in ampiezza **BFS** del grafo $G=(V,E)$, a partire dal vertice 1, in cui i vertici adiacenti sono analizzati in ordine crescente, ha profondità

A. 3 B. 4 C. 5 D. Nessuna delle risposte precedenti

Sia G=(V,E) un grafo con V={1,2,3,4,5,6,7} e E={(1,2), (1,4), (2,4), (2,5), (3,6), (4,5)}. In seguito alla visita in ampiezza **BFS** del grafo G=(V,E), a partire dal vertice 1, quanti saranno i vertici scoperti (*Discovered*) (compreso 1)?

A. 6 B. 4 C. 5 D. Nessuna delle risposte precedenti

Quali di queste affermazioni è vera?

A. Un grafo diretto è aciclico se e solo se ha un ordinamento topologico
B. Un grafo diretto aciclico può avere un ordinamento topologico, ma può anche non averlo
C. Un grafo diretto con un ordinamento topologico può essere aciclico oppure no
D. Nessuna delle risposte precedenti

Se G=(V,E) è un grafo rappresentato con una matrice di adiacenza, verificare se (u,v) è un arco di E, richiede tempo

A. Θ (1)   B. Θ (n)   C. Θ (n^2)   D. Nessuna delle risposte precedenti

Un ordinamento topologico per il grafo diretto G=(V,E) con V={u, v, x, y, z}, E={(u,x), (v,x), (v,y), (v,u), (x,y), (y,z)} è:

A. v, u, z, y, x

B. v, u, x, y, z

C. G non ha un ordinamento topologico

D. Nessuna delle risposte precedenti

Sia G=(V,E), il grafo in cui V={1,2,3,4,5,6,7} ed E={(1,2), (1,3), (1,4), (2,5), (2,6), (3,6), (3,7), (4,6), (4,7)}. Allora

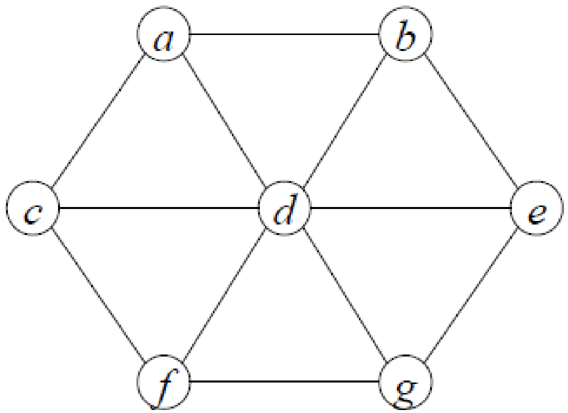
A. G non è bipartito perché non ha cicli di lunghezza dispari

B. G è bipartito perché contiene cicli di lunghezza dispari

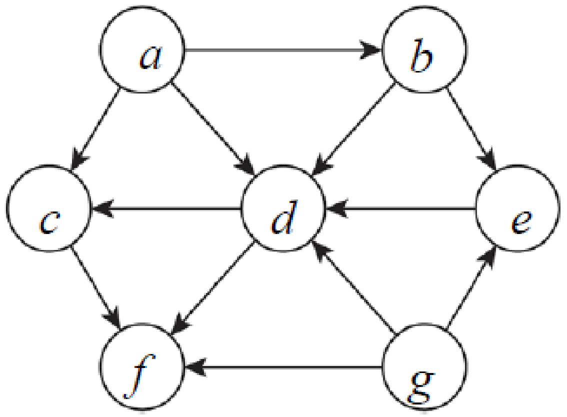
C. G è bipartito perché contiene cicli di lunghezza pari

D. Nessuna delle risposte precedenti

1)Descrivere **verbalmente** un algoritmo per decidere se un grafo **non orientato** è aciclico o no.
2) **Eseguire** l'algoritmo descritto sul grafo G1 in basso.
3)Descrivere **verbalmente** un algoritmo per decidere se un grafo **orientato** è aciclico o no.
4)Eseguire l'algoritmo descritto sul grafo G2 in basso.



G1



G2