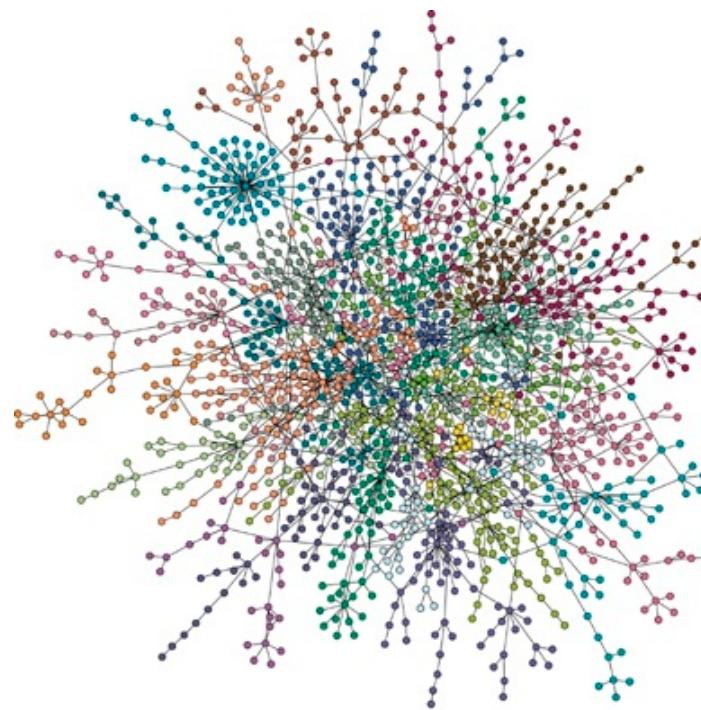


Grafi - Rappresentazione e visita

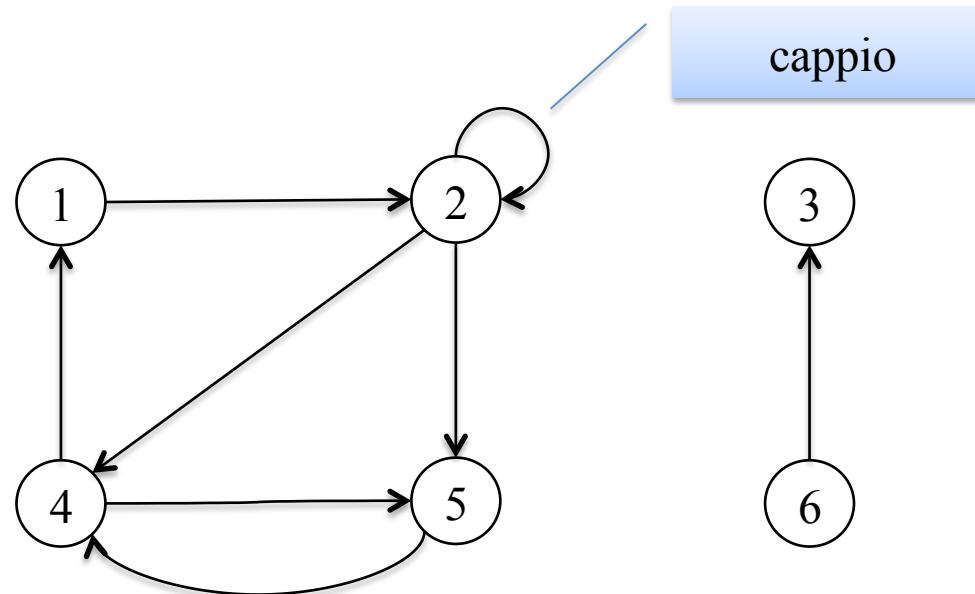


Algoritmi e strutture dati
Lezione 15, a.a. 2016-17
Ugo de'Liguoro, Andras Horvath

Grafo orientato

$G = (V, E)$ dove

- V insieme (finito) di *vertici*
- $E \subseteq V \times V$ insieme di *archi*

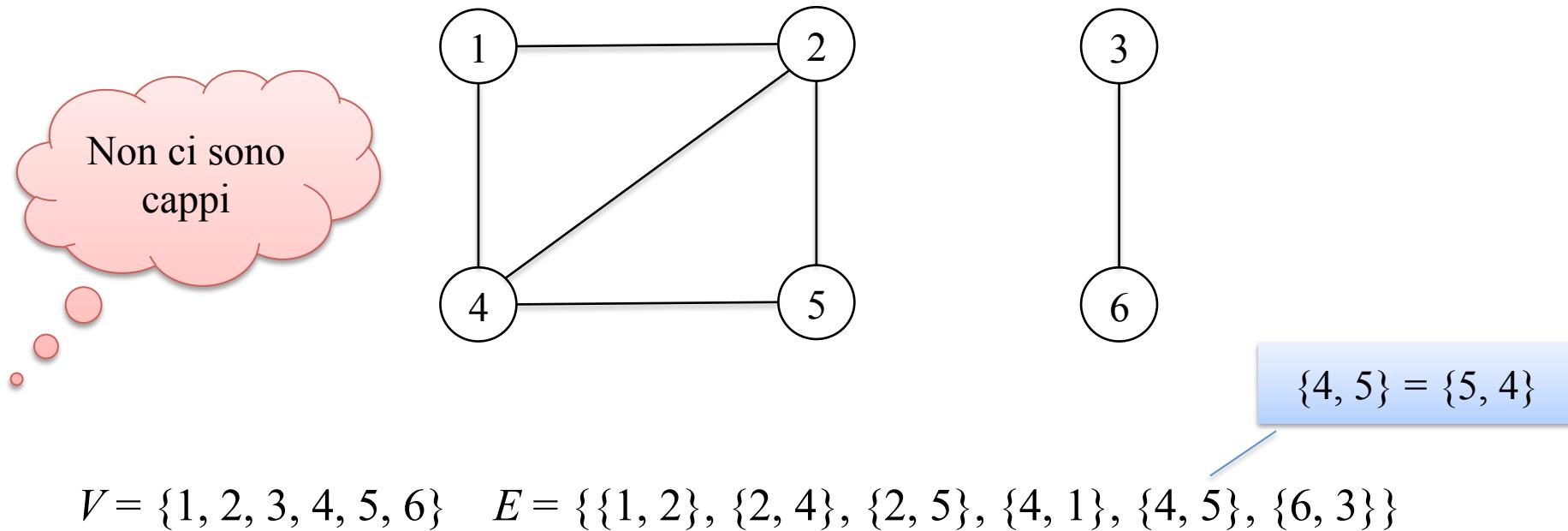


$$V = \{1, 2, 3, 4, 5, 6\} \quad E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$$

Grafo non orientato

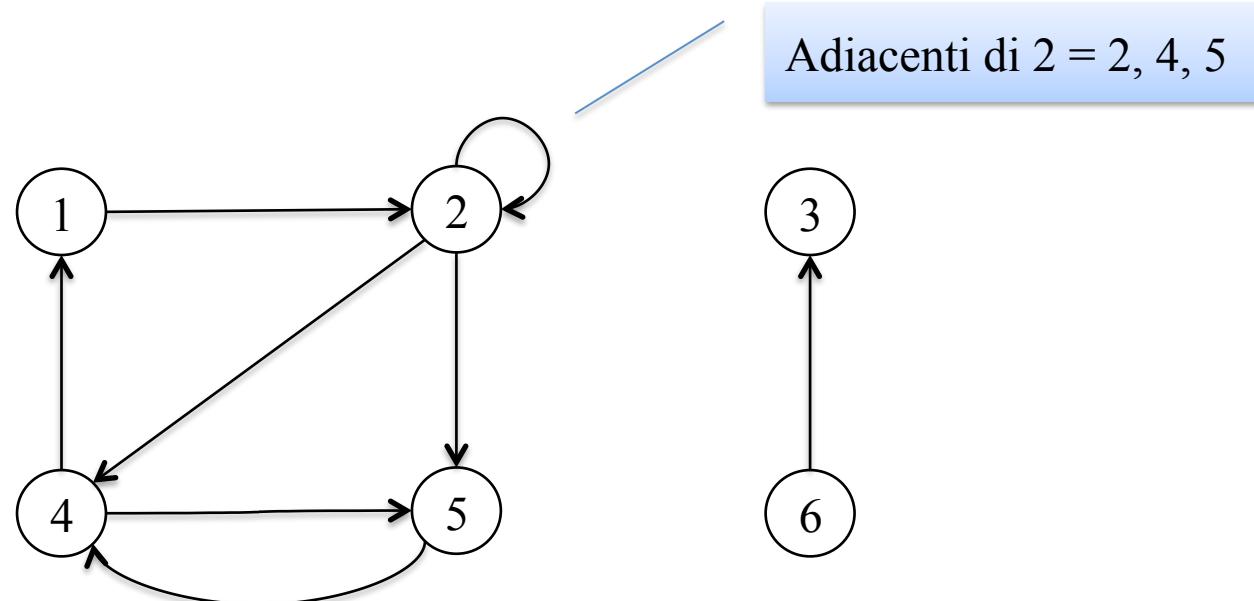
$G = (V, E)$ dove

- V insieme (finito) di *vertici*
- $E \subseteq \{\{u, v\} \mid u, v \in V \& u \neq v\}$ insieme di *archi*



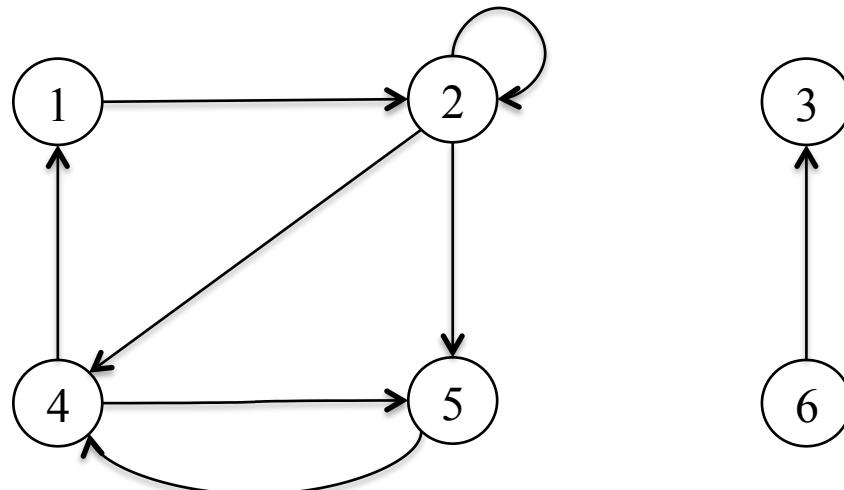
Terminologia

- (u, v) arco *incidente* in u, v - analogamente per $\{u, v\}$
- se $(u, v) \in E$ allora v è *adiacente* ad u
- se $\{u, v\} \in E$ allora v è *adiacente* ad u e viceversa



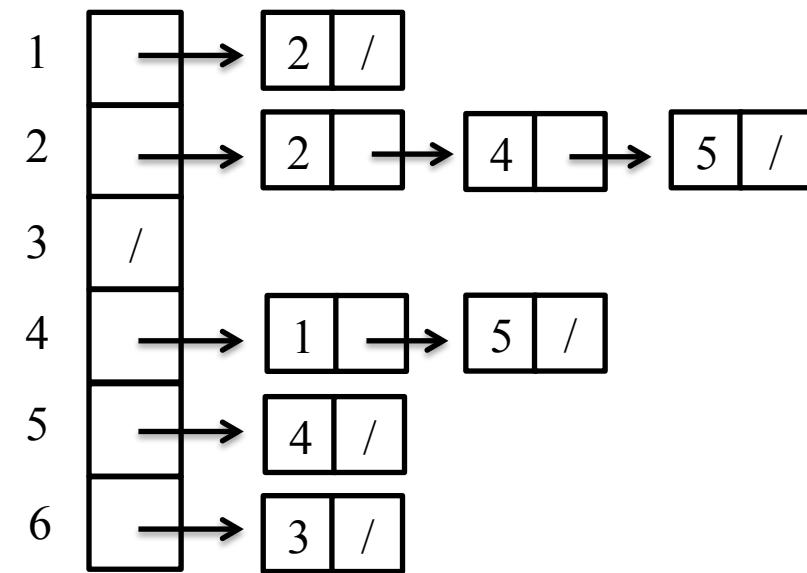
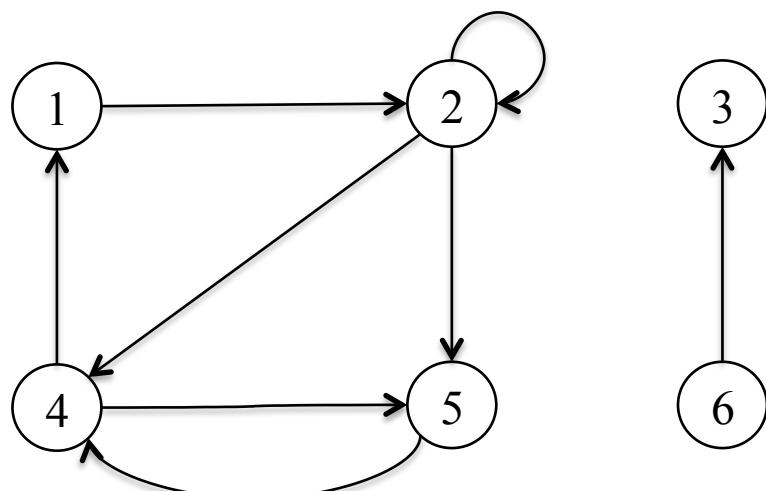
Matrice di adiacenza (G orientato)

$$M_{i,j} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{altrimenti} \end{cases}$$



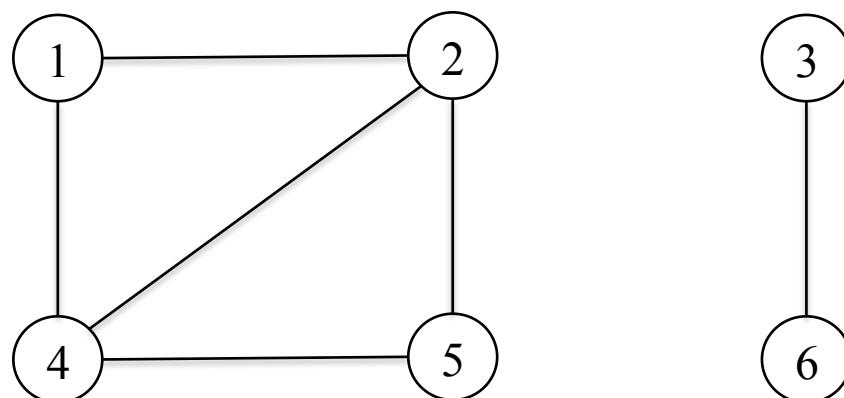
	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

Liste di adiacenza



Matrice di adiacenza (G non orientato)

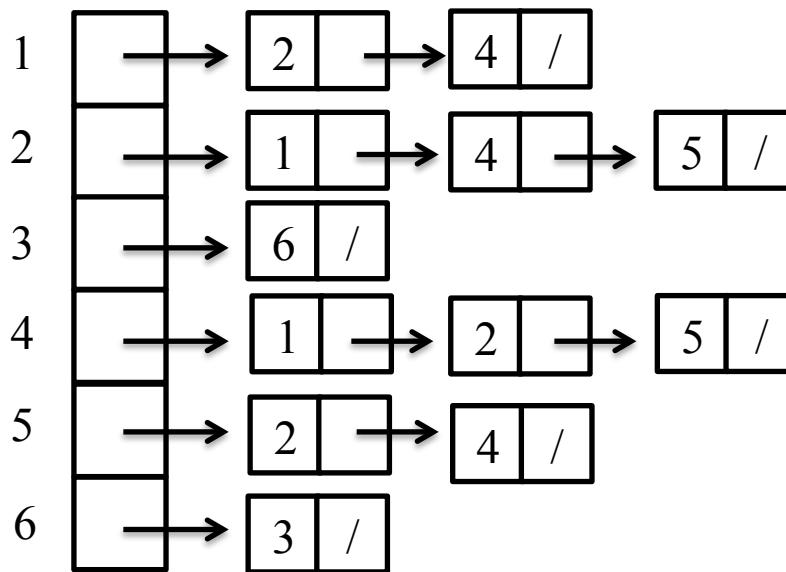
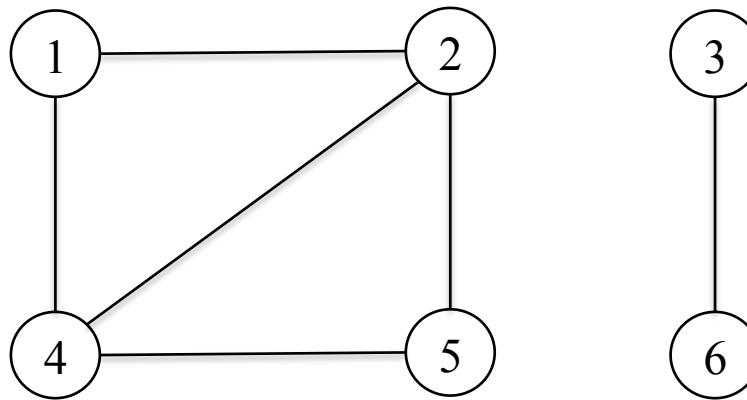
$$M_{i,j} = \begin{cases} 1 & \text{se } \{i,j\} \in E \\ 0 & \text{altrimenti} \end{cases}$$



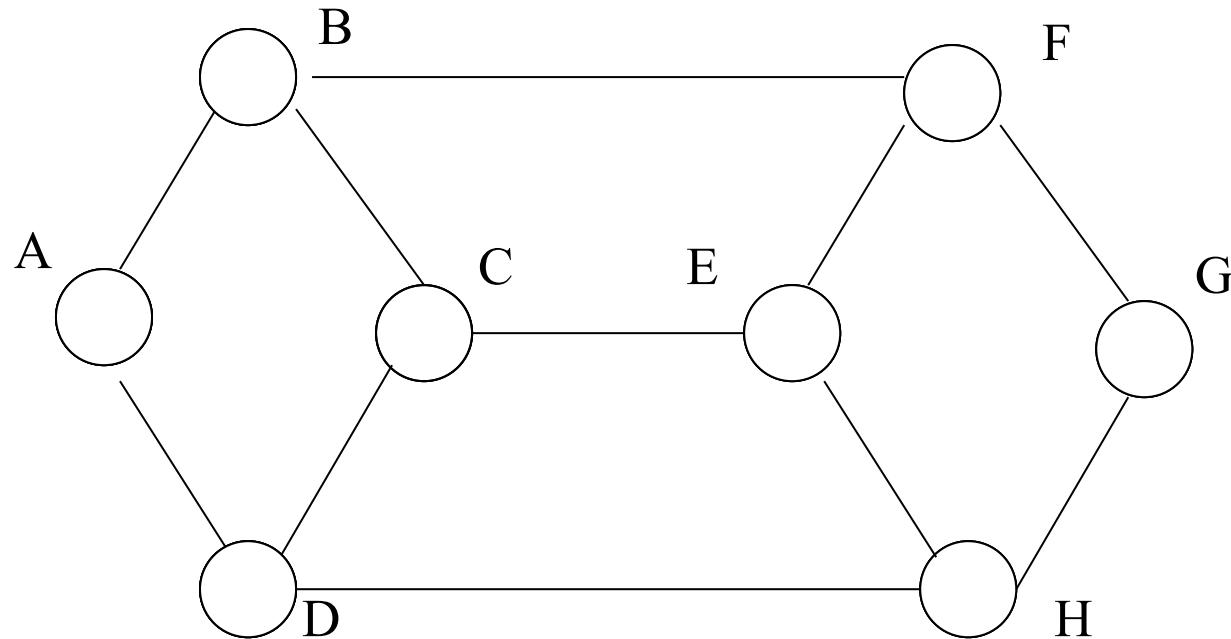
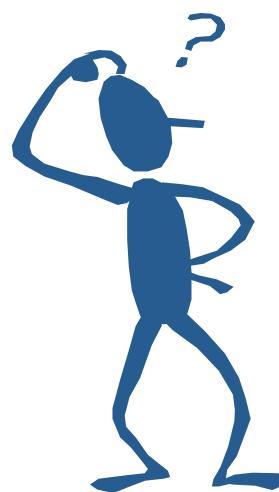
M è simmetrica con 0
sulla diag. principale

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	1	1	0
3	0	0	0	0	0	1
4	1	1	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

Liste di adiacenza

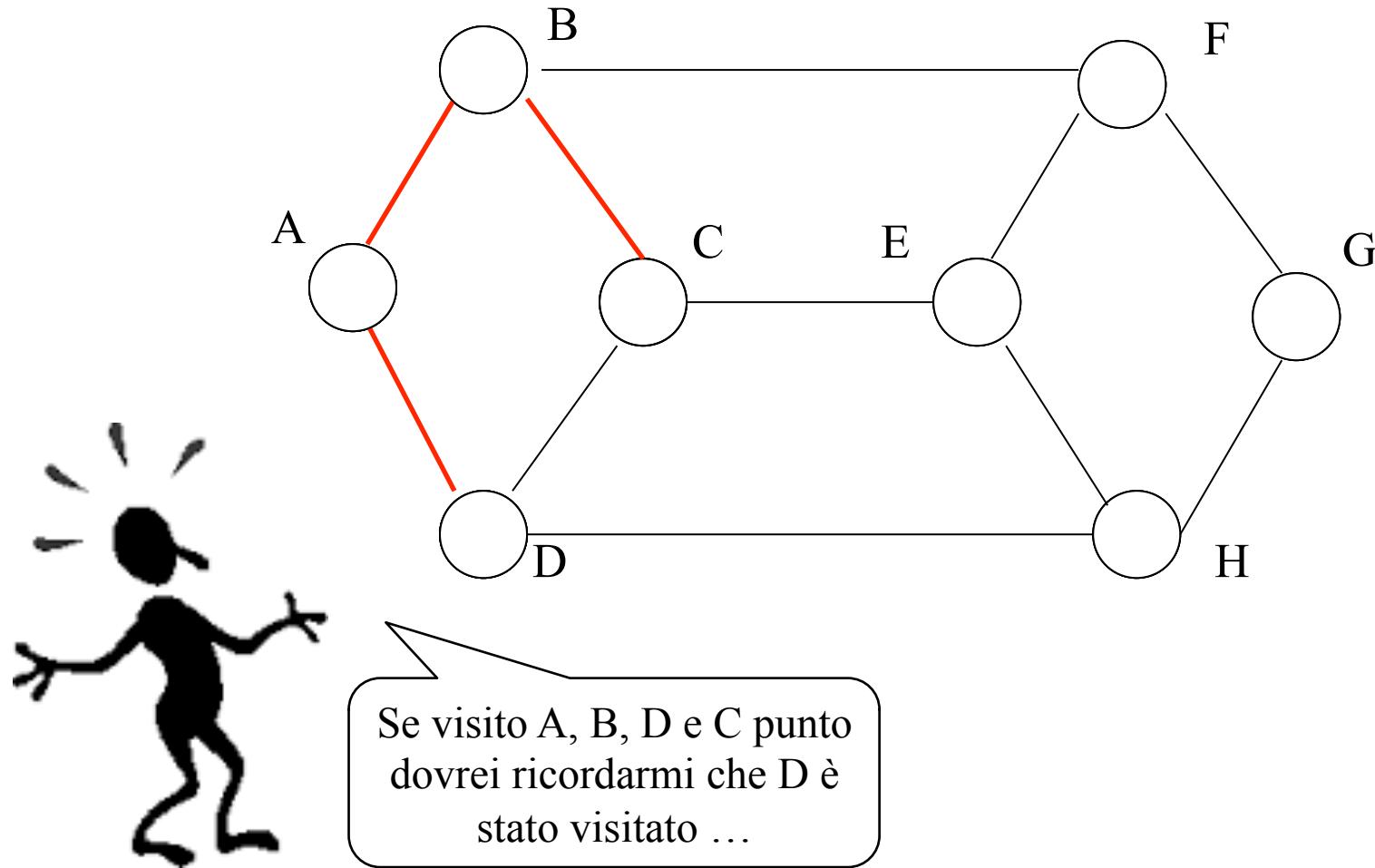


Visita di un grafo

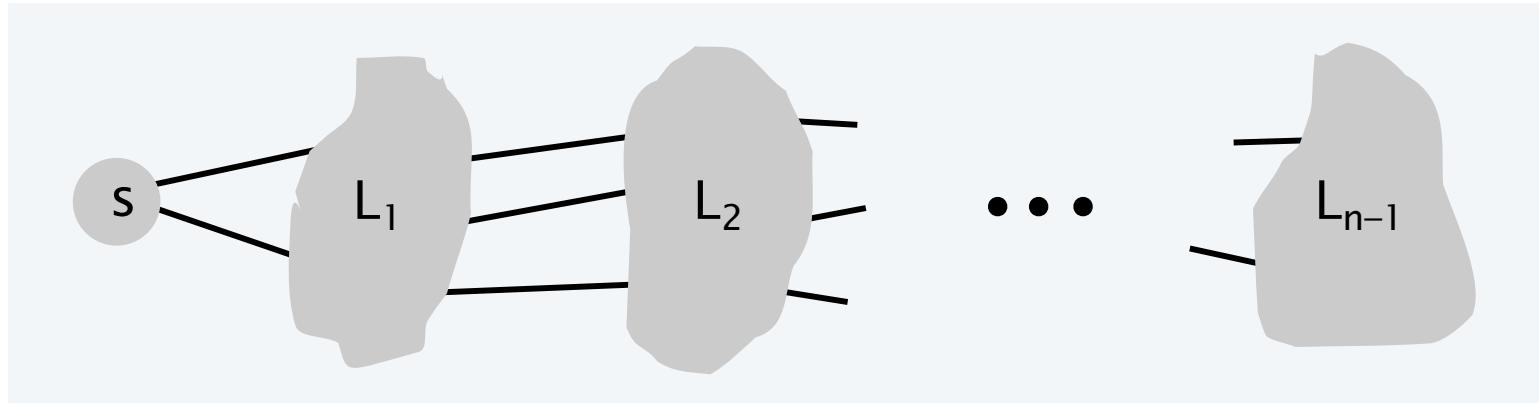


Come posso visitare tutti i vertici esattamente una volta?

Visita di un grafo



Visita in ampiezza - BFS



Per evitare i cicli
procediamo per *layer* di
vertici equidistanti da s



BFS una versione astratta

BFS-LEYER(G, s)

visita s

$L_0 = \{s\}$, $T \leftarrow \emptyset$, $i \leftarrow 0$

while $L_i \neq \emptyset$ **do**

$L_{i+1} \leftarrow \emptyset$

for all $u \in L_i$ **do**

for all $v \in adj[u]$ **do**

if $v \notin \bigcup_{j \leq i} L_j$ **then**

 visita v

$L_{i+1} \leftarrow L_{i+1} \cup \{v\}$

$T \leftarrow T \cup \{(u, v)\}$

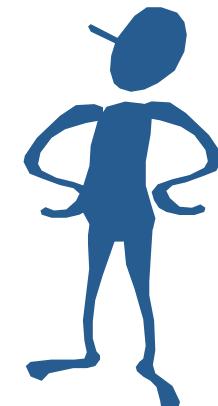
end if

end for

end for

$i \leftarrow i + 1$

end while



BFS e distanza

Cammino di lunghezza n : v_0, \dots, v_n di vertici t.c. $(v_i, v_{i+1}) \in E$ per ogni $0 \leq i < n$

Distanza di v da u :

$$\delta(u, v) = \begin{cases} n & n \text{ lunghezza minima di un cammino da } u \text{ a } v \text{ se esiste} \\ \infty & \text{altrimenti} \end{cases}$$

Teorema. Siano L_0, L_1, \dots i layer generati da una BFS di $G = (V, E)$ a partire dalla sorgente $s \in V$; allora i layer sono due a due disgiunti e

$$L_i = \{v \in V \mid \delta(s, v) = i\}$$

Inoltre il layer coincidono con i livelli dell'albero T , i cui vertici sono esattamente quelli raggiungibili da s .

BFS e distanza

$$L_i = \{v \in V \mid \delta(s, v) = i\}$$

$$v \in L_{i+1} \Rightarrow v \notin \bigcup_{j \leq i} L_j \ \& \ \exists u \in L_i. \ (u, v) \in E$$



$$\delta(s, v) \leq \delta(s, u) + 1 = i + 1 \quad (\text{ip. ind.})$$

Se fosse $\delta(s, v) = j < i+1$ allora per induzione $v \in L_j$ con $j \leq i$, quindi $\delta(s, v) = i+1$

BFS una versione astratta

BFS-LEYER(G, s)

visita s

$L_0 = \{s\}$, $T \leftarrow \emptyset$, $i \leftarrow 0$

while $L_i \neq \emptyset$ **do**

$L_{i+1} \leftarrow \emptyset$

for all $u \in L_i$ **do**

for all $v \in adj[u]$ **do**

if $v \notin \bigcup_{j \leq i} L_j$ **then**
 visita v

$L_{i+1} \leftarrow L_{i+1} \cup \{v\}$

$T \leftarrow T \cup \{(u, v)\}$

end if

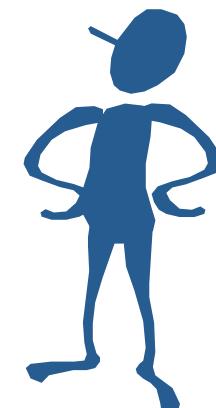
end for

end for

$i \leftarrow i + 1$

end while

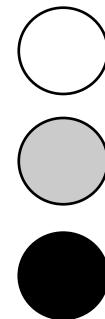
Il test $v \notin L_j$ per ogni $j \leq i$ è
molto costoso; si può fare di
meglio?



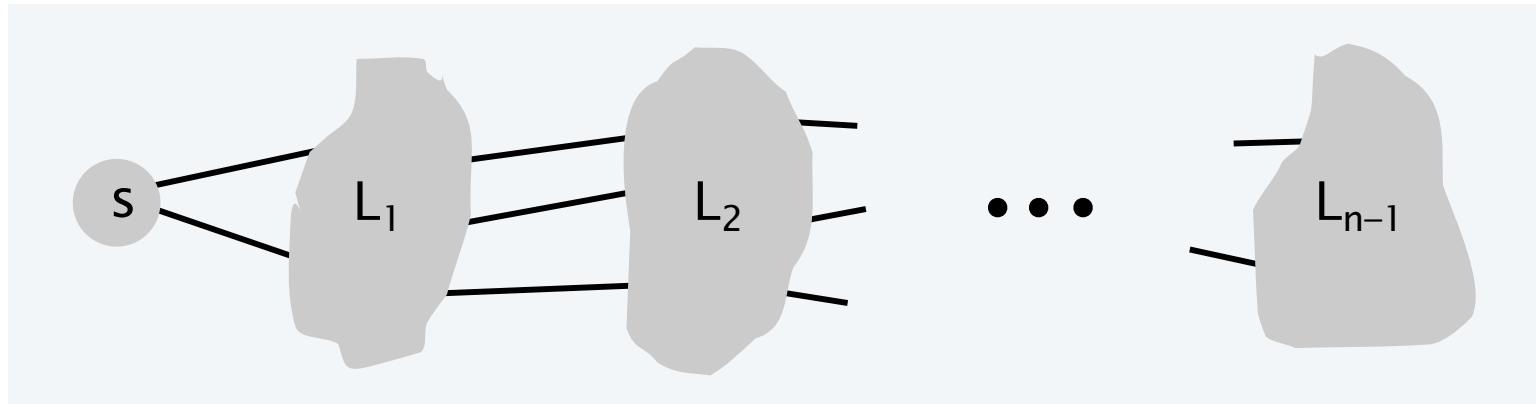
Visita di un grafo

Coloriamo i vertici:

- Bianco: *esterno*, non ancora scoperto né visitato
- Grigio: sulla *frontiera*, scoperto ma non visitato
- Nero: interno, visitato



Visita in ampiezza - BFS



Procediamo come con gli alberi in modo che la **coda** rappresenti la frontiera

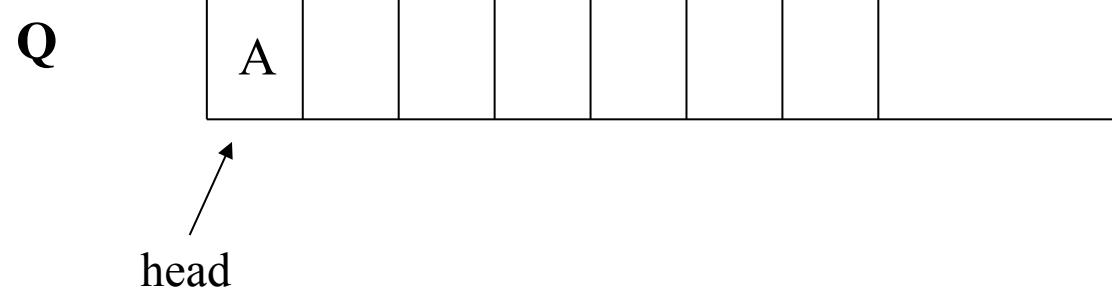
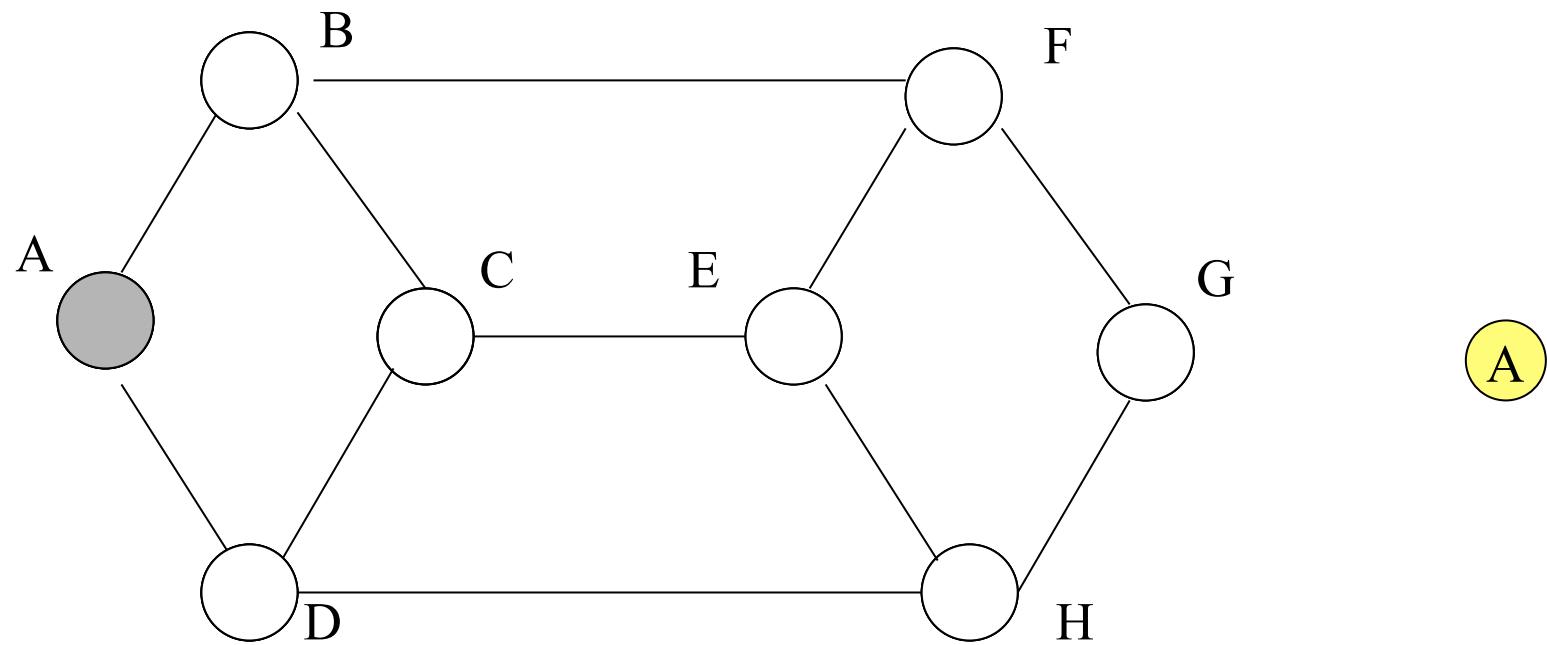


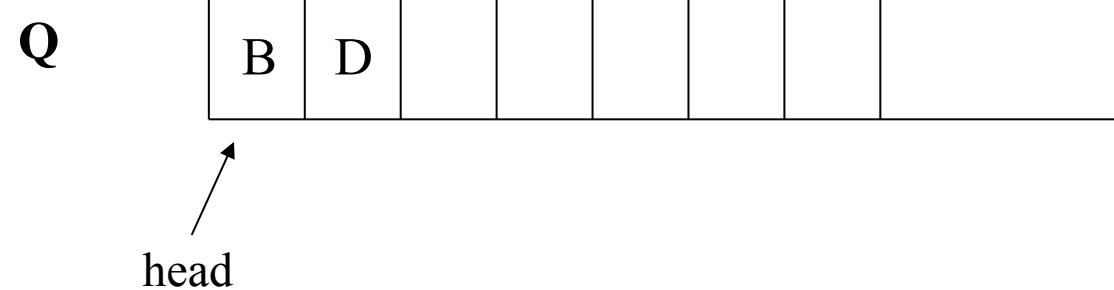
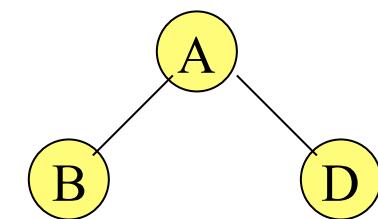
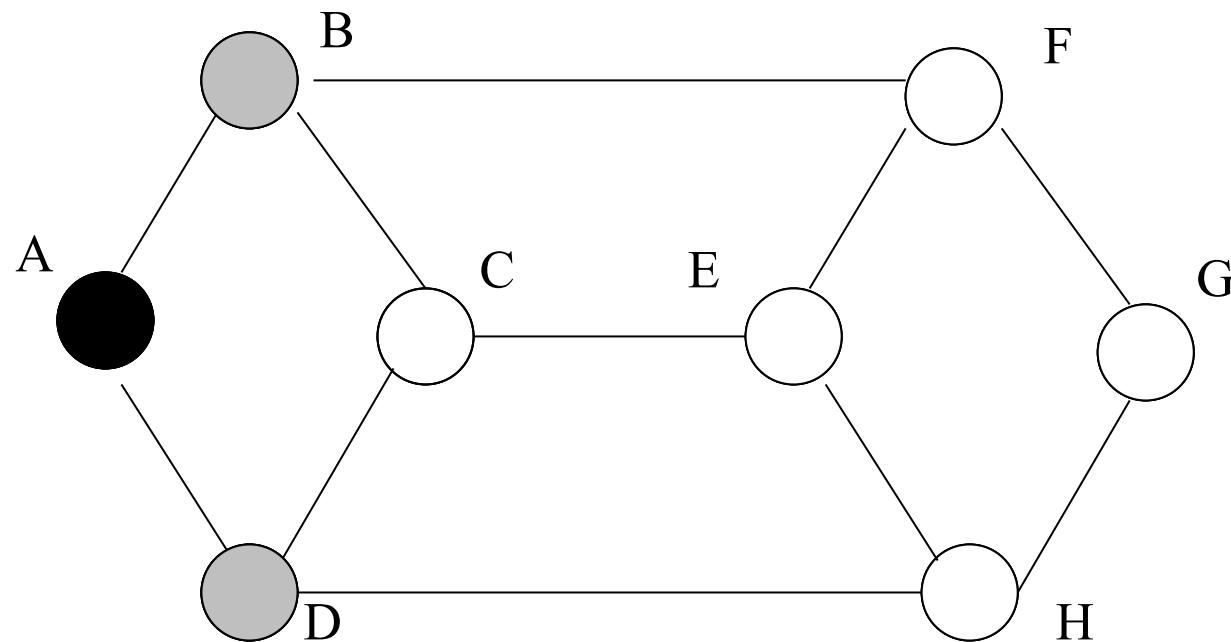
Visita in ampiezza - BFS

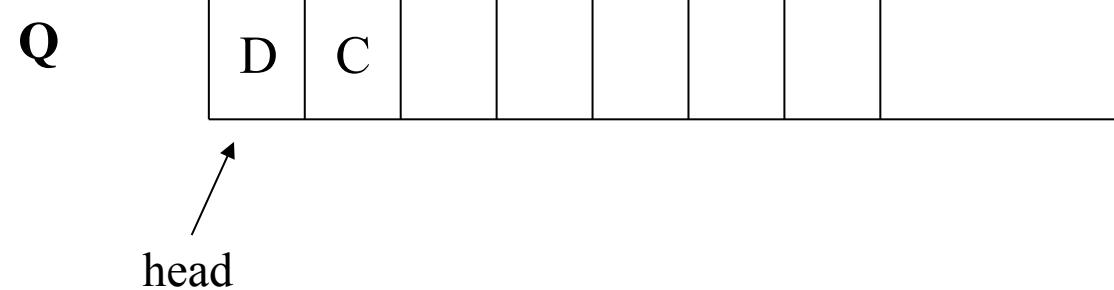
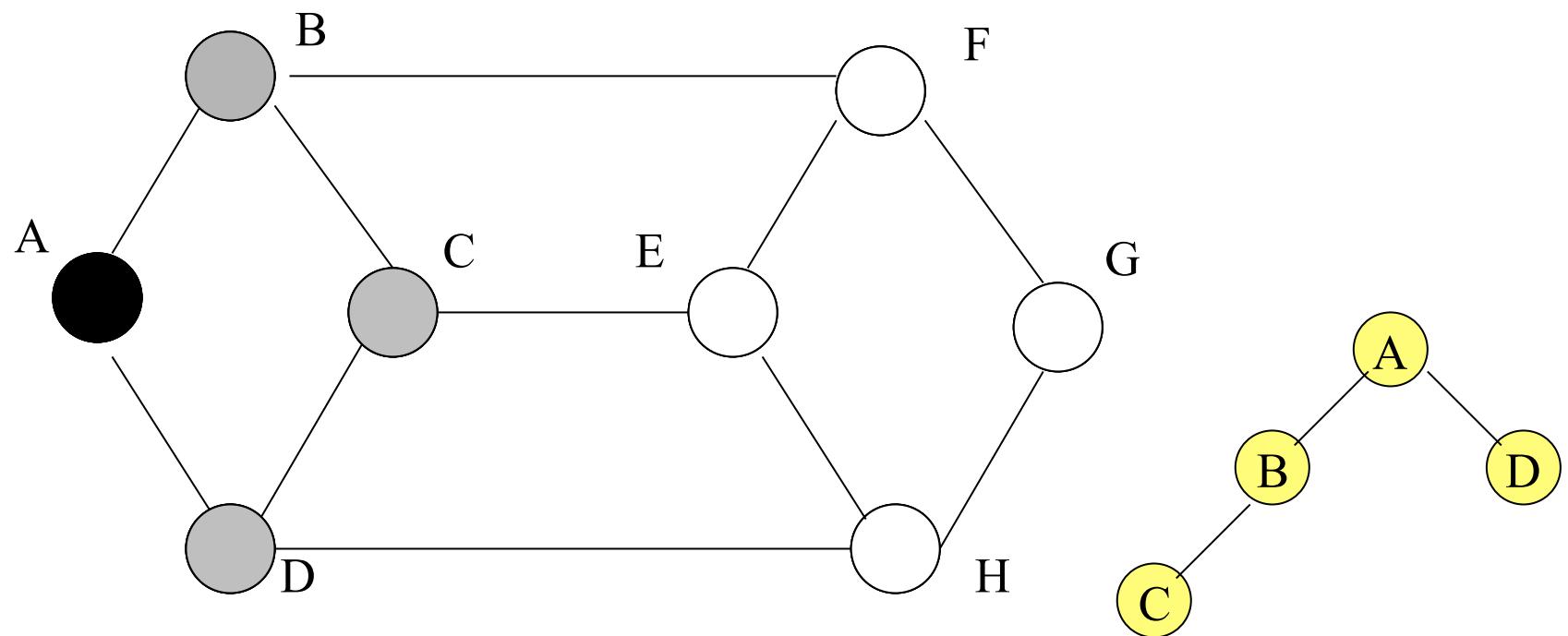
```
BFS( $G, s$ )
 $Q \leftarrow \text{EMPTY-QUEUE}$ 
 $s.\text{color} \leftarrow \text{grigio}$ 
ENQUEUE( $Q, s$ )
while NON-EMPTY( $Q$ ) do
     $u \leftarrow \text{FIRST}(Q)$ 
    if  $\exists v \text{ bianco} \in \text{adj}[u]$  then
         $v.\text{color} \leftarrow \text{grigio}$ 
         $v.\pi \leftarrow u$ 
        ENQUEUE( $Q, v$ )
    else
         $u.\text{color} \leftarrow \text{black}$ 
    end if
end while
```

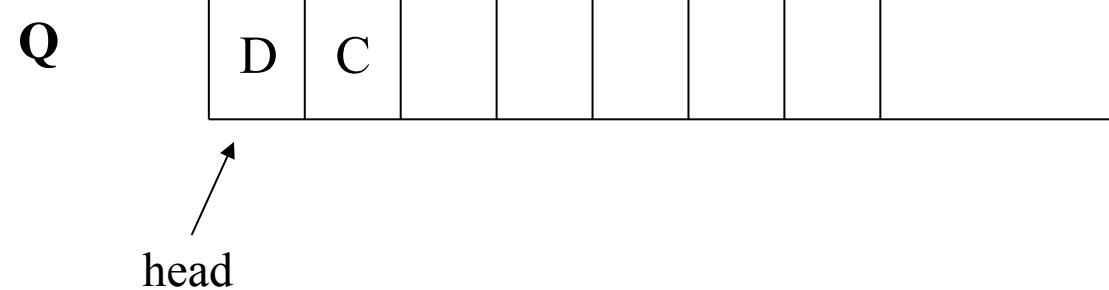
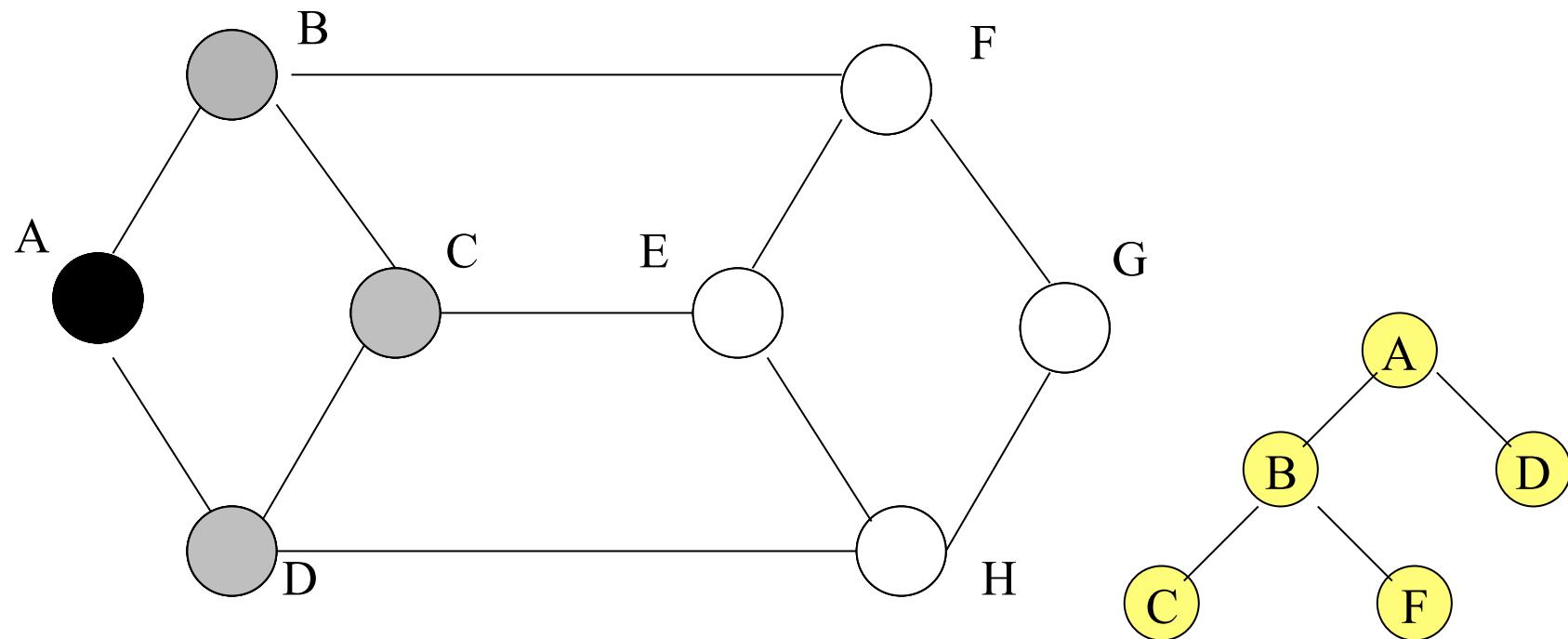
Procediamo come con gli alberi in modo che la **coda** rappresenti la frontiera

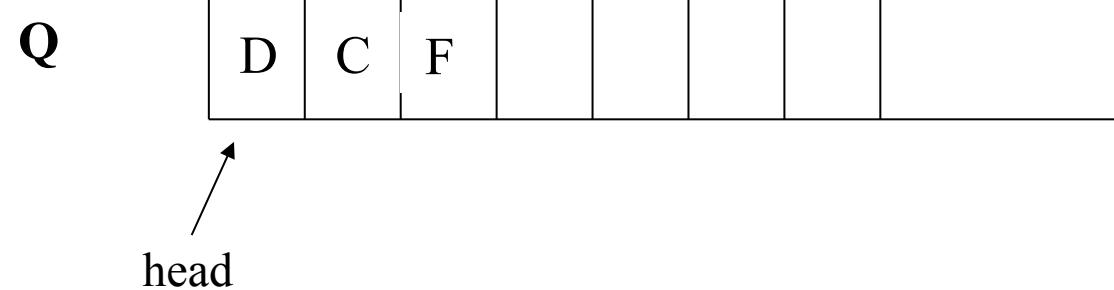
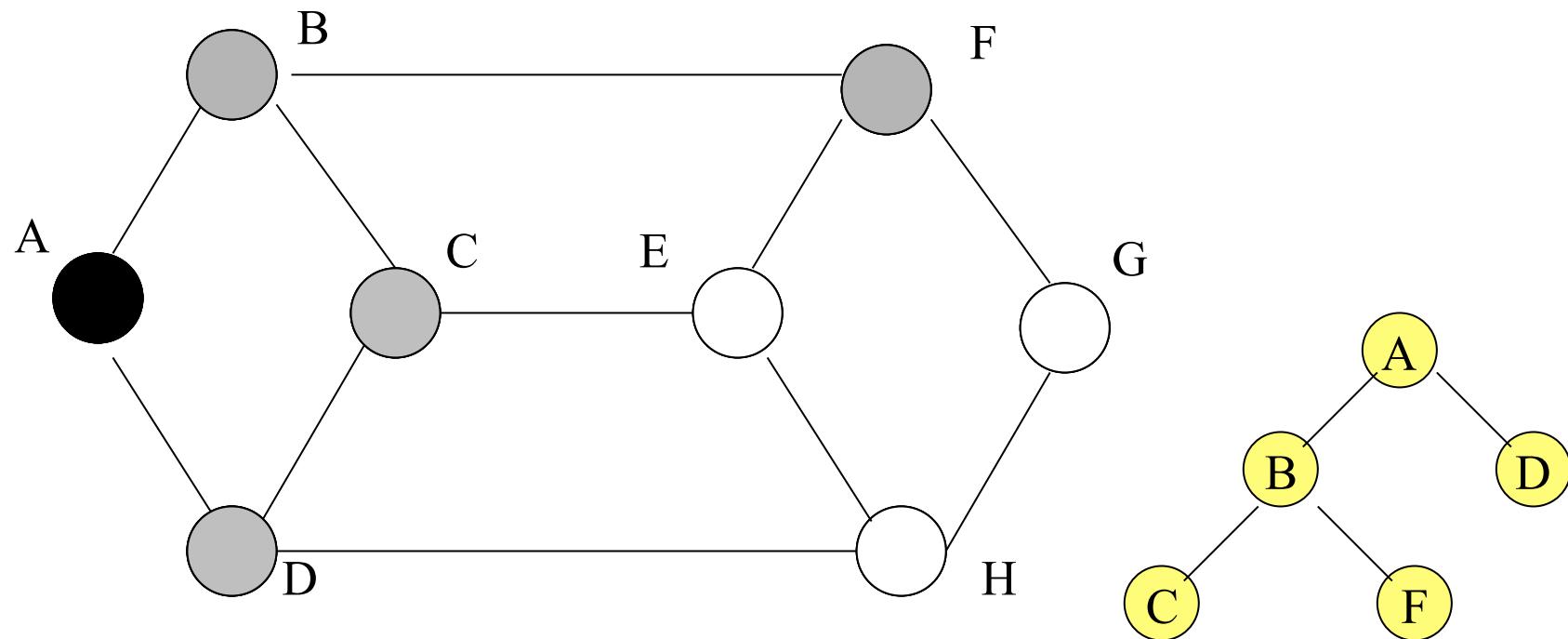


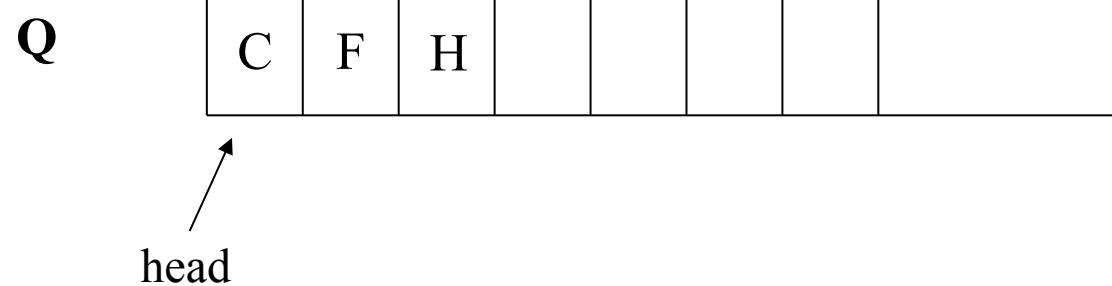
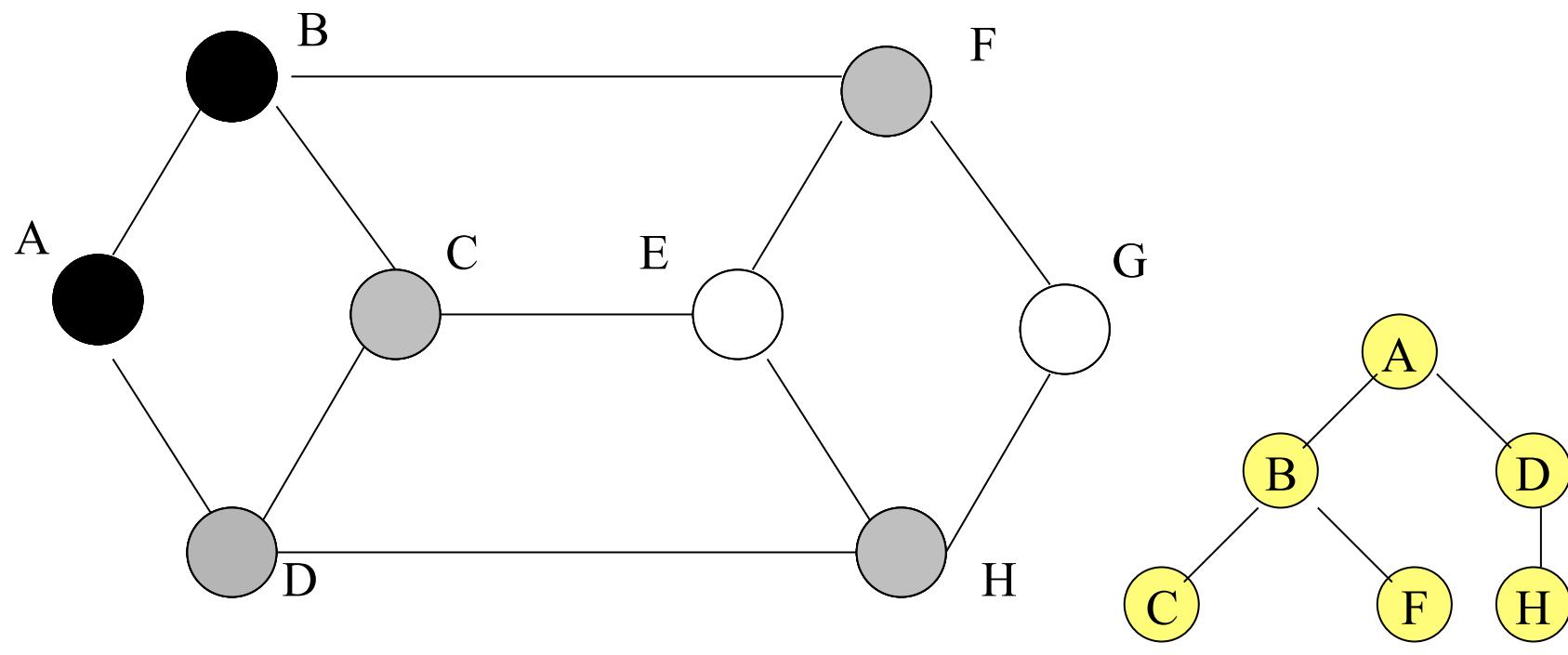


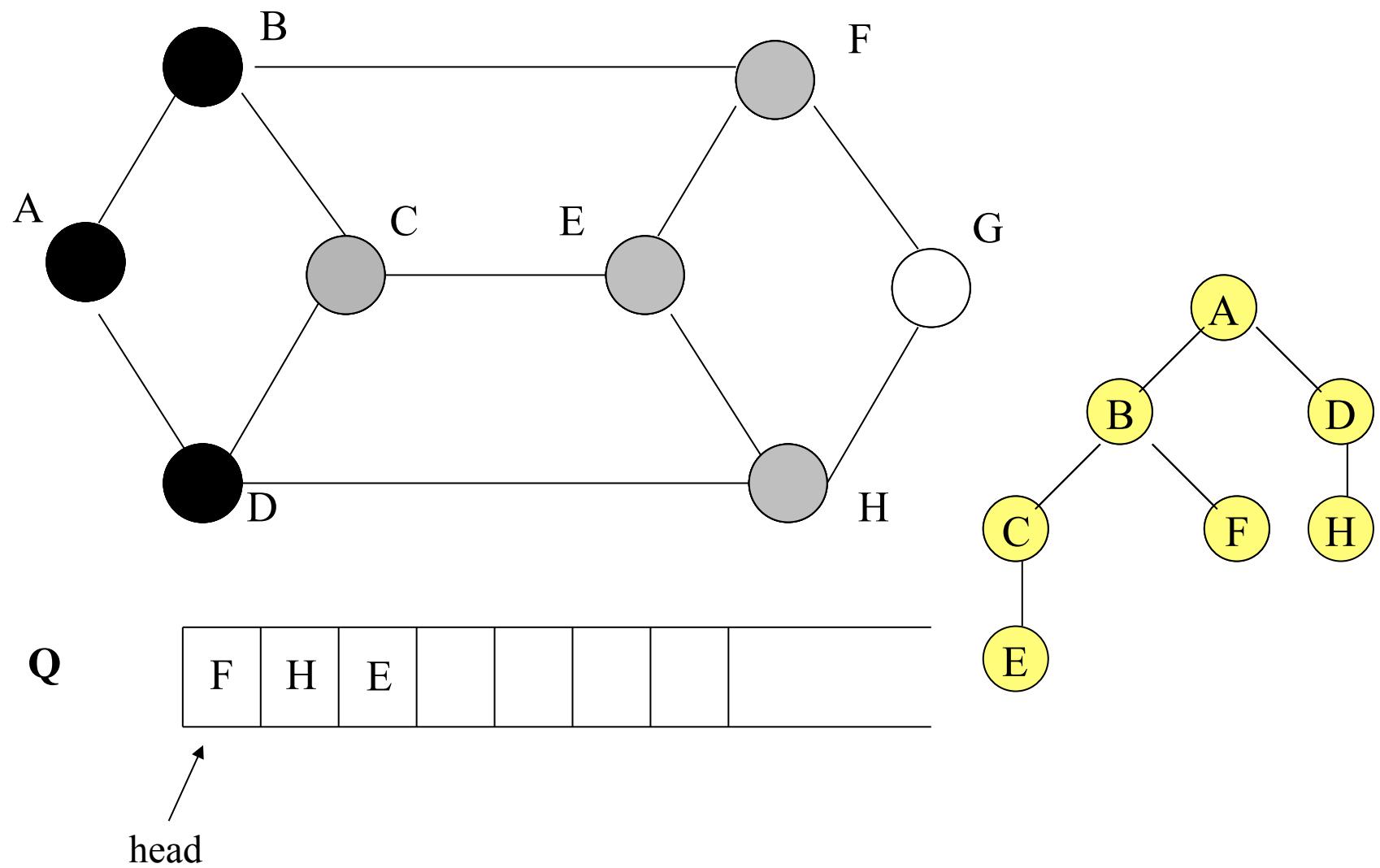


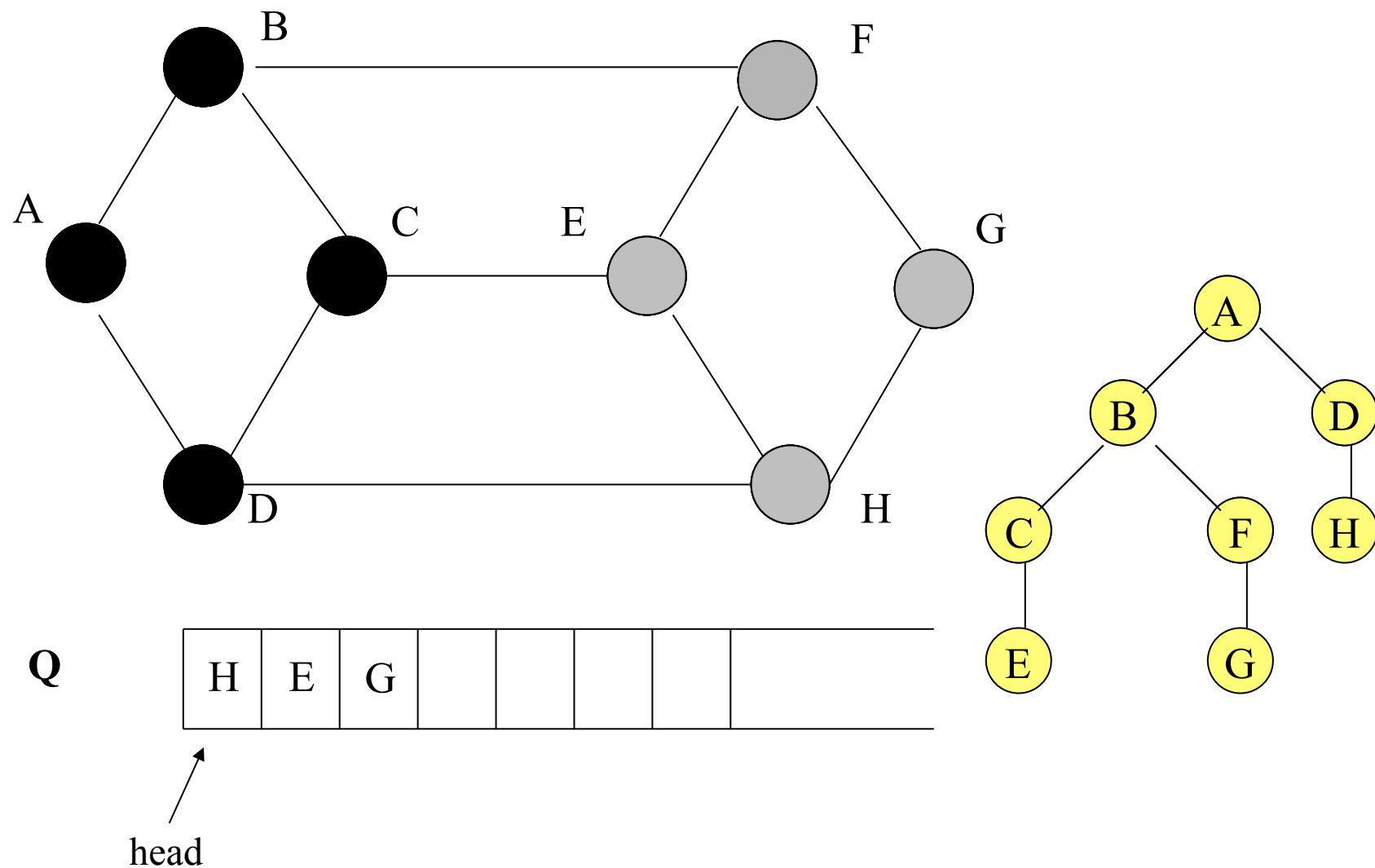


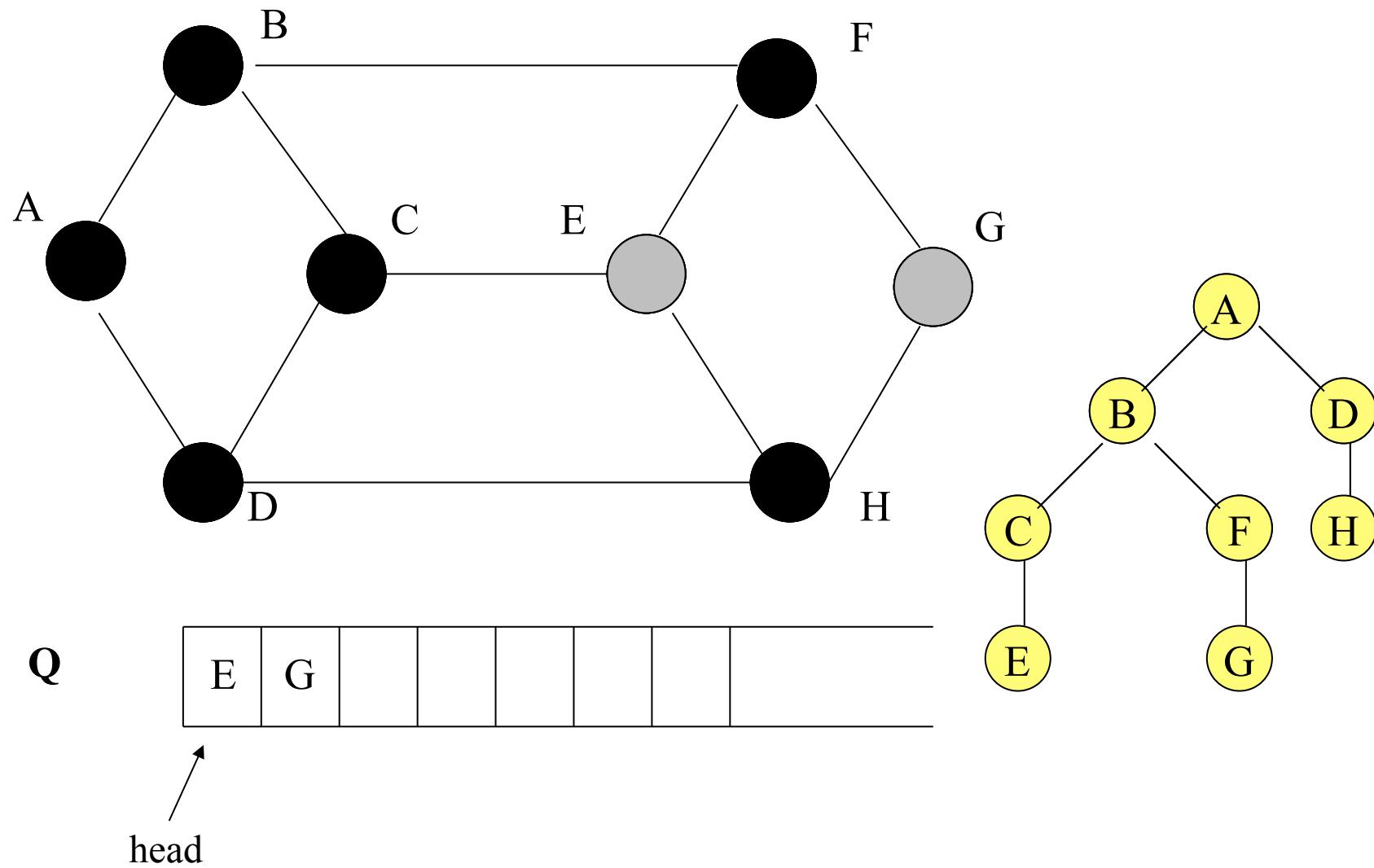


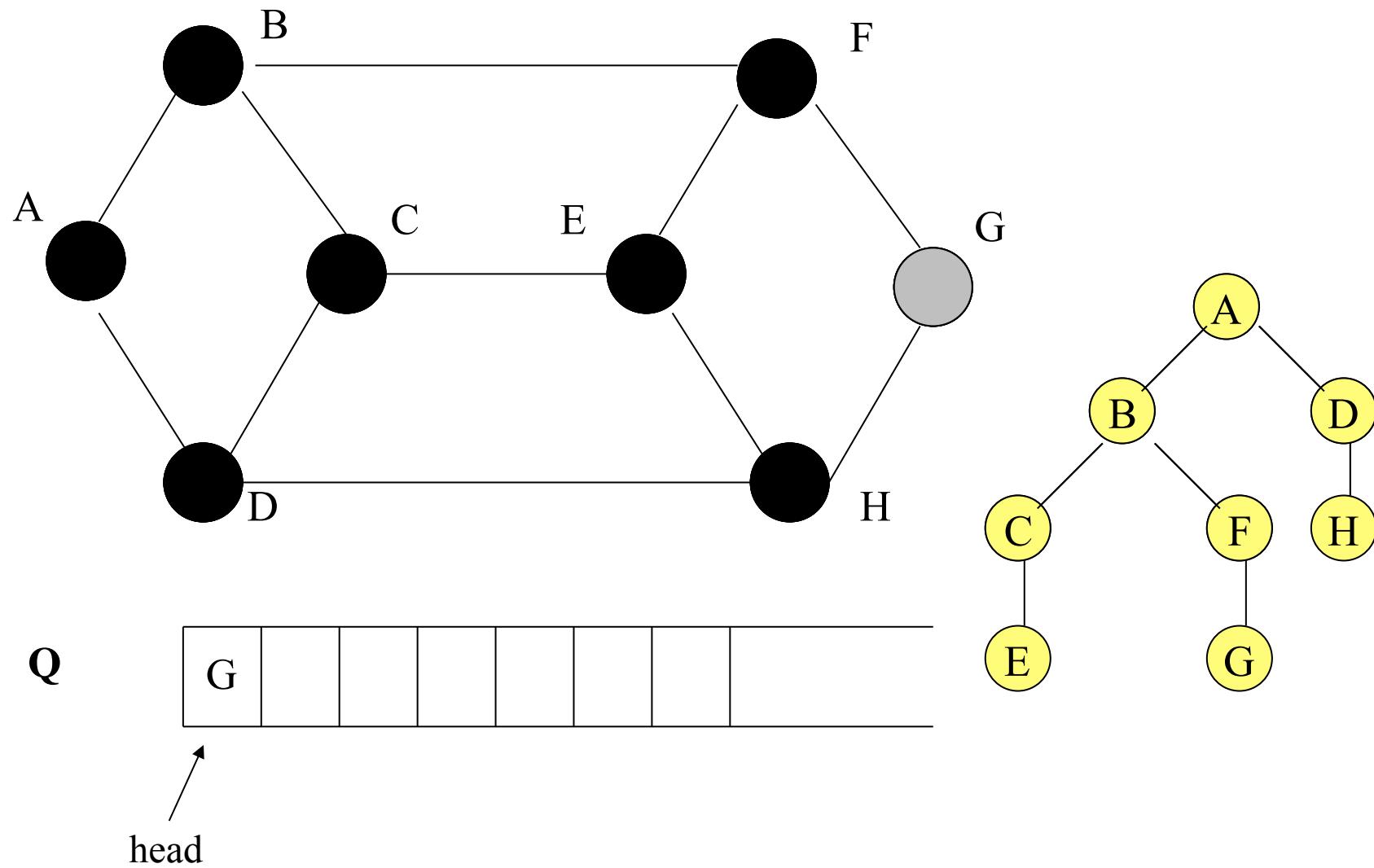


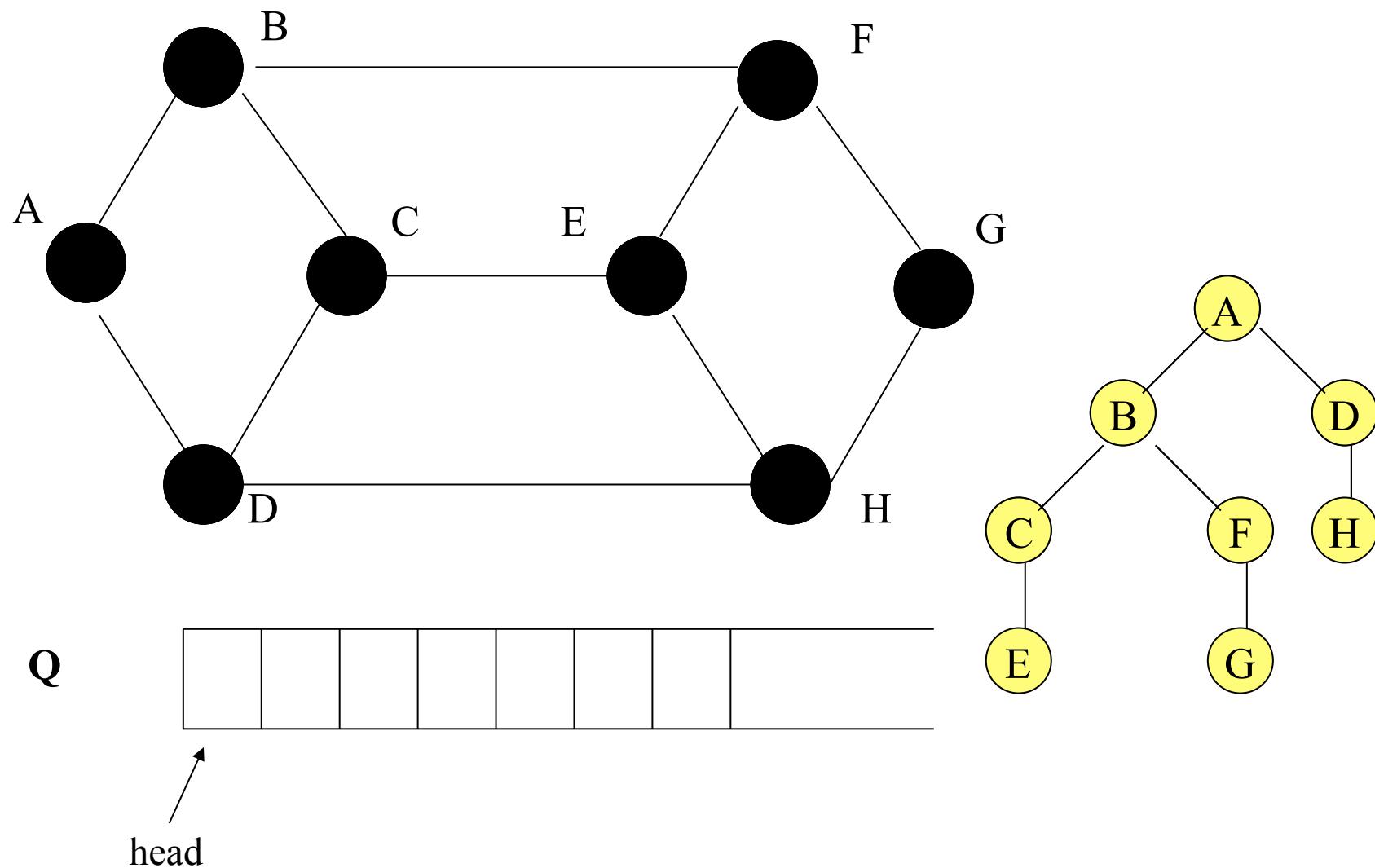








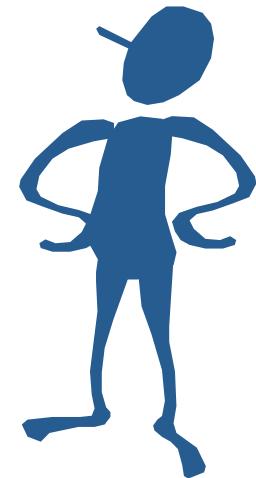




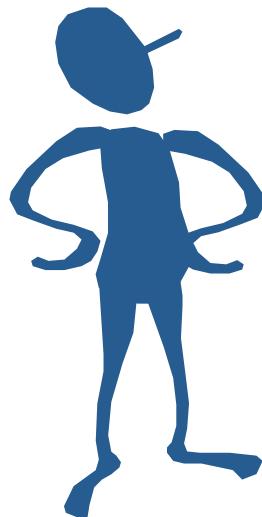
Complessità della BFS

```
BFS( $G, s$ )
 $Q \leftarrow \text{EMPTY-QUEUE}$ 
 $s.\text{color} \leftarrow \text{grigio}$ 
ENQUEUE( $Q, s$ )
while NON-EMPTY( $Q$ ) do
     $u \leftarrow \text{FIRST}(Q)$ 
    if  $\exists v \text{ bianco} \in \text{adj}[u]$  then
         $v.\text{color} \leftarrow \text{grigio}$ 
         $v.\pi \leftarrow u$ 
        ENQUEUE( $Q, v$ )
    else
         $u.\text{color} \leftarrow \text{black}$ 
        DEQUEUE( $Q$ )
    end if
end while
```

La complessità è
 $O(|V| + O|E|)$



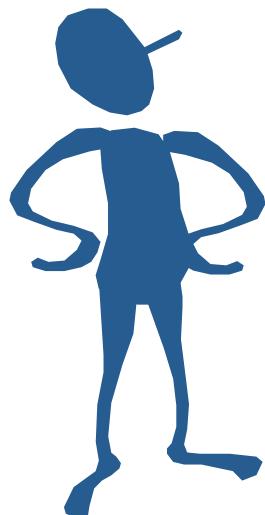
Visita in profondità - DFS



Procediamo come con gli
alberi in modo che la **pila**
rappresenti la frontiera

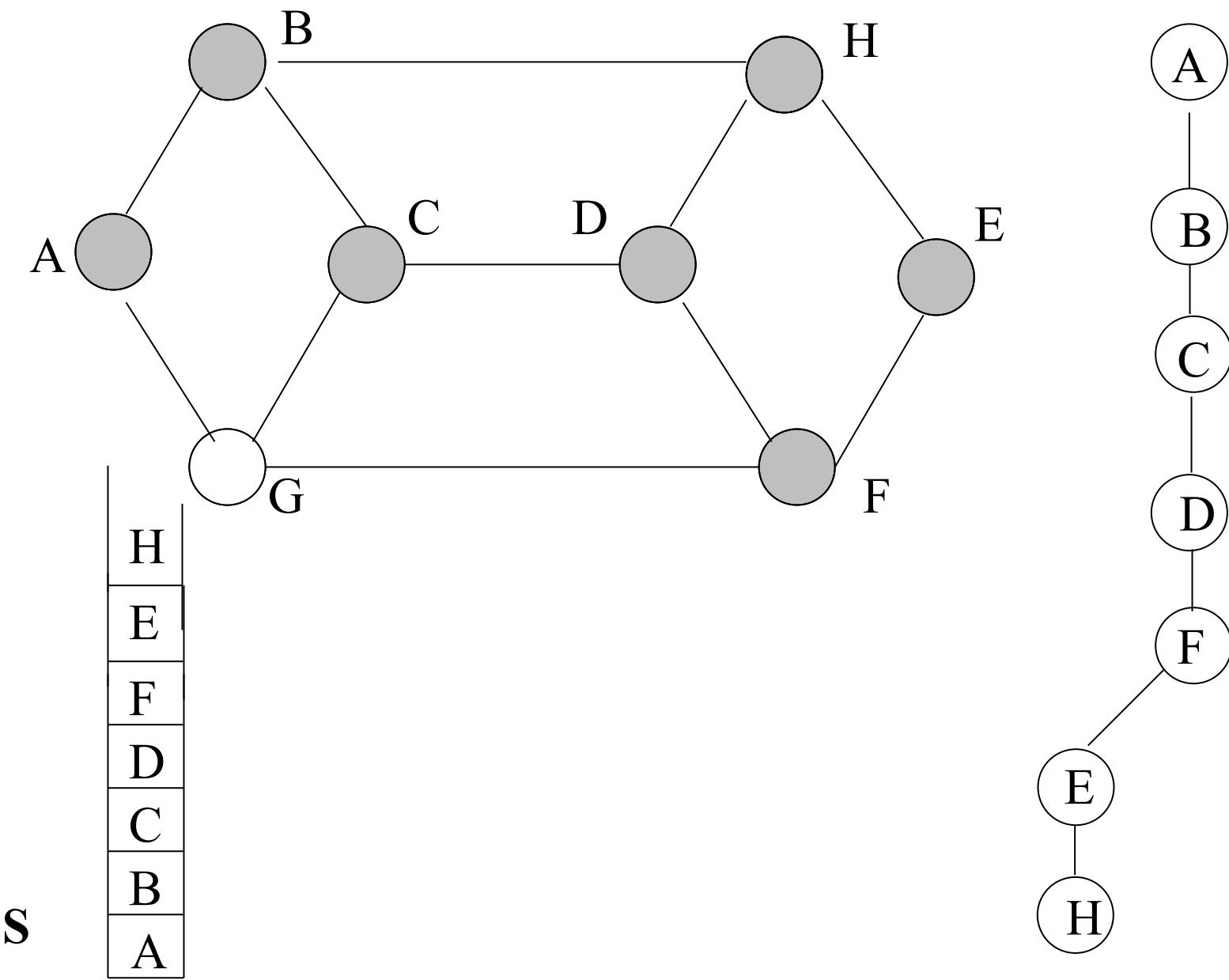
Visita in profondità - DFS

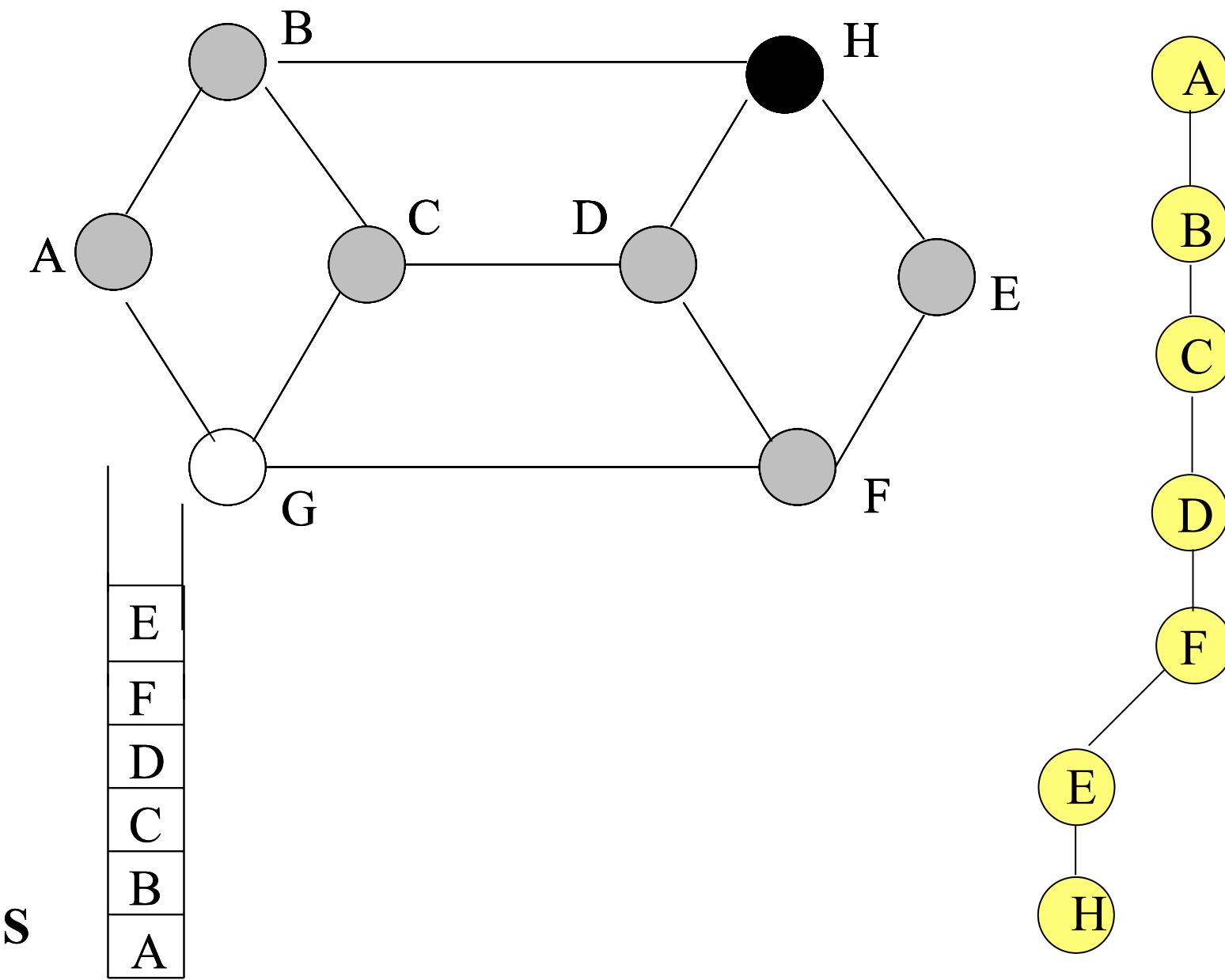
```
DFS-REC( $G, s$ )
 $s.color \leftarrow grigio$ 
for all  $v$  bianco  $\in \text{adj}[s]$  do
     $v.\pi \leftarrow s$ 
    DFS-REC( $G, v$ )
end for
 $s.color \leftarrow nero$ 
```

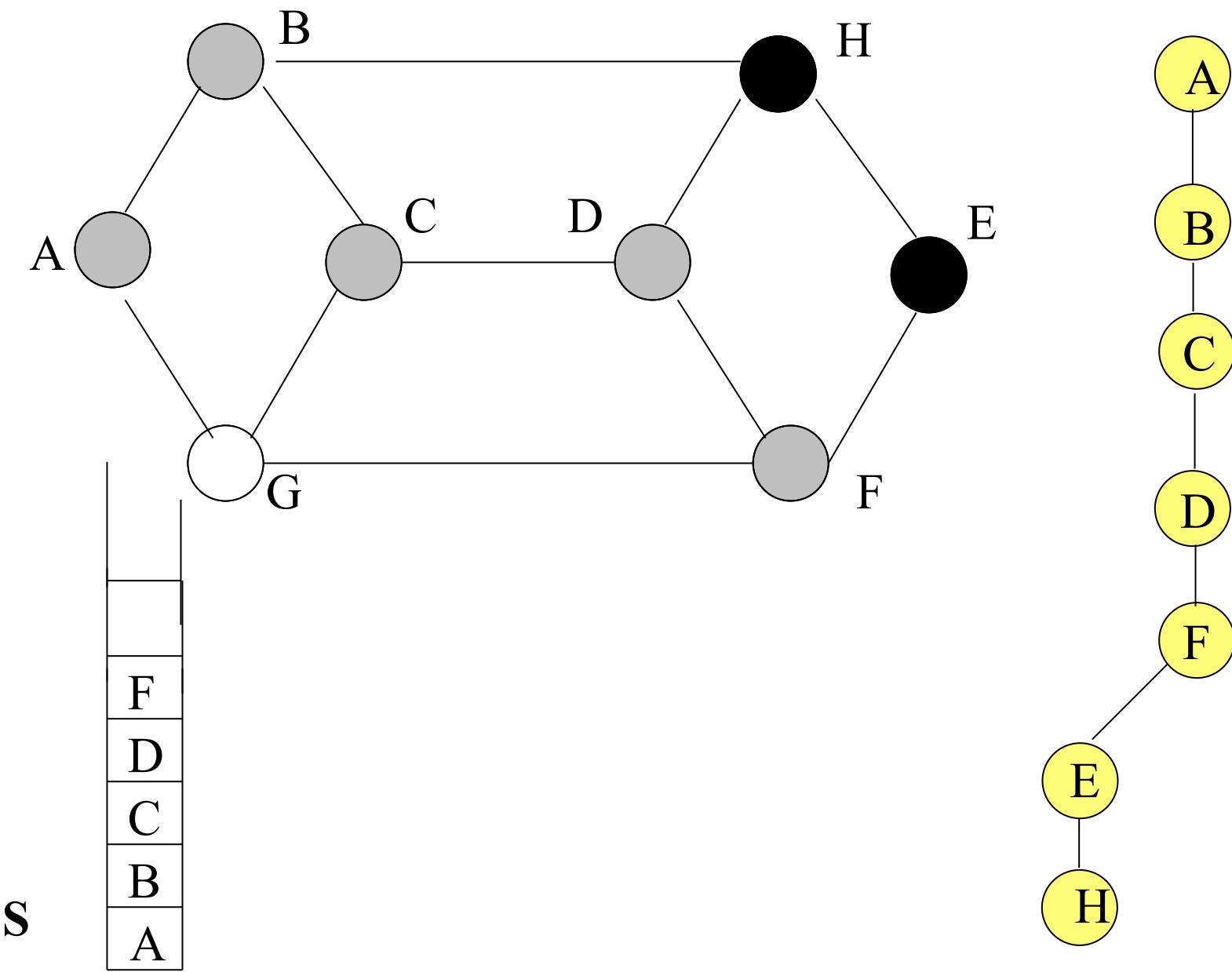


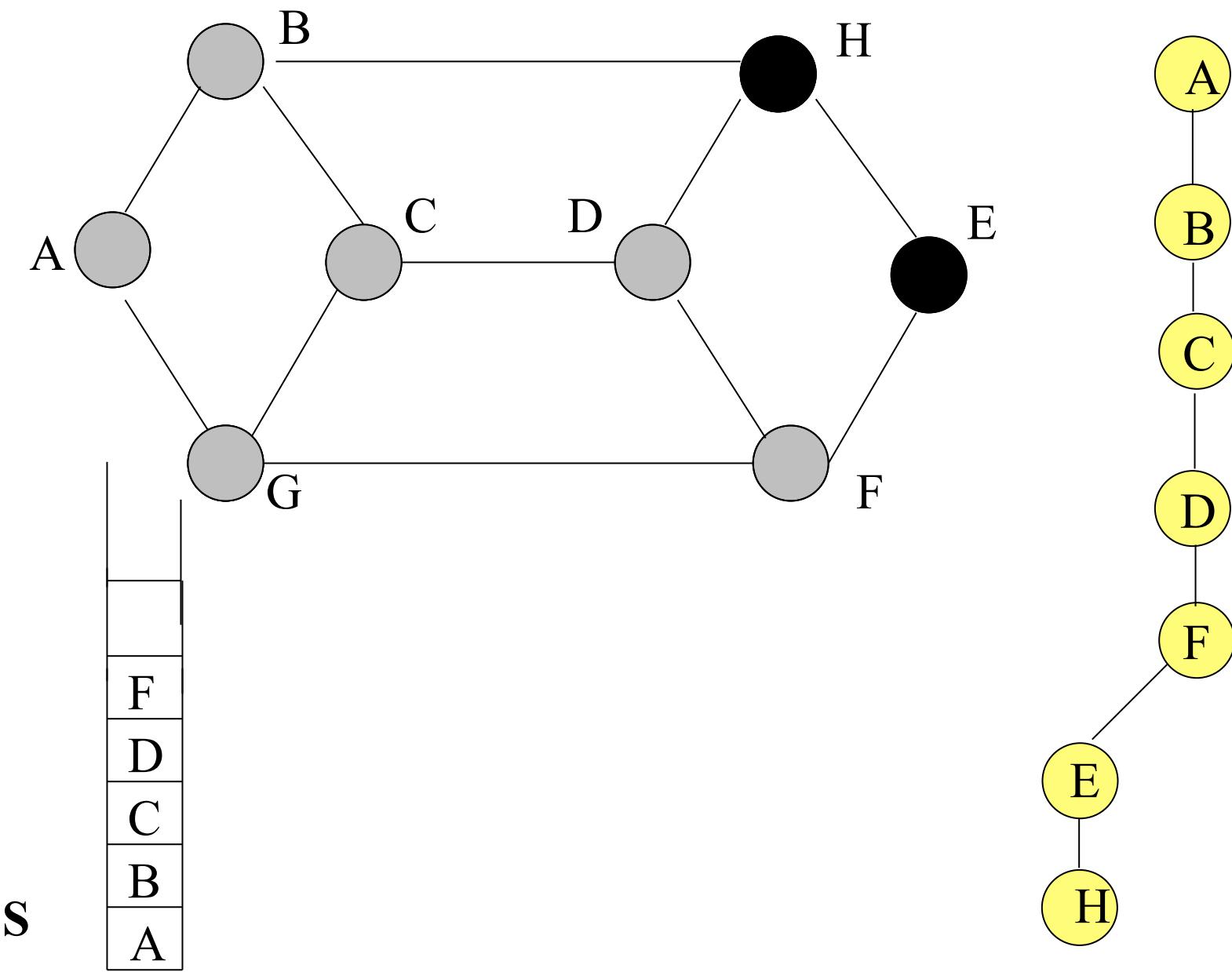
Queste due versioni
sono equivalenti

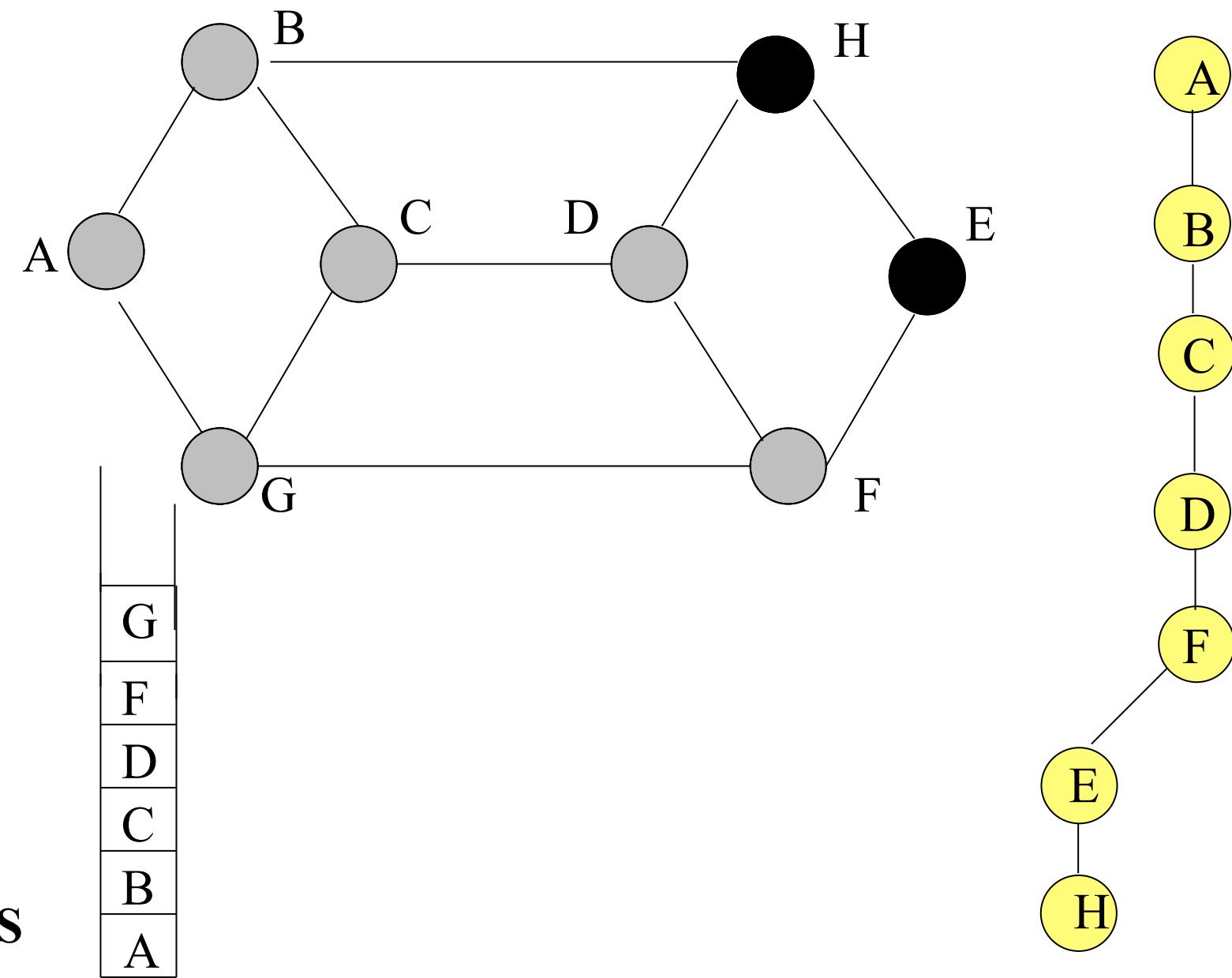
```
DFS( $G, s$ )
 $S \leftarrow \text{EMPTY-STACK}$ 
 $s.color \leftarrow grigio$ 
PUSH( $S, s$ )
while NON-EMPTY( $S$ ) do
     $u \leftarrow \text{TOP}(S)$ 
    if  $\exists v : v$  bianco  $\in \text{adj}[u]$  then
         $v.color \leftarrow grigio$ 
         $v.\pi \leftarrow u$ 
        PUSH( $S, v$ )
    else
         $u.color \leftarrow black$ 
        POP( $S$ )
    end if
end while
```

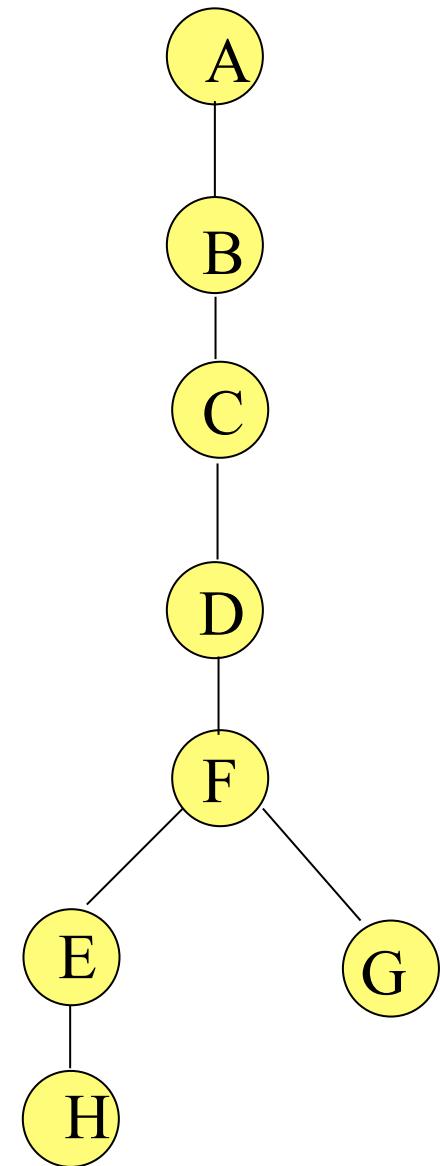
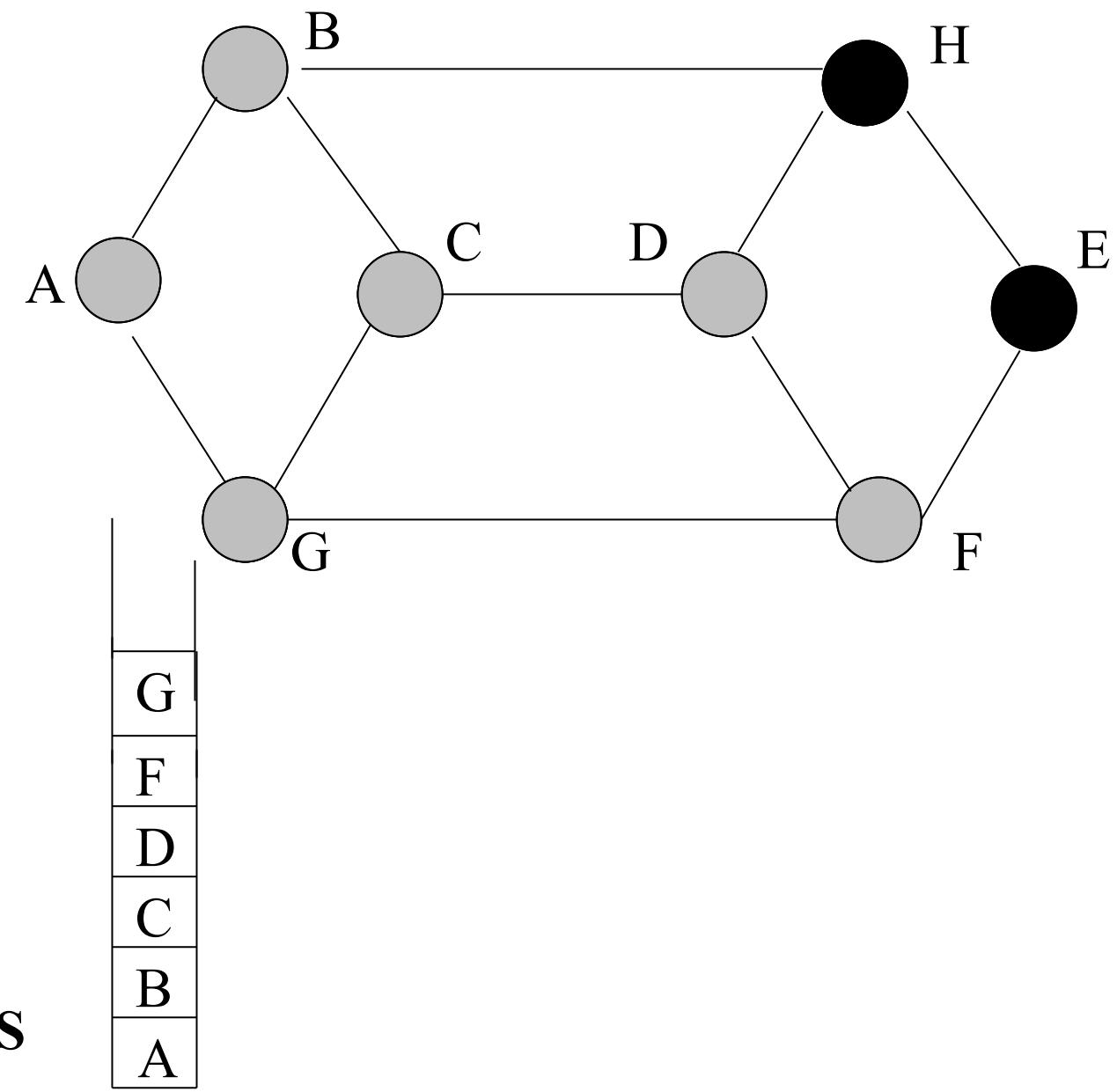


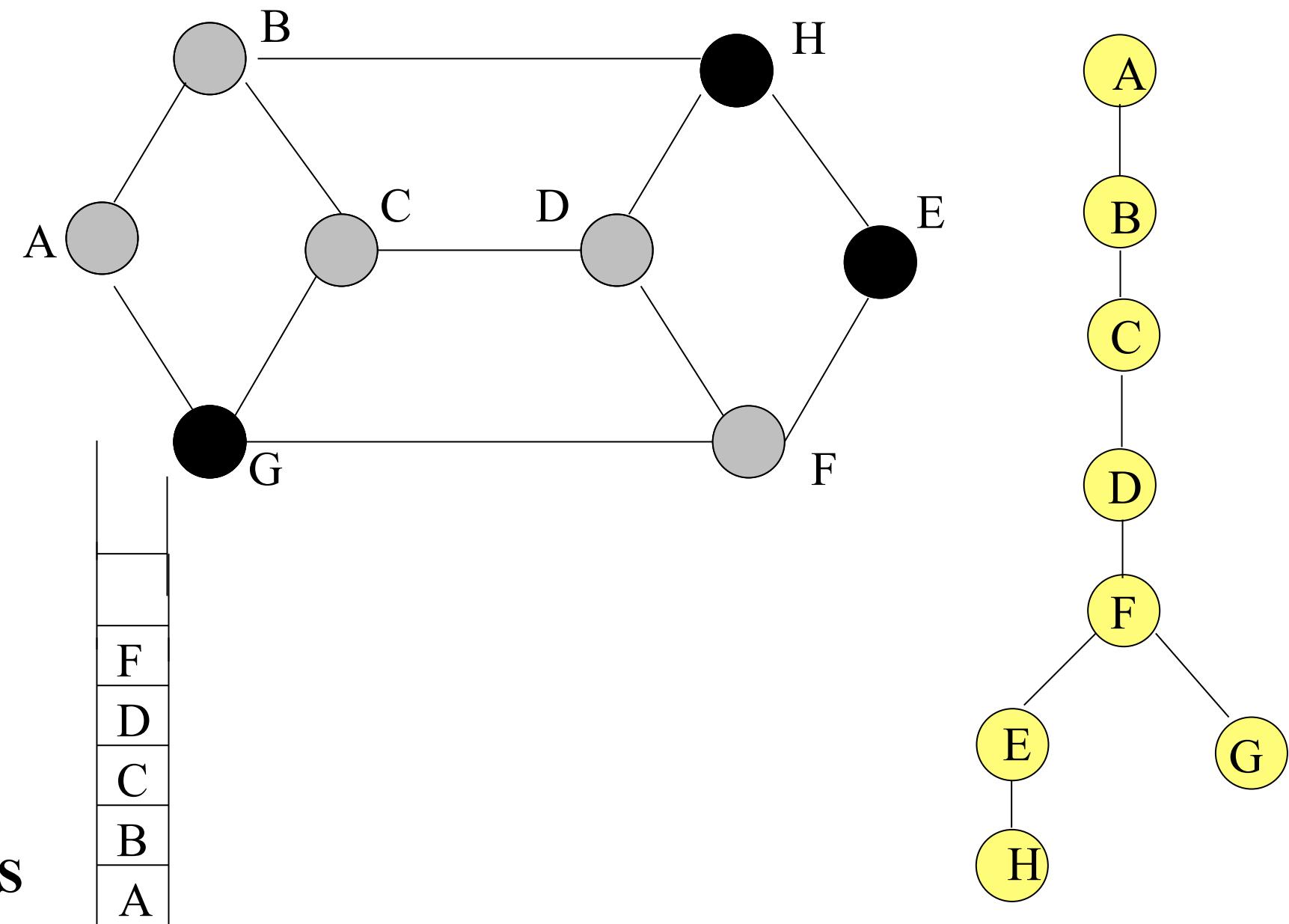


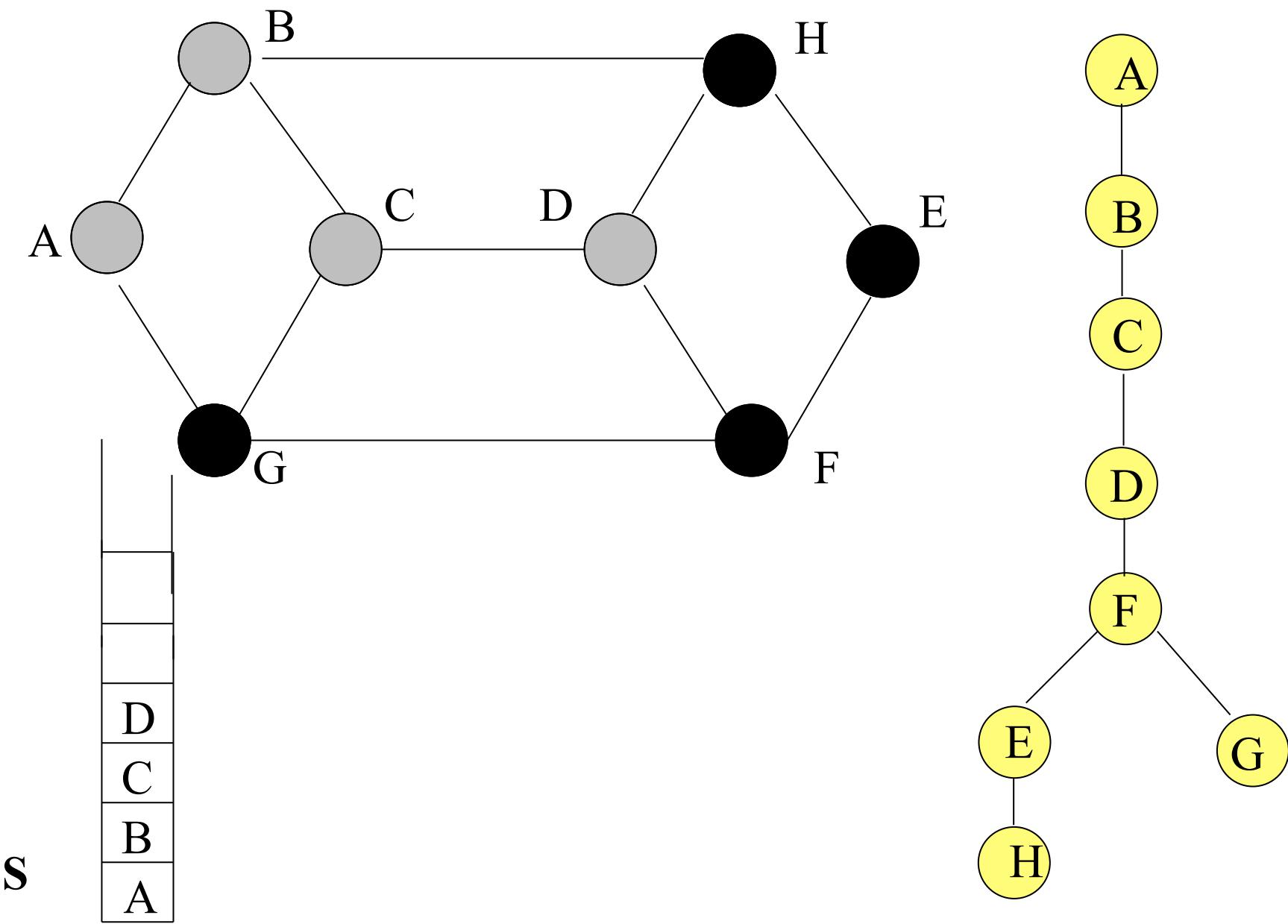


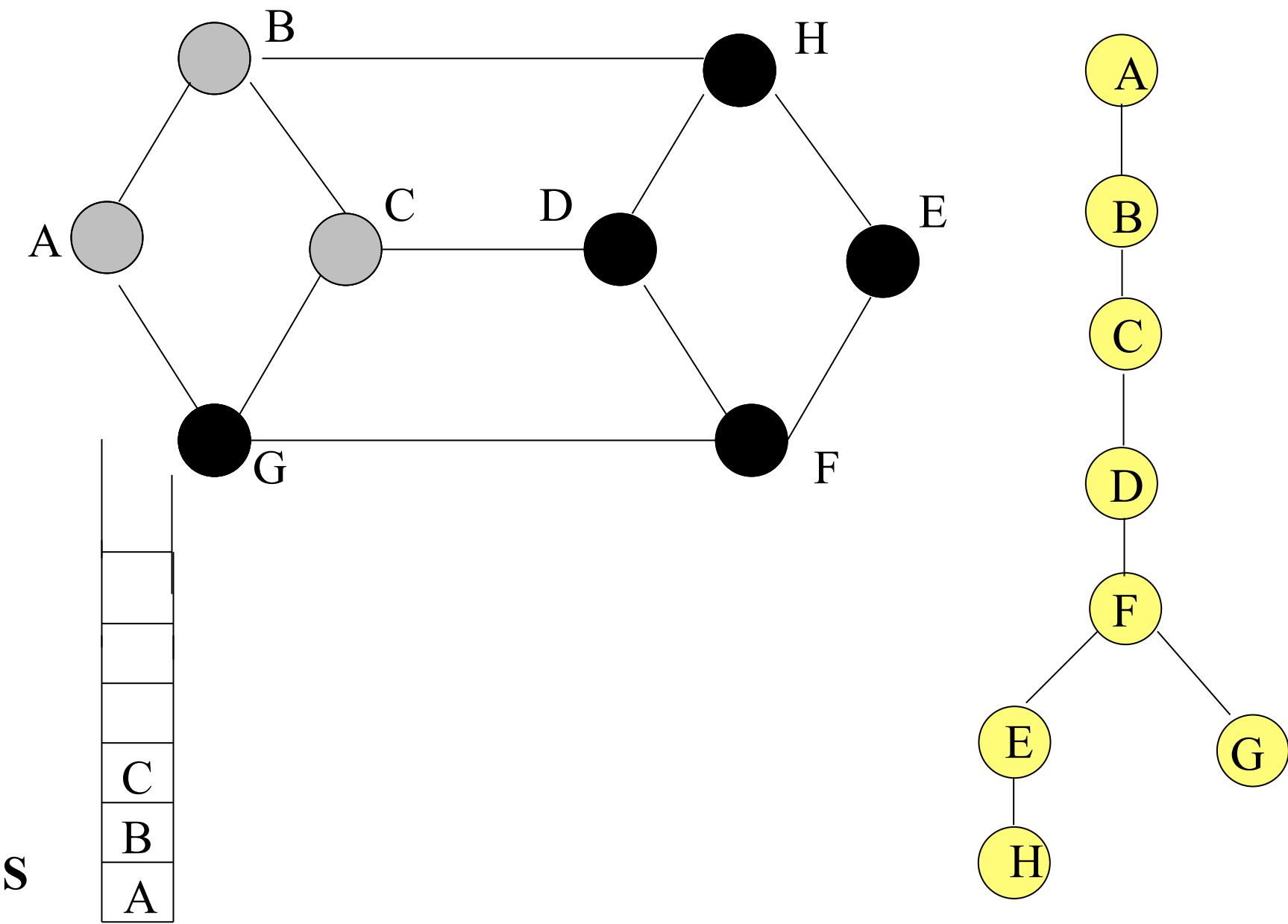


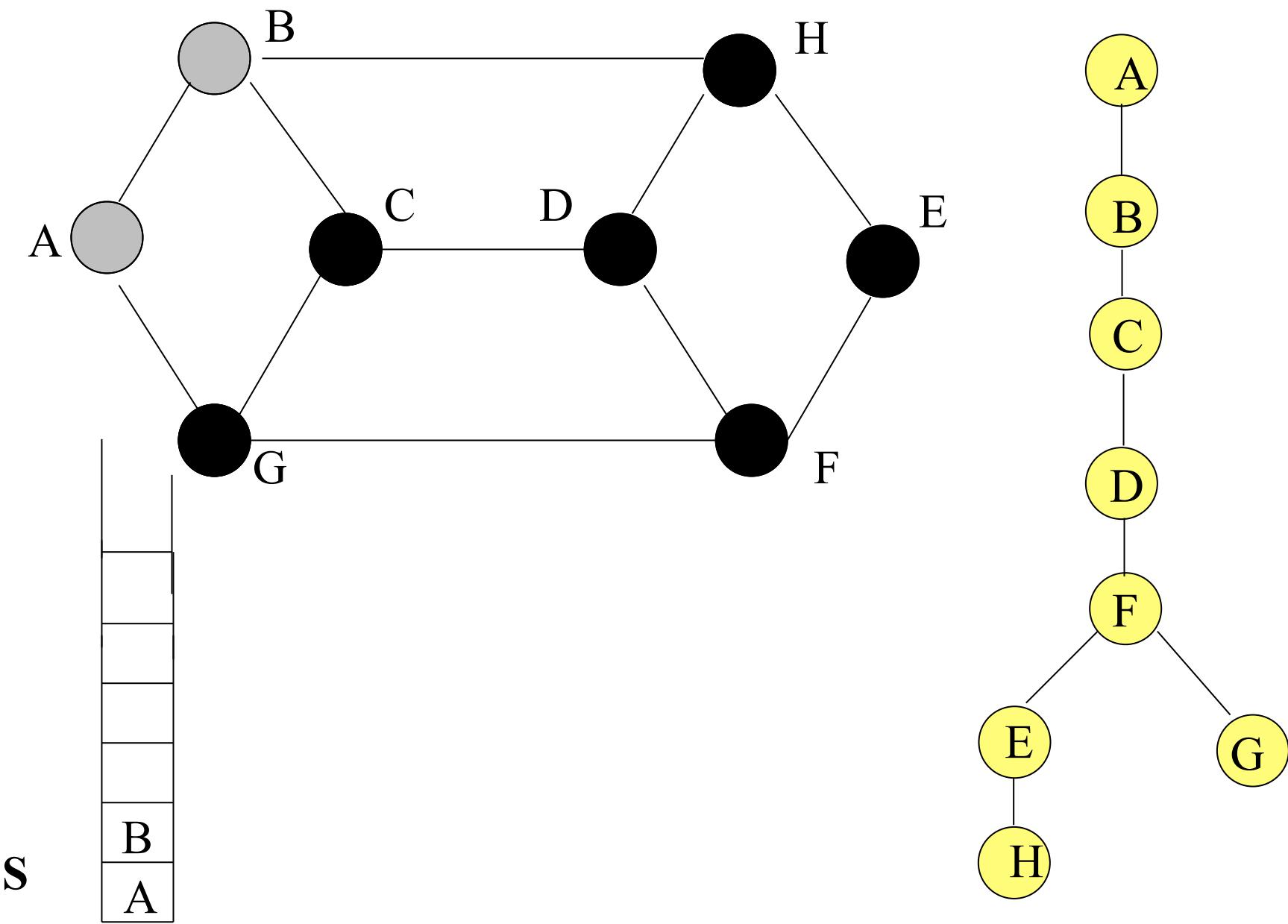


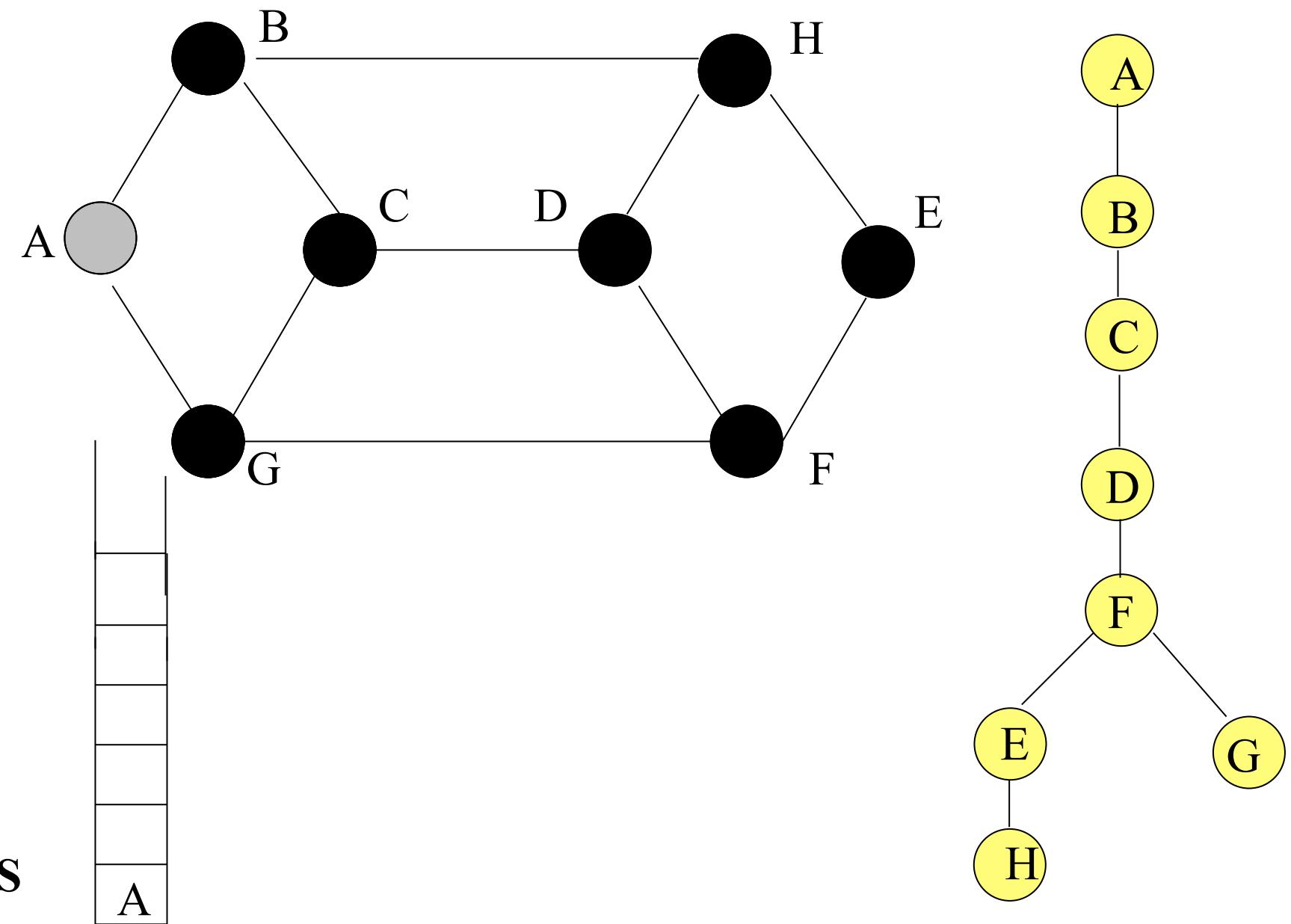


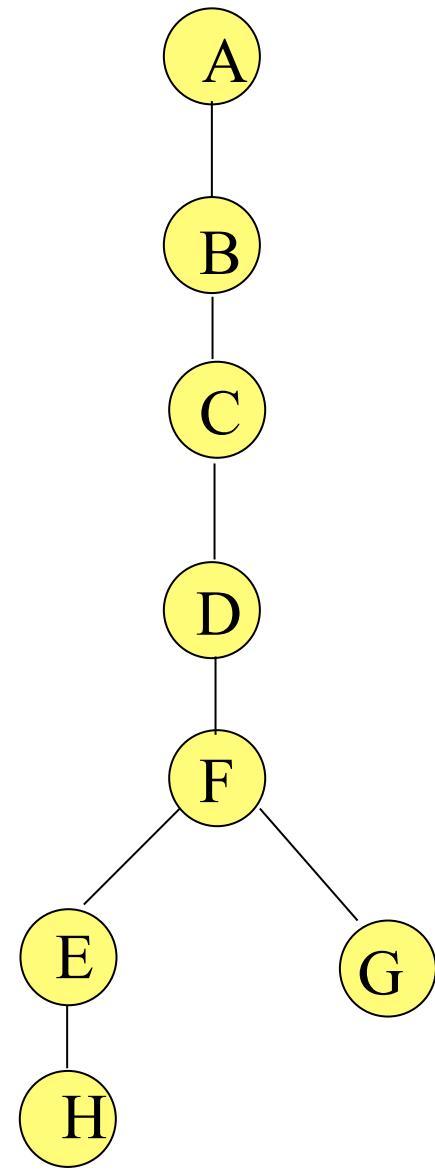
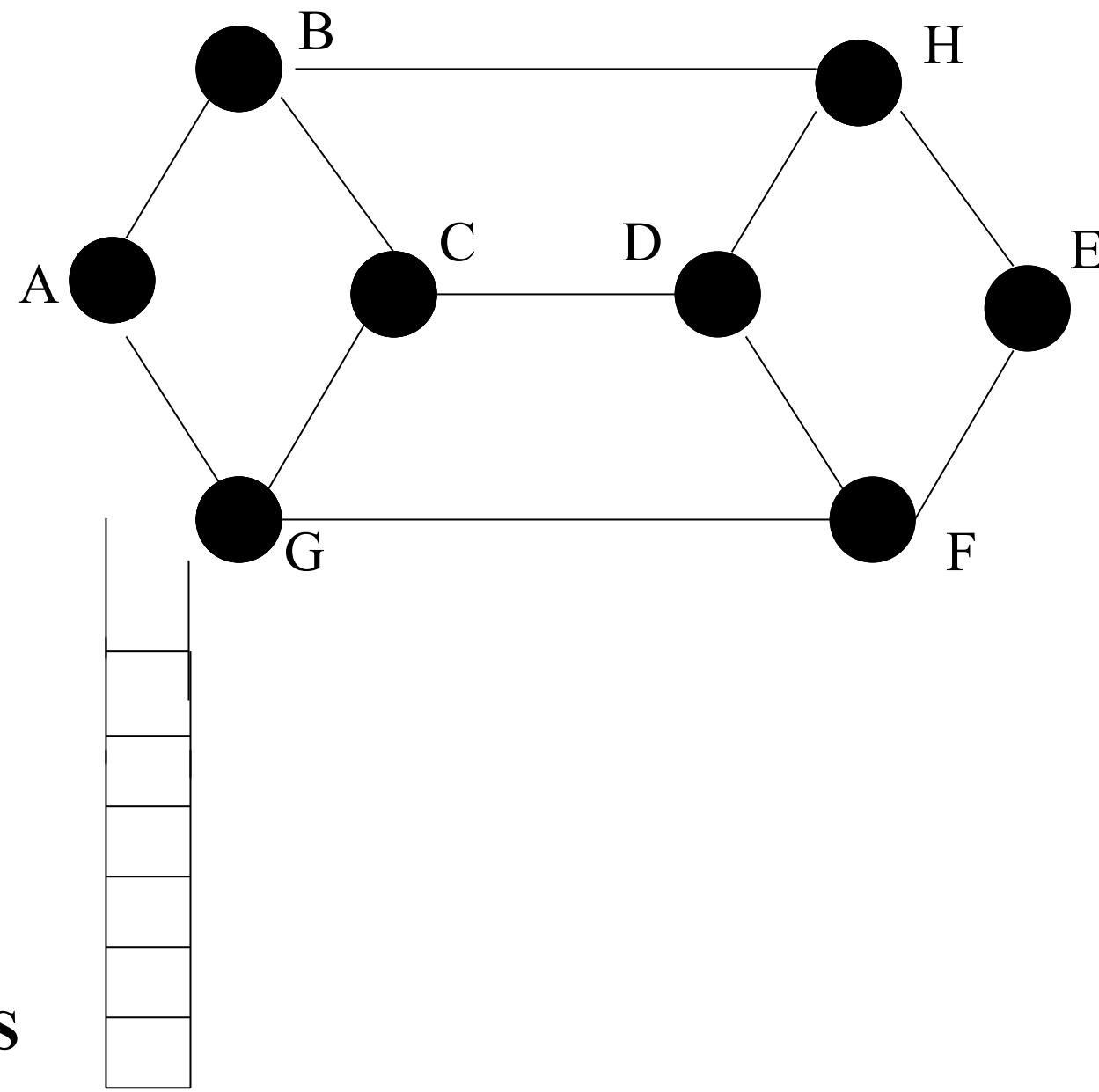












DFS temporizzata

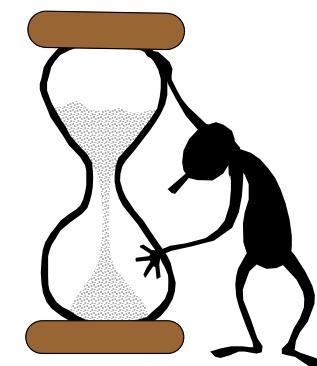
```
DFS-REC( $G, u$ )
 $u.color \leftarrow grigio$ 
 $time \leftarrow time + 1$ 
 $u.d \leftarrow time$ 
for all  $v$  bianco  $\in \text{adj}[u]$  do
     $v.\pi \leftarrow u$ 
    DFS-REC( $G, v$ )
end for
 $u.color \leftarrow nero$ 
 $time \leftarrow time + 1$ 
 $u.f \leftarrow time$ 
```

time è globale ed inizialmente vale 0

Marchiamo:

$u.d$ = tempo di scoperta

$u.f$ = tempo di terminazione



DF – Foresta dopo la DFS

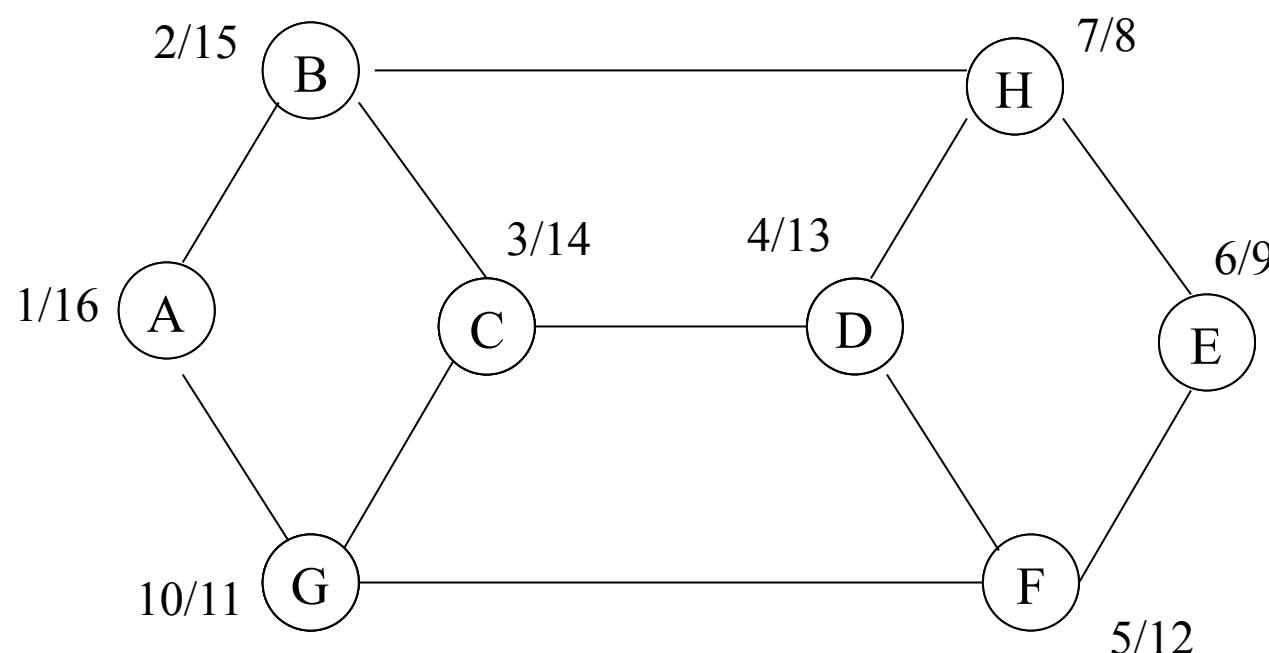
Sia $G_\pi = (V_\pi, E_\pi)$ il grafo delle visite DFS-Graph di $G = (V, E)$ dove:

```
DFS-GRAF(G = (V, E))
for all  $v \in V$  do
     $v.color \leftarrow$  bianco
     $v.\pi \leftarrow nil$ 
end for
 $time \leftarrow 0$ 
for all  $v$  bianco  $\in V$  do
    DFS( $G, v$ )
end for
```

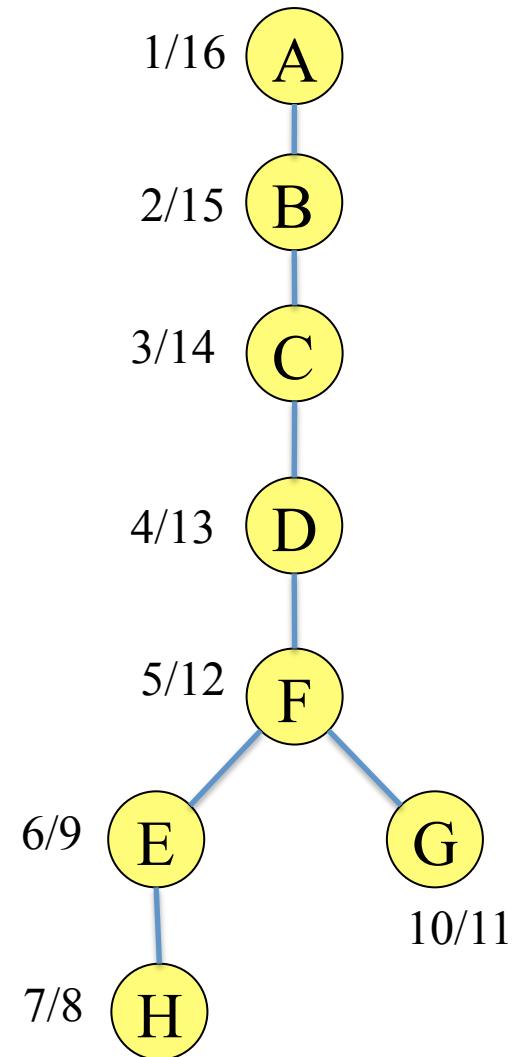
Poiché G non sarà in generale (fortemente) connesso, cosa sarà il grafo G_π ?



DFS temporizzata



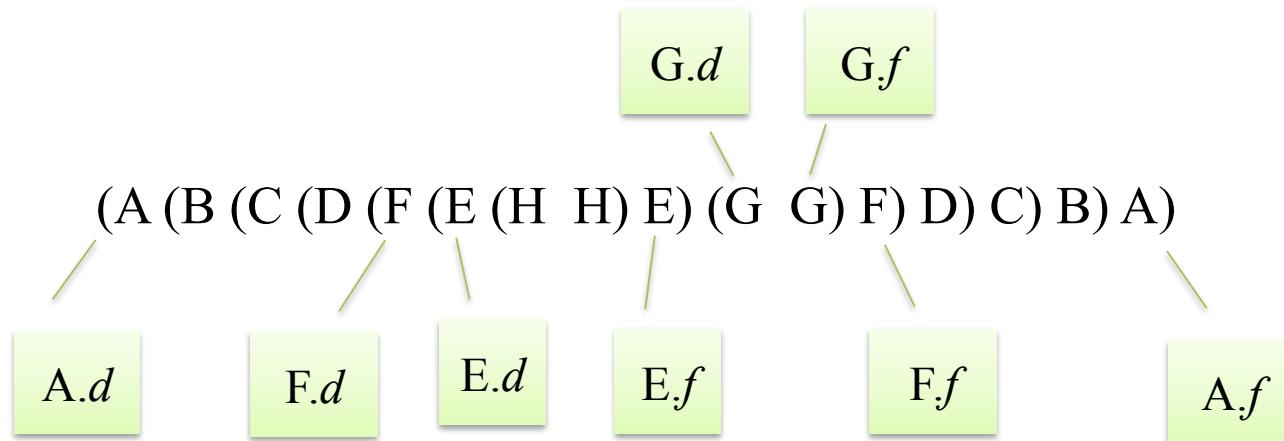
(A (B (C (D (F (E (H H) E) (G G) F) D) C) B) A)



Teorema delle parentesi

Teorema. Al termine di una DFS del grafo $G = (V, E)$ (orientato o non orientato) per ogni coppia di vertici u, v vale esattamente una delle seguenti relazioni:

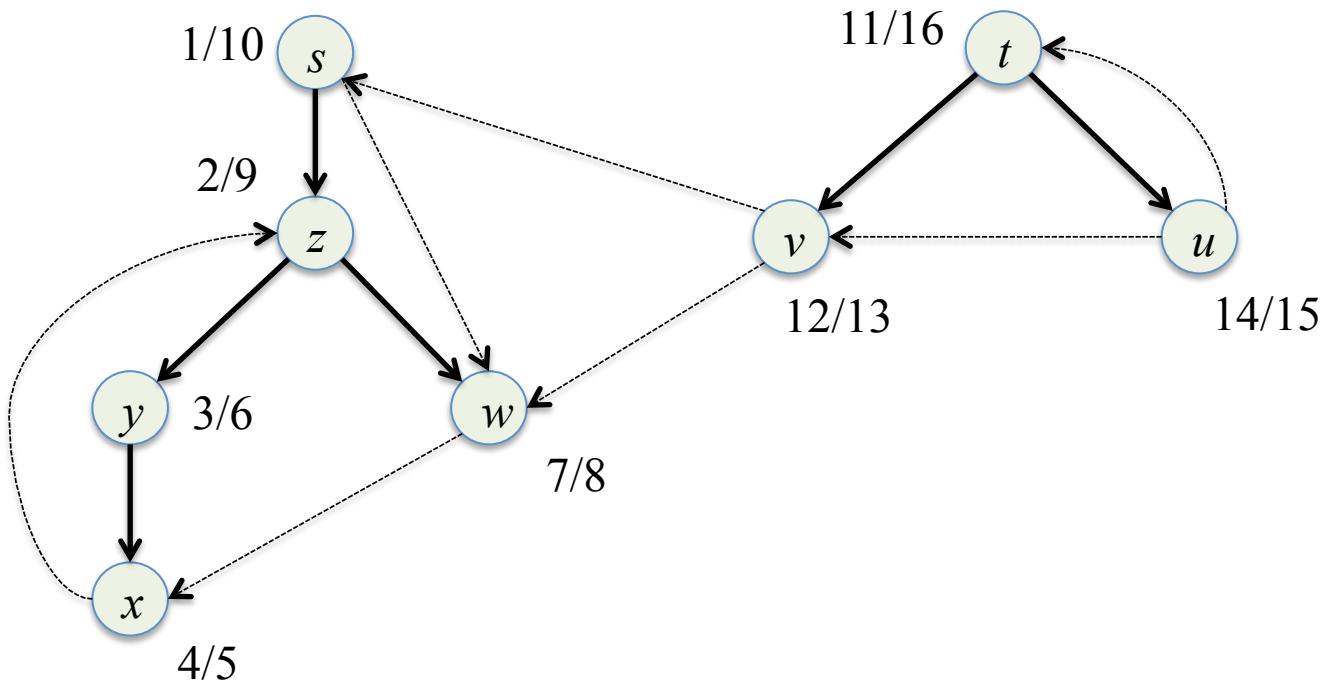
- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
- $[u.d, u.f] \subset [v.d, v.f]$ e più precisamente $v.d < u.d < u.f < v.f$
- $[u.d, u.f] \supset [v.d, v.f]$ e più precisamente $u.d < v.d < v.f < u.f$



Teorema delle parentesi

Corollario. In una DF (foresta generata dalla DFS) del grafo $G = (V, E)$:

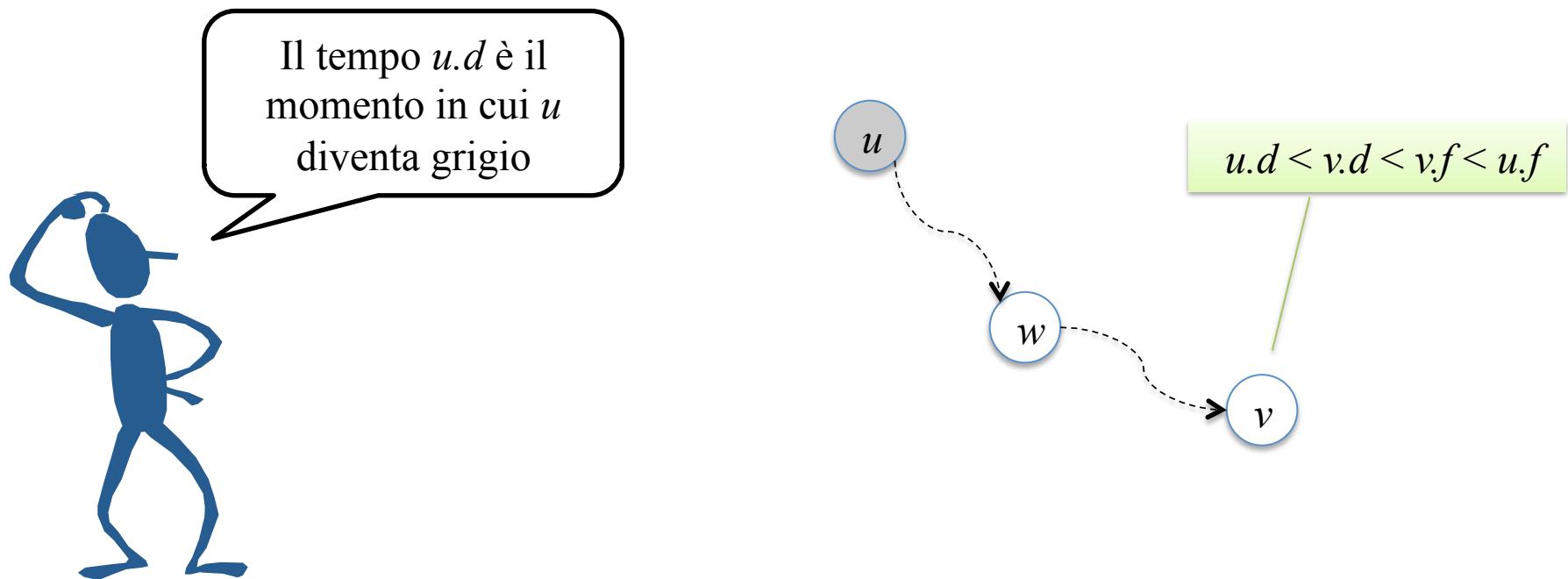
$$v \text{ discendente di } u \Leftrightarrow u.d < v.d < v.f < u.f$$



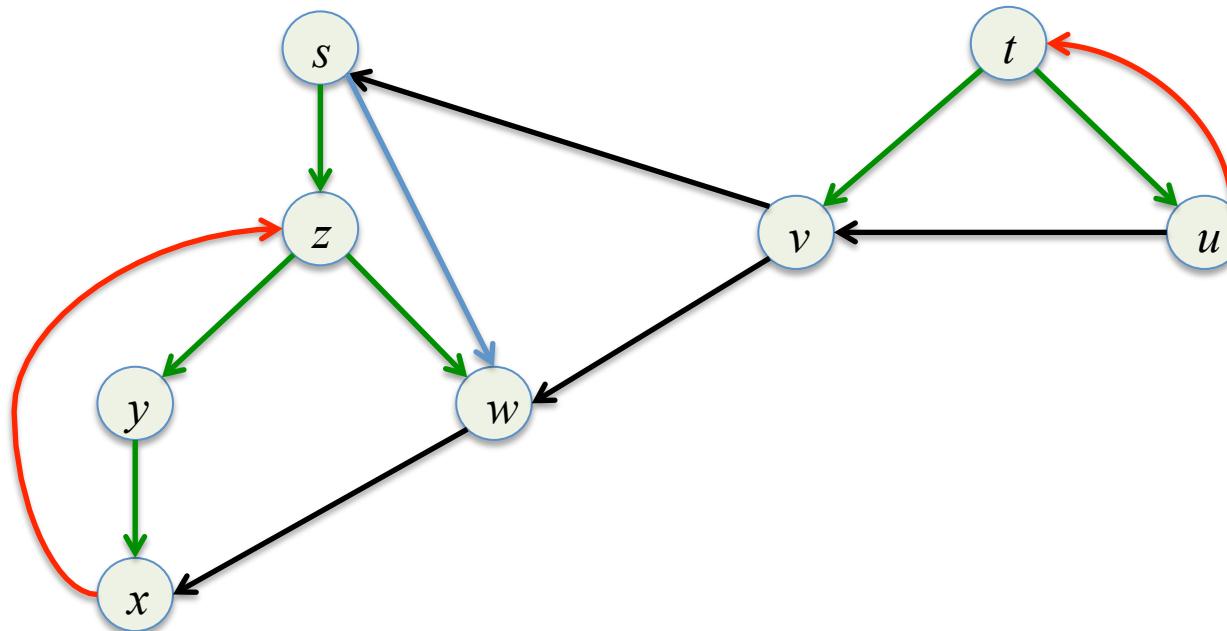
Teorema del cammino bianco

Teorema. In una DF del grafo $G = (V, E)$ (orientato o non):

v discendente di $u \Leftrightarrow$ al tempo $u.d$ esiste un cammino bianco da u a v

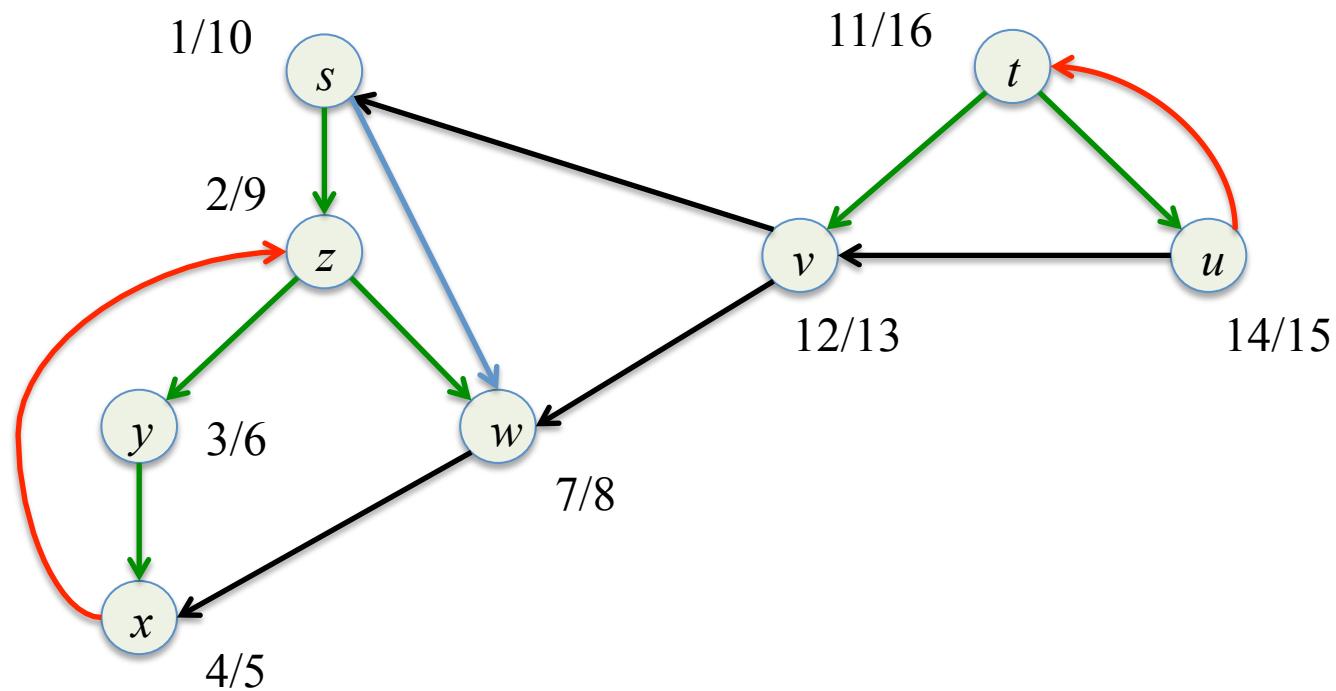


Classificazione degli archi



- **archi d'albero:** sono quelli in E_π
- **archi all'indietro:** (u, v) t.c. v antenato di u in G_π
- **archi in avanti:** (u, v) non in E_π t.c. v discendente di u in G_π
- **archi trasversali:** in tutti gli altri casi

Classificazione degli archi



L'arco (u, v) è

- d'albero o **in avanti** sse $u.d < v.d < v.f < u.f$
- **all'indietro** sse $v.d \leq u.d < u.f \leq v.f$
- **trasversale** sse $v.d < v.f < u.d < u.f$

Ordinamento topologico e componenti connesse

Algoritmi e strutture dati
Lezione 16, a.a. 2016-17

Ugo de'Liguoro, Andras Horvath

Obiettivi

- Studio di proprietà dei grafi applicando la DFS
 - Aciclicità
 - Linearizzazione di grafi aciclici
 - Scomposizione di un grafo orientato nelle sue parti fortemente connesse

DFS temporizzata

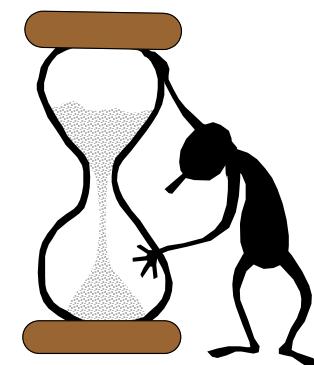
```
DFS-REC( $G, u$ )
 $u.color \leftarrow grigio$ 
 $time \leftarrow time + 1$ 
 $u.d \leftarrow time$ 
for all  $v$  bianco  $\in \text{adj}[u]$  do
     $v.\pi \leftarrow u$ 
    DFS-REC( $G, v$ )
end for
 $u.color \leftarrow nero$ 
 $time \leftarrow time + 1$ 
 $u.f \leftarrow time$ 
```

time è globale ed inizialmente vale 0

Marchiamo:

$u.d$ = tempo di scoperta

$u.f$ = tempo di terminazione



DF – Foresta dopo la DFS

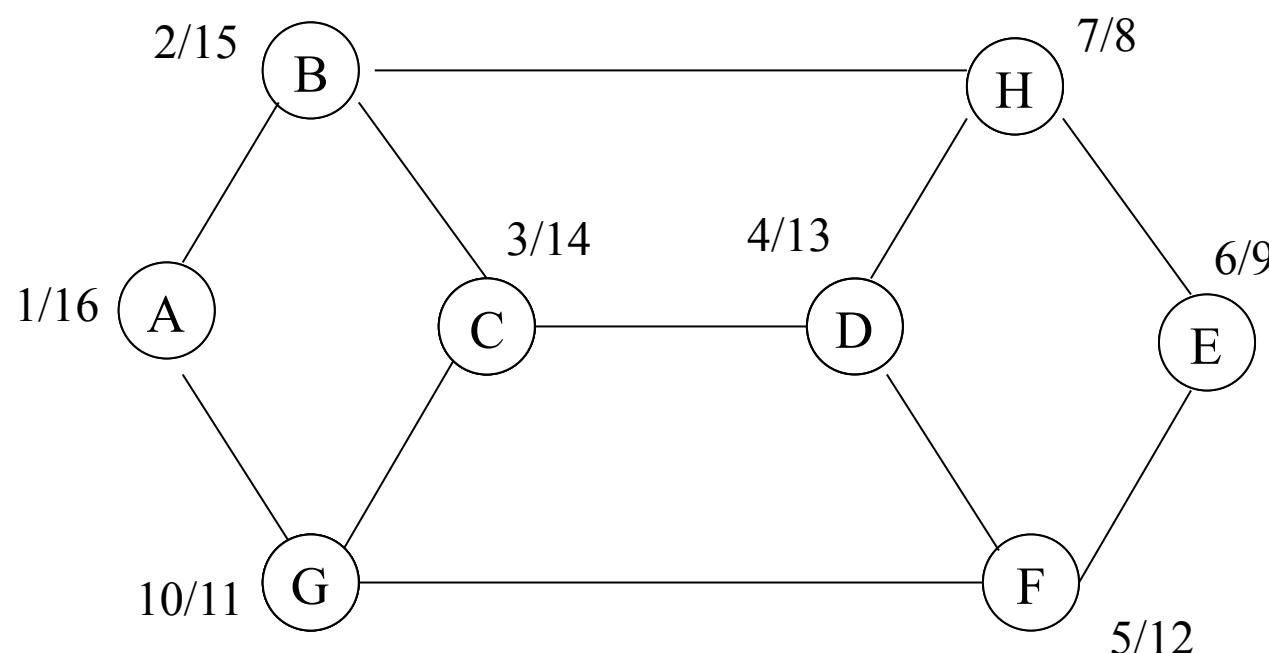
Sia $G_\pi = (V_\pi, E_\pi)$ il grafo delle visite DFS-Graph di $G = (V, E)$ dove:

```
DFS-GRAF(G = (V, E))
for all  $v \in V$  do
     $v.color \leftarrow$  bianco
     $v.\pi \leftarrow nil$ 
end for
 $time \leftarrow 0$ 
for all  $v$  bianco  $\in V$  do
    DFS( $G, v$ )
end for
```

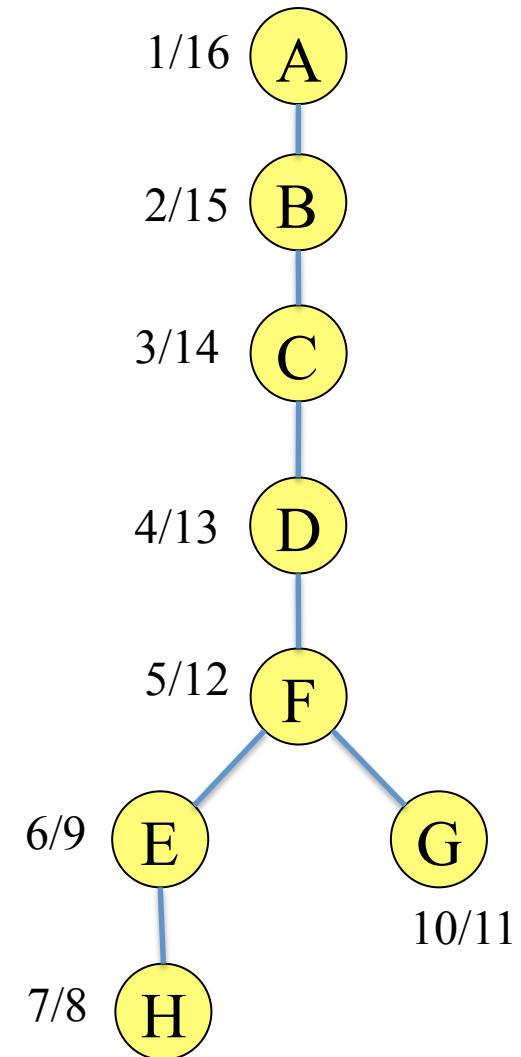
Poiché G non sarà in generale (fortemente) connesso, cosa sarà il grafo G_π ?



DFS temporizzata



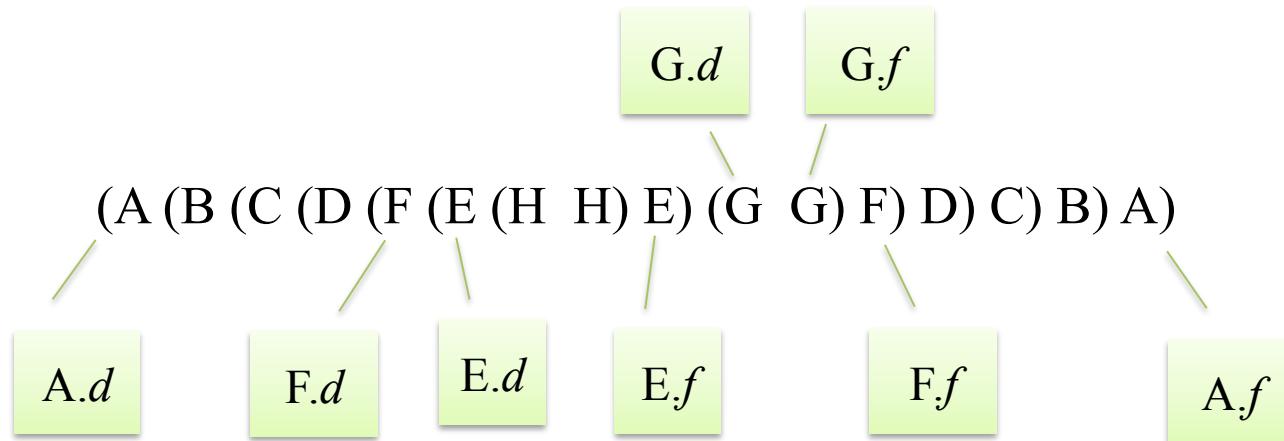
(A (B (C (D (F (E (H H) E) (G G) F) D) C) B) A)



Teorema delle parentesi

Teorema. Al termine di una DFS del grafo $G = (V, E)$ (orientato o non orientato) per ogni coppia di vertici u, v vale esattamente una delle seguenti relazioni:

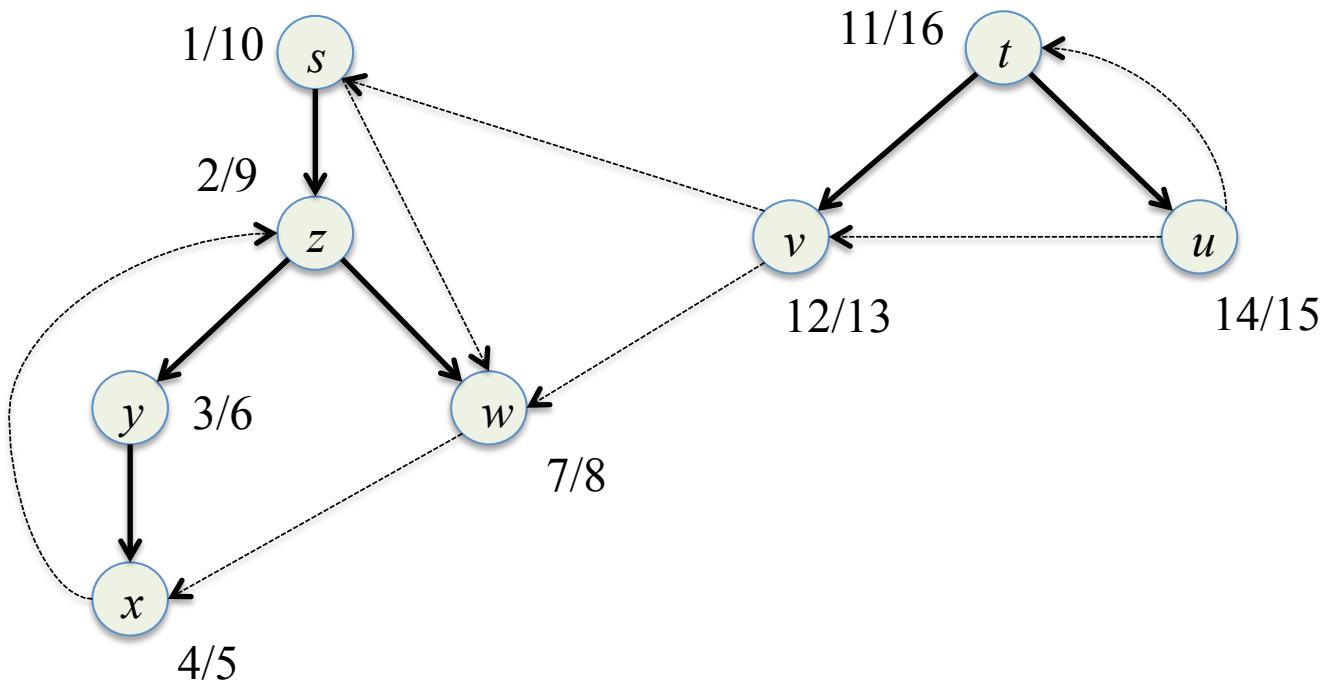
- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
- $[u.d, u.f] \subset [v.d, v.f]$ e più precisamente $v.d < u.d < u.f < v.f$
- $[u.d, u.f] \supset [v.d, v.f]$ e più precisamente $u.d < v.d < v.f < u.f$



Teorema delle parentesi

Corollario. In una DF (foresta generata dalla DFS) del grafo $G = (V, E)$:

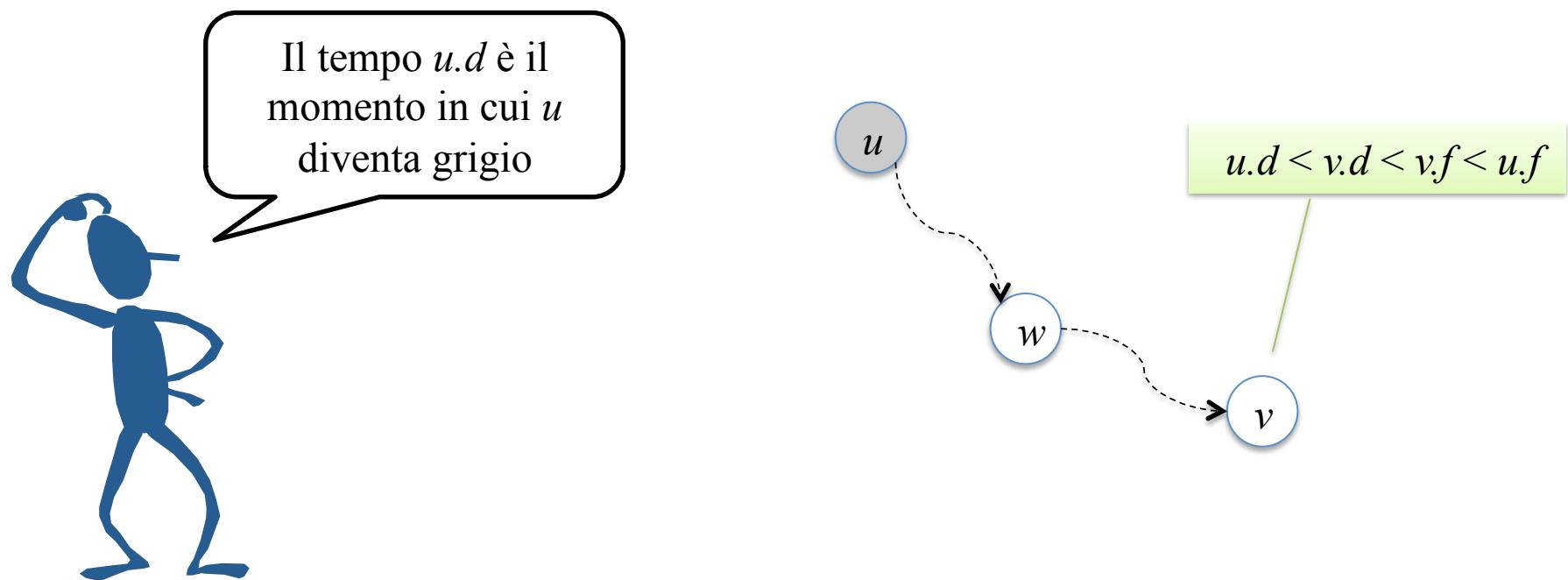
$$v \text{ discendente di } u \Leftrightarrow u.d < v.d < v.f < u.f$$



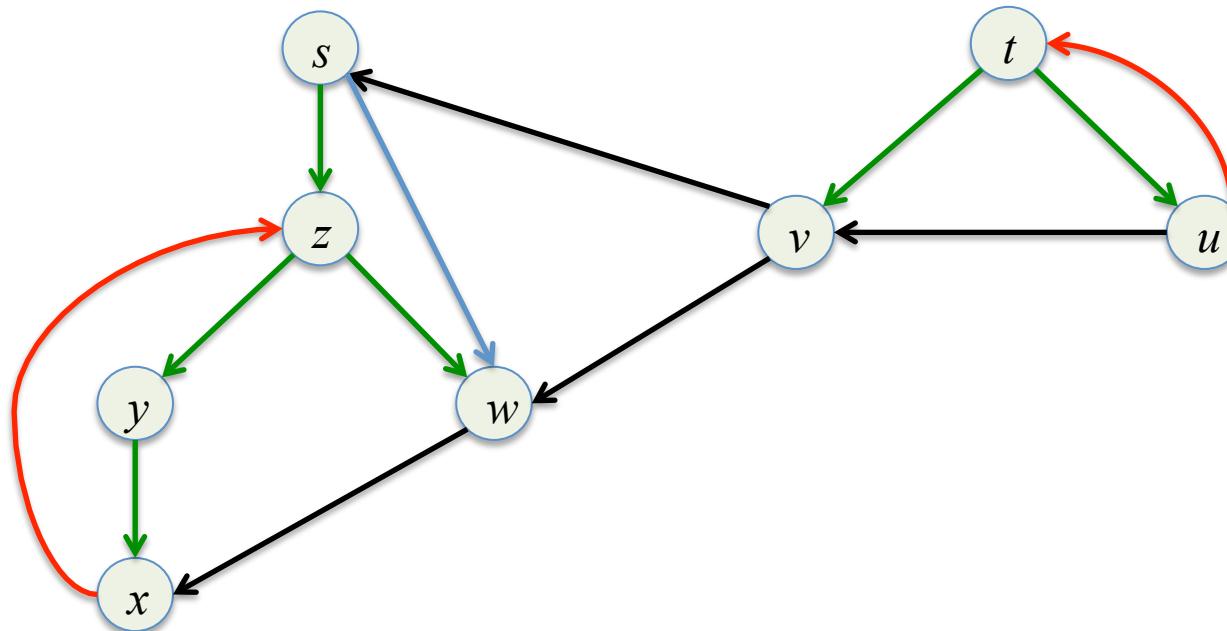
Teorema del cammino bianco

Teorema. In una DF del grafo $G = (V, E)$ (orientato o non):

v discendente di $u \Leftrightarrow$ al tempo $u.d$ esiste un cammino bianco da u a v

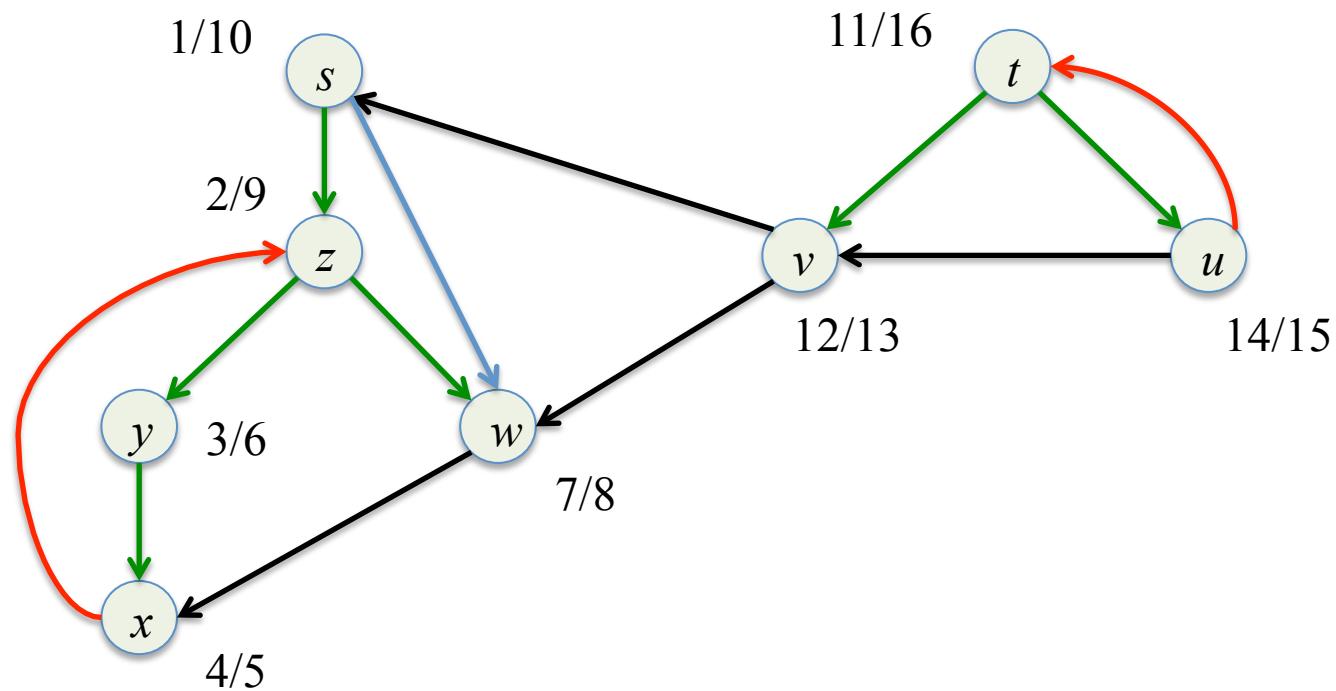


Classificazione degli archi



- **archi d'albero:** sono quelli in E_π
- **archi all'indietro:** (u, v) t.c. v antenato di u in G_π
- **archi in avanti:** (u, v) non in E_π t.c. v discendente di u in G_π
- **archi trasversali:** in tutti gli altri casi

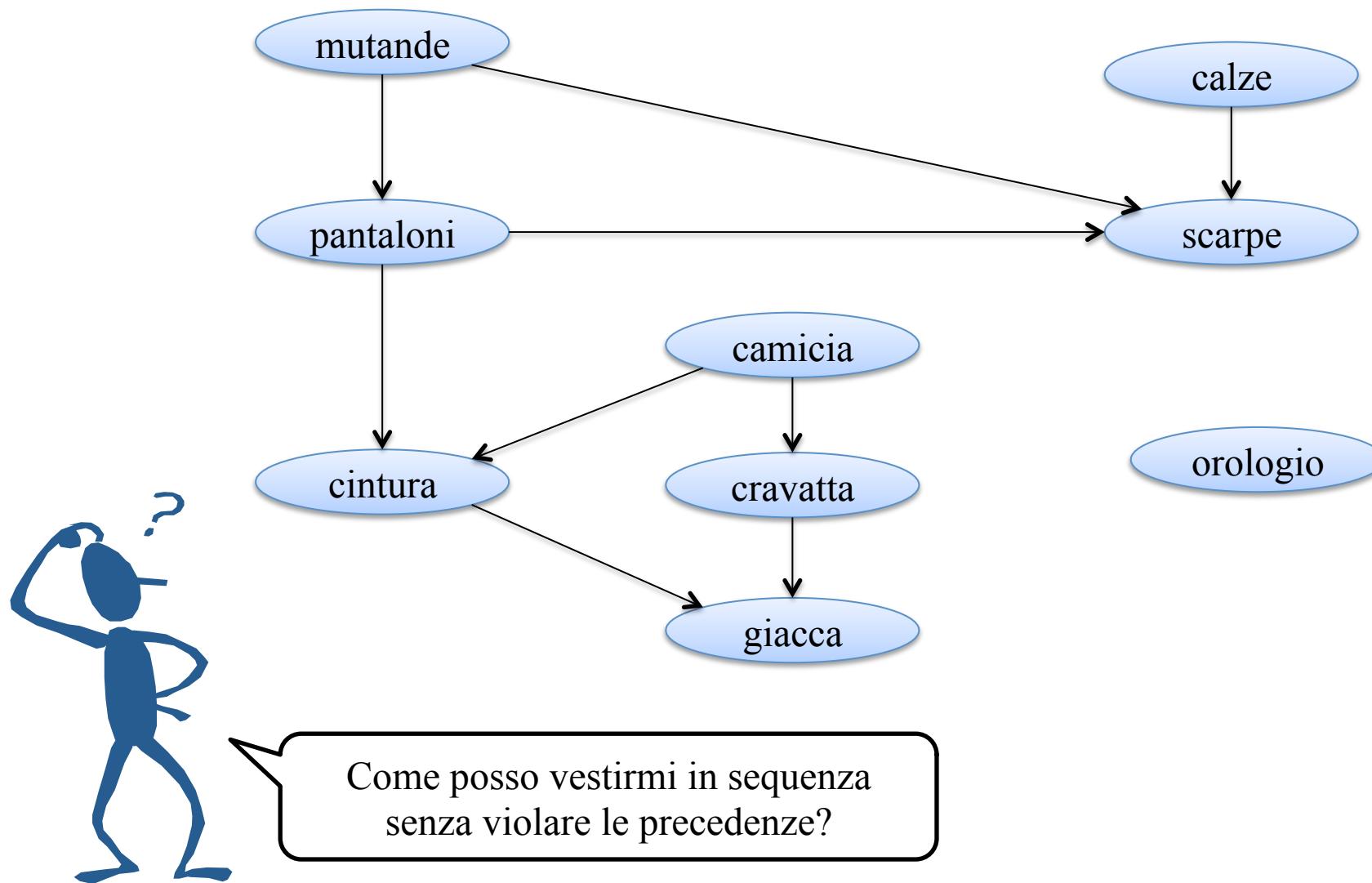
Classificazione degli archi



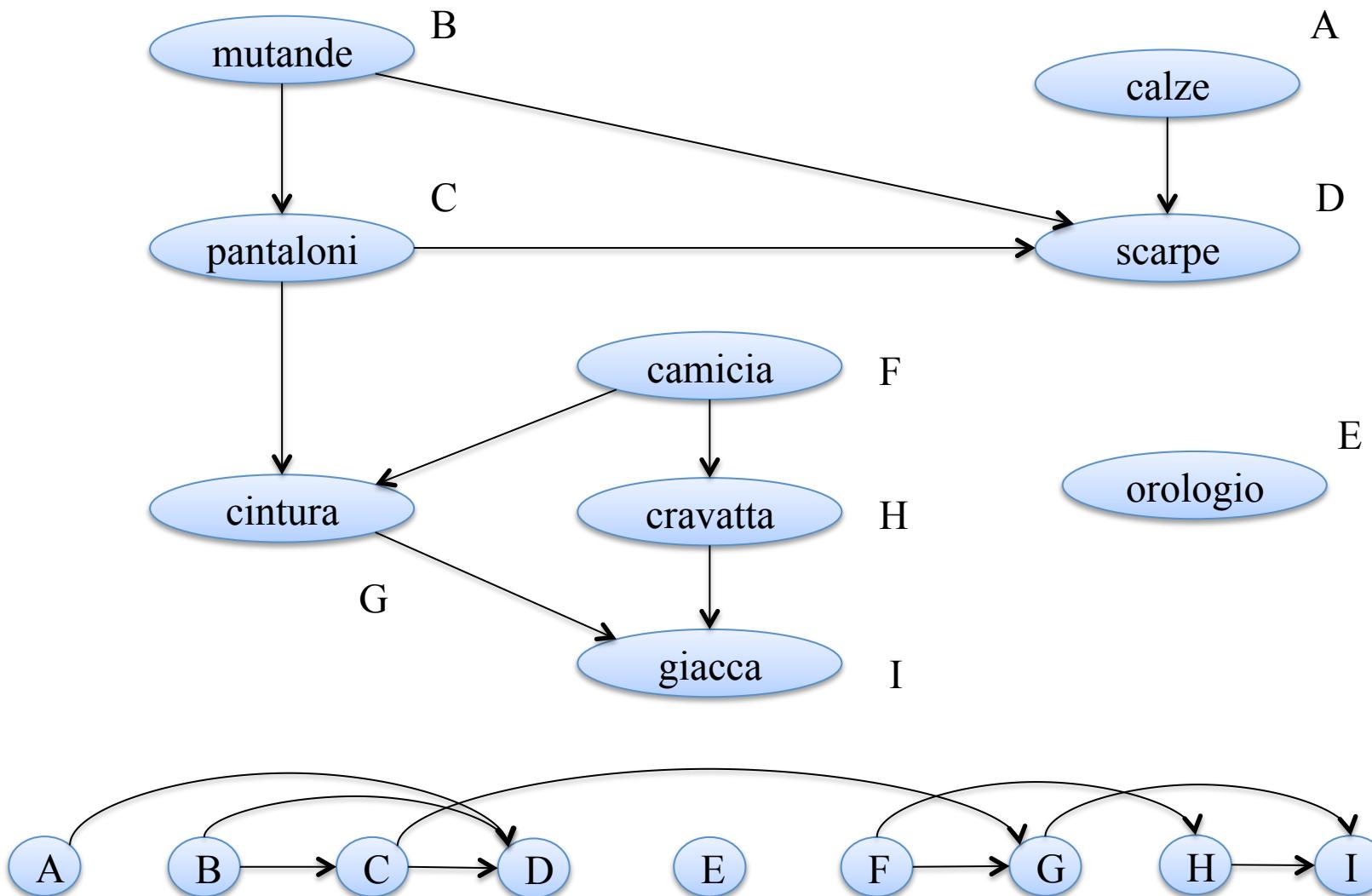
L'arco (u, v) è

- d'albero o **in avanti** sse $u.d < v.d < v.f < u.f$
- **all'indietro** sse $v.d \leq u.d < u.f \leq v.f$
- **trasversale** sse $v.d < v.f < u.d < u.f$

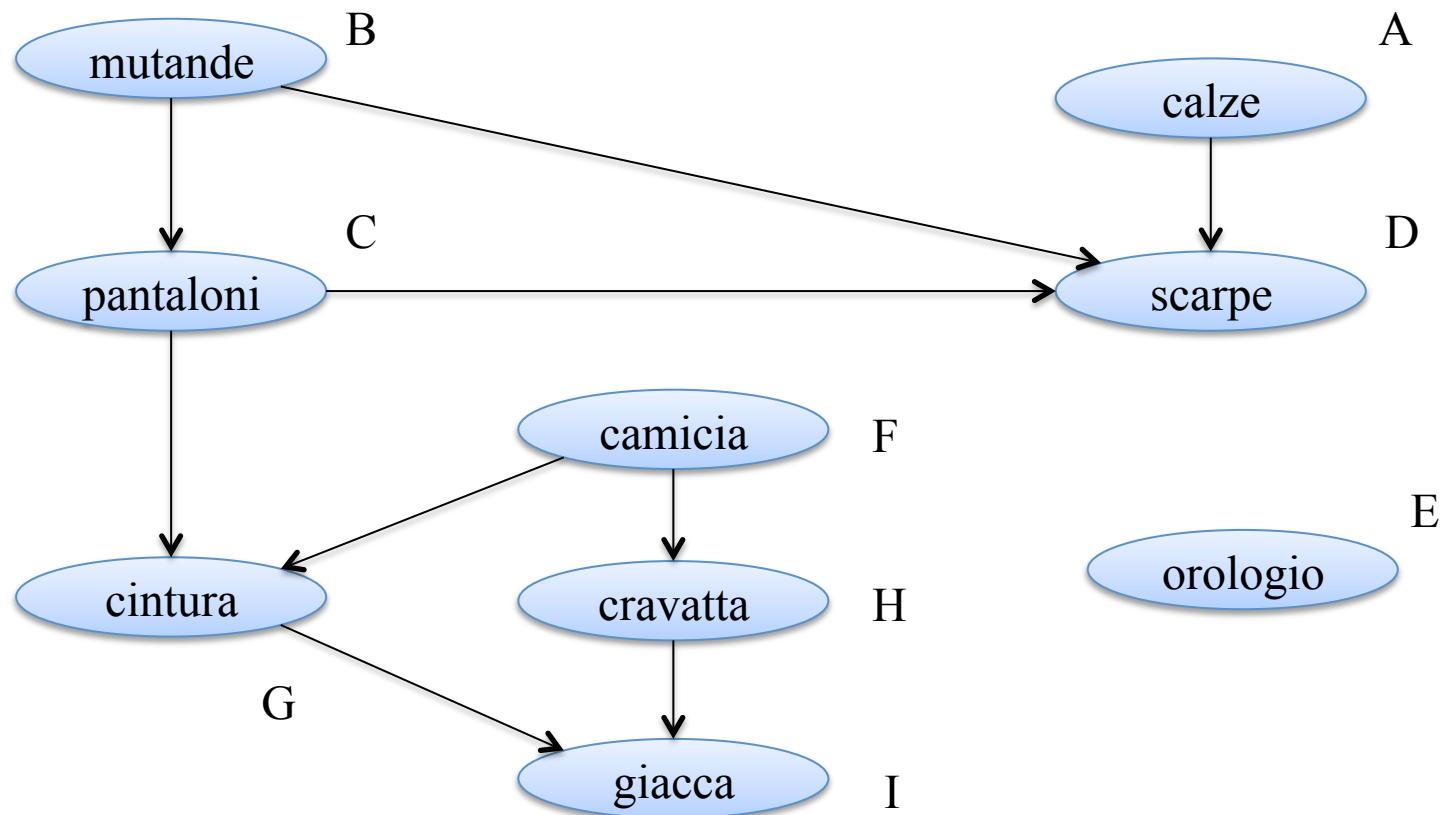
Ordinamento topologico



Ordinamento topologico



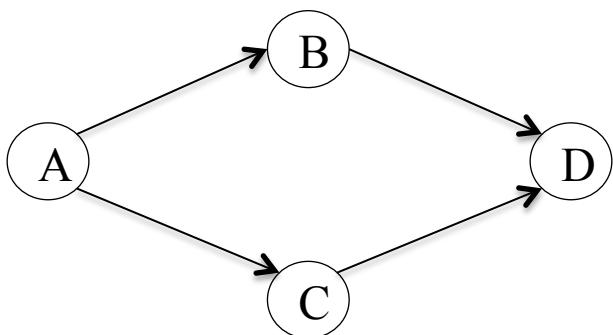
Ordinamento topologico



Definizione. v_1, \dots, v_n è un ordinamento topologico di $G = (V, E)$ orientato se
 $V = \{v_1, \dots, v_n\}$ e $i < j \Rightarrow (v_j, v_i) \notin E$

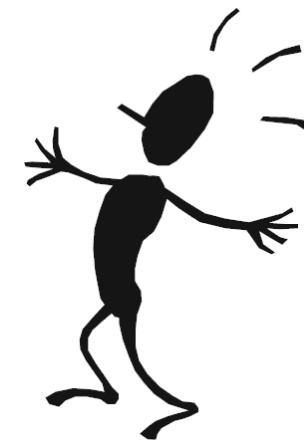
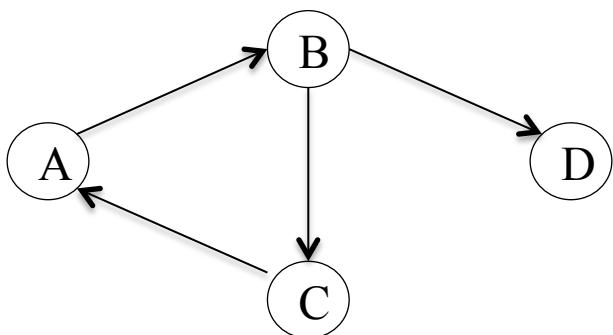
Ordinamento topologico

Ci può essere più di un ordinamento topologico:

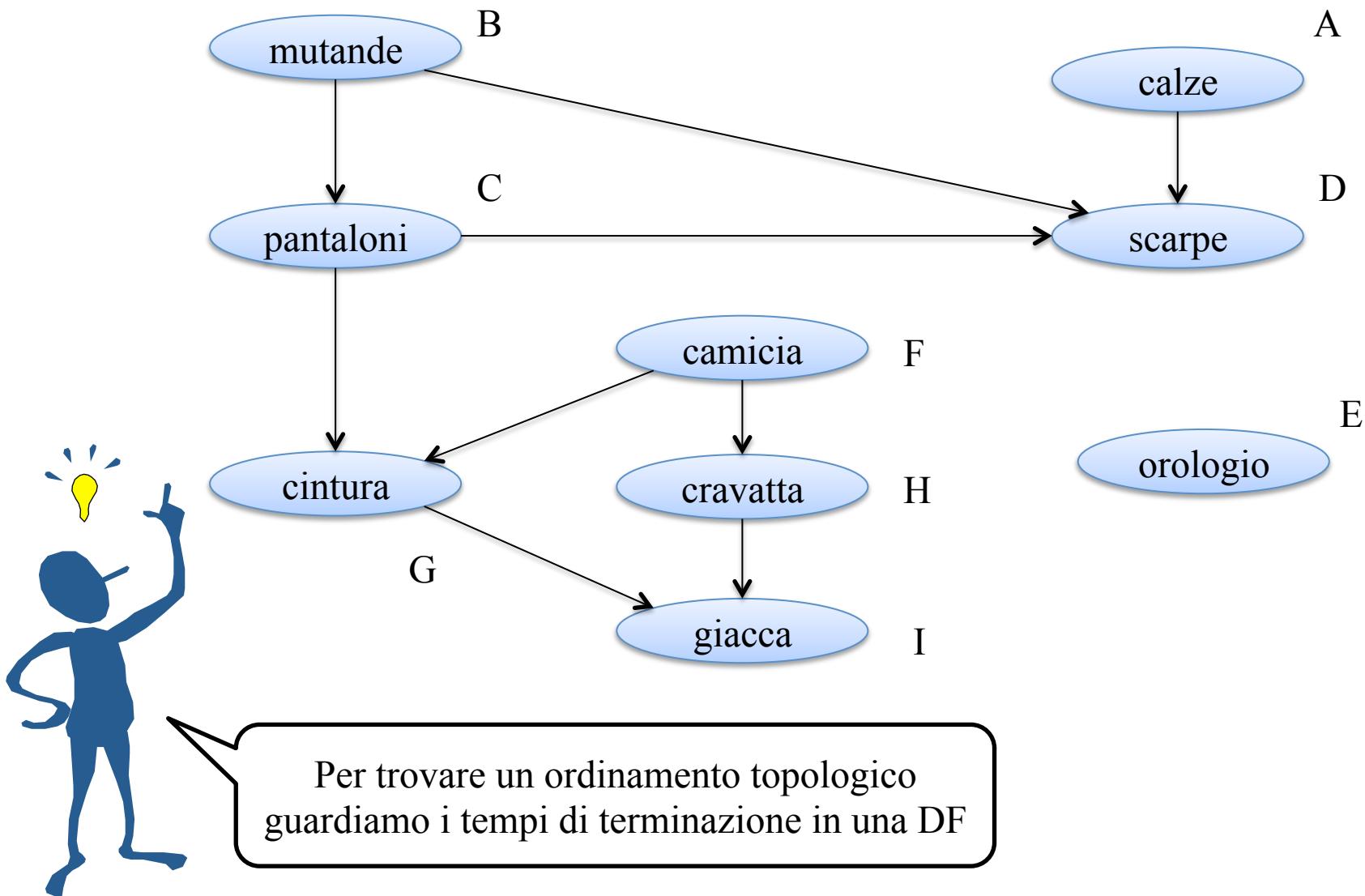


A B C D
A C B D

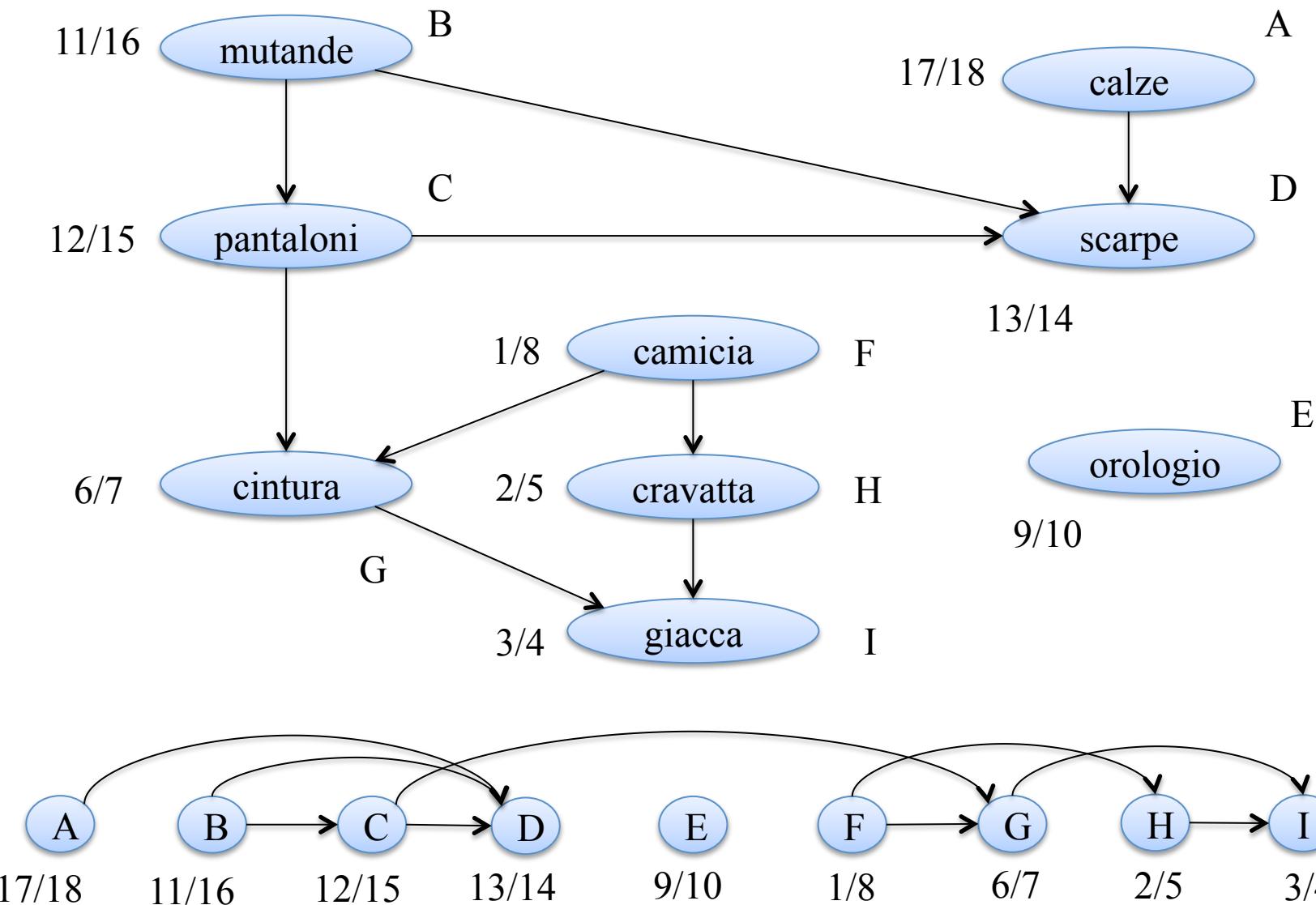
Ma nessuno se il grafo è ciclico:



Ordinamento topologico

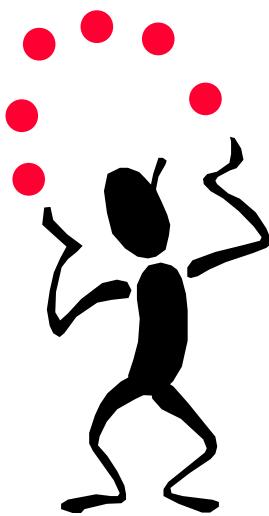


Ordinamento topologico



Topological-Sort

```
TOPOLOGICAL-SORT( $G$ )
 $L \leftarrow$  lista vuota di vertici
INIZIALIZZA( $G$ )
for  $\forall u \in V$  do
    if  $u.color = bianco$  then
        DFS-TOPOLOGICAL( $G, u, L$ )
    end if
end for
restituisci  $L$ 
```



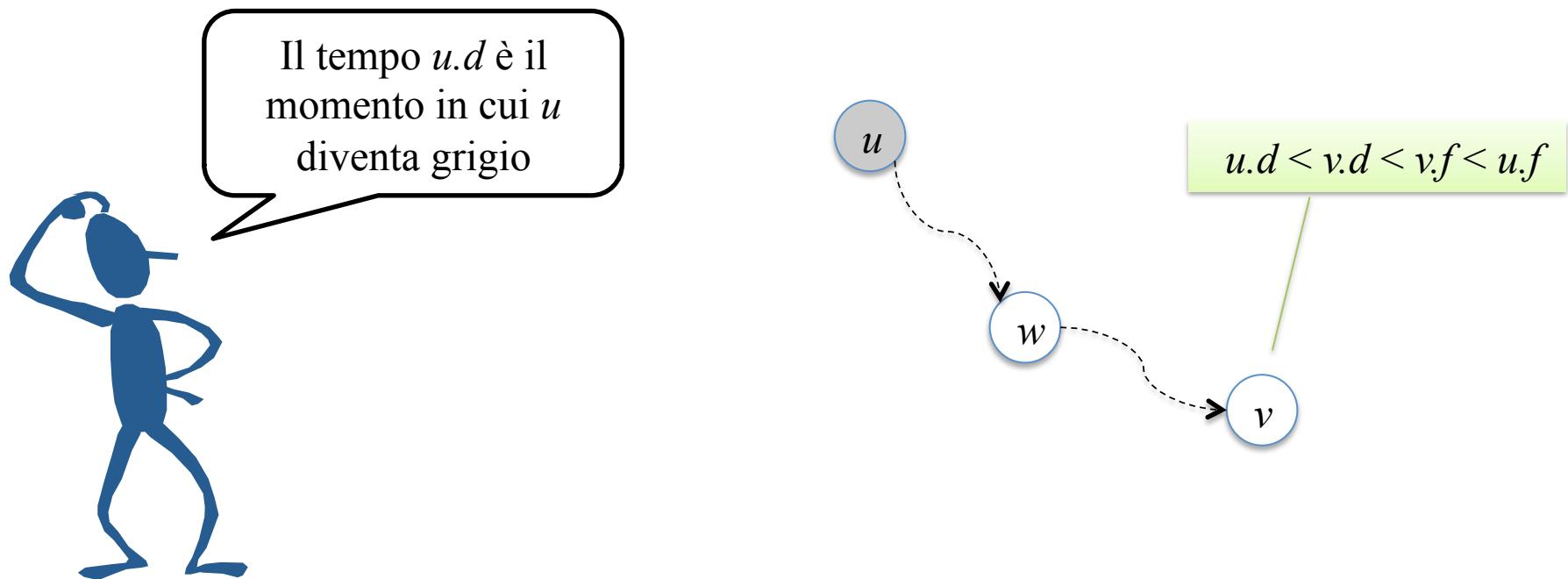
```
DFS-TOPOLOGICAL( $G, s, L$ )
 $s.color \leftarrow grigio$ 
 $s.d \leftarrow time$ 
 $time \leftarrow time + 1$ 
for  $\forall v : v \text{ è bianco ed è } \in \text{adj}[s]$  do
     $v.\pi = s$ 
    DFS-TOPOLOGICAL( $G, v, L$ )
end for
 $s.color \leftarrow nero$ 
 $s.f \leftarrow time$ 
 $time \leftarrow time + 1$ 
inserisci  $s$  in testa ad  $L$ 
```

Se corretto Topological-Sort è ottimo perché
 $\Theta(|V| + |E|)$

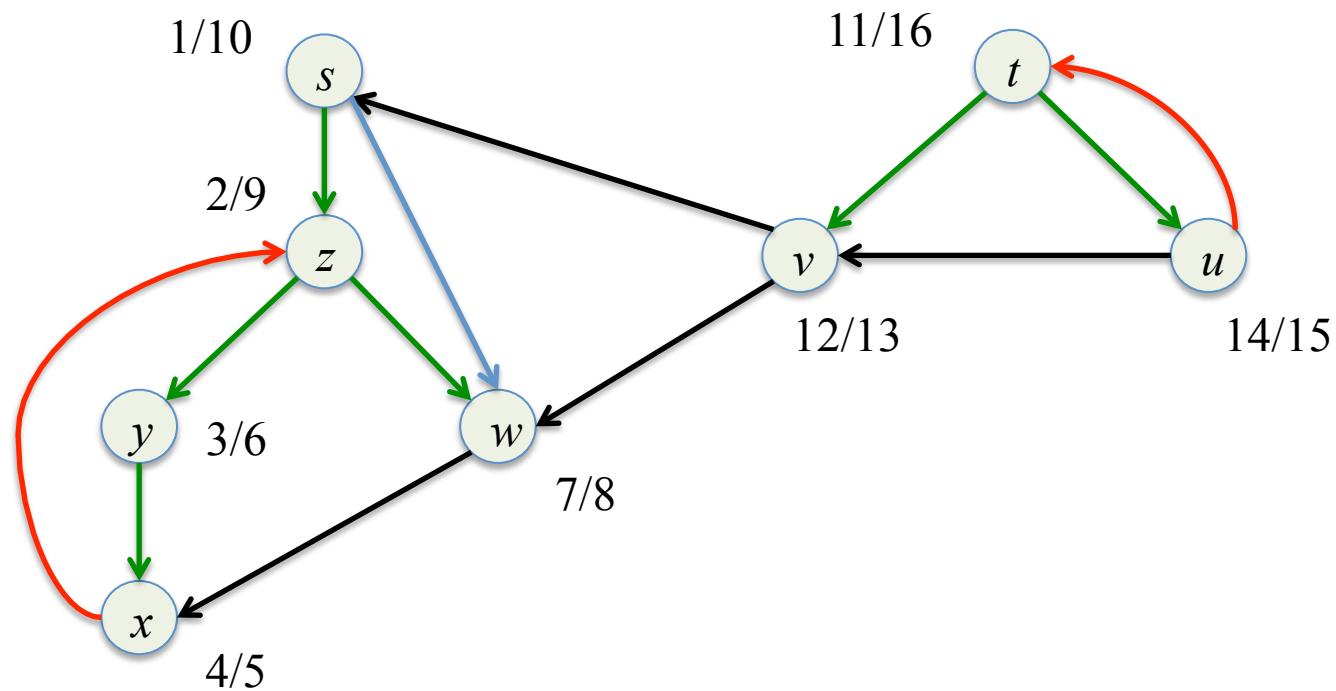
Teorema del cammino bianco

Teorema. In una DF del grafo $G = (V, E)$ (orientato o non):

v discendente di $u \Leftrightarrow$ al tempo $u.d$ esiste un cammino bianco da u a v



Classificazione degli archi



L'arco (u, v) è

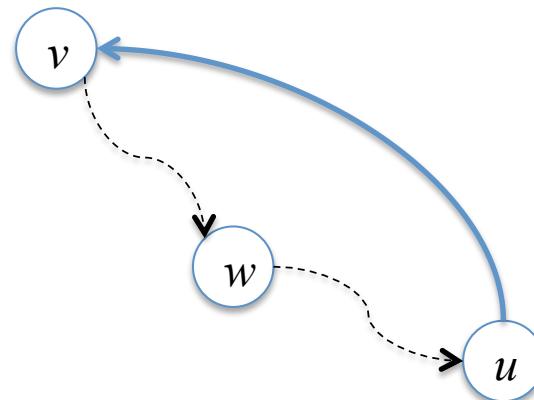
- d'albero o **in avanti** sse $u.d < v.d < v.f < u.f$
- **all'indietro** sse $v.d \leq u.d < u.f \leq v.f$
- **trasversale** sse $v.d < v.f < u.d < u.f$

Aciclicità ed archi all'indietro

Lemma. Un grafo $G = (V, E)$ (orientato o non) è aciclico se e solo se una DF non genera archi all'indietro o equivalentemente:

$$\exists (u, v) \text{ all'indietro in una DF di } G \Leftrightarrow G \text{ ciclico}$$

(\Rightarrow) L'arco (u, v) è **all'indietro** sse $v.d \leq u.d < u.f \leq v.f$



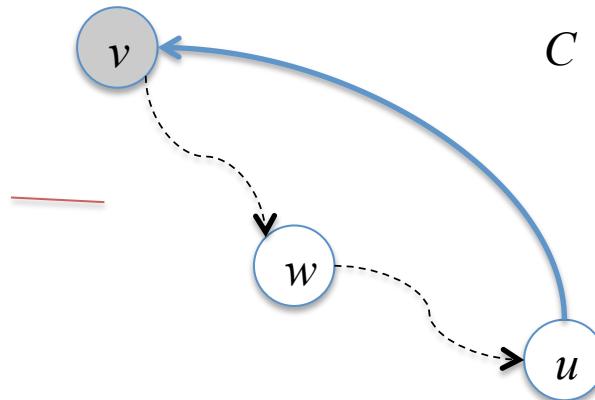
Aciclicità ed archi all'indietro

Lemma. Un grafo $G = (V, E)$ (orientato o non) è aciclico se e solo se una DF non genera archi all'indietro o equivalentemente:

$$\exists (u, v) \text{ all'indietro in una DF di } G \Leftrightarrow G \text{ ciclico}$$

(\Leftarrow) Sia C un ciclo in G e $(u, v) \in C$ con $v.d$ minimo

Per il teorema del cammino bianco



Allora $v.d < u.d < u.f < v.f$ ossia (u, v) è all'indietro

DFS-Topological-Sort

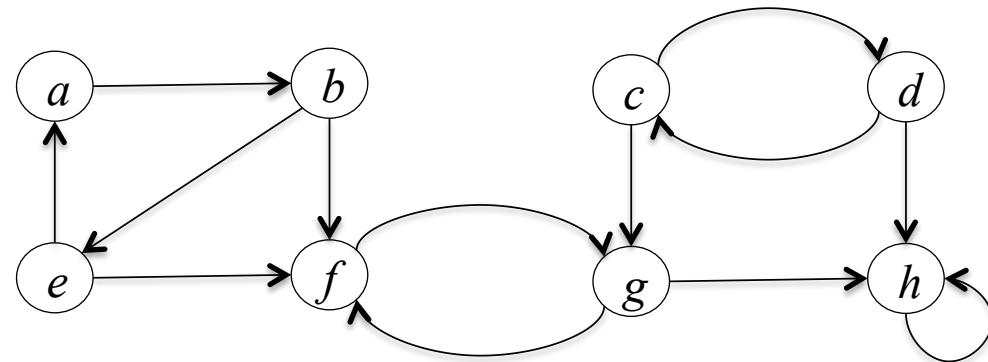
Teorema. Se $G = (V, E)$ è orientato aciclico allora Topological-Sort(G) produce un ordinamento topologico

Tesi: se $(u, v) \in E$ allora $v.f < u.f$

Per il teorema delle parentesi abbiamo quattro possibilità:

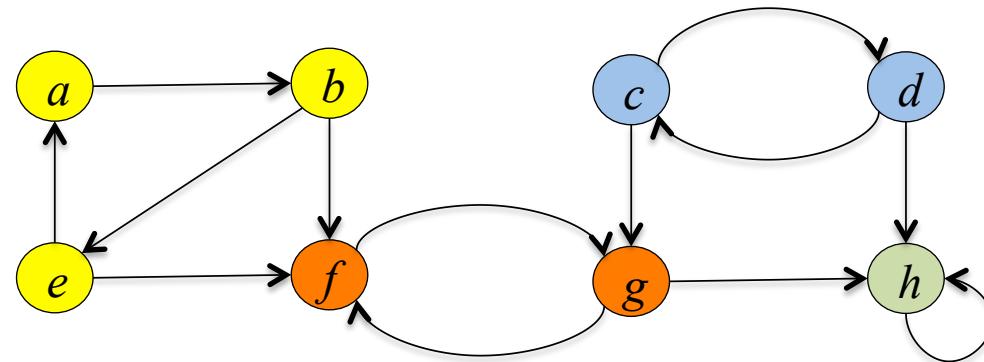
1. $v.d < v.f < u.d < u.f$: OK
2. $u.d < u.f < v.d < v.f$: allora u è nero quando v è bianco 
3. $u.d < v.d < v.f < u.f$: OK
4. $v.d < u.d < u.f < v.f$: allora (u, v) è all'indietro 

Parti fortemente connesse



In quali parti si può
raggiungere un vertice
da qualunque altro?

Parti fortemente connesse



Definizione. In $G = (V, E)$ orientato:

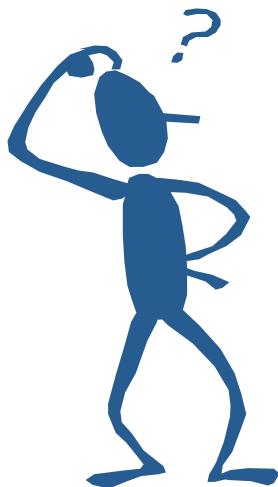
- $u \sim v \Leftrightarrow v$ raggiungibile da u
- $C \subseteq V$ è una *componente fortemente connessa (cfc)* se
 1. massimale (non si può estendere)
 2. se $u, v \in C$ allora $u \leftrightarrow v$ (cioè $u \sim v$ e $v \sim u$)

Parti fortemente connesse

Per ogni $u \in V$:

- $D(u) = \{v \in V \mid u \rightsquigarrow v \text{ in } G\}$ discendenti di u
- $A(u) = \{v \in V \mid v \rightsquigarrow u \text{ in } G\}$ antecedenti di u

$$C(u) = \text{la cfc cui appartiene } u = D(u) \cap A(u)$$



Con una DFS posso calcolare
 $D(u)$, ma come fare per $A(u)$?

Parti fortemente connesse

Per ogni $u \in V$:

- $D(u) = \{v \in V \mid u \rightsquigarrow v \text{ in } G\}$ discendenti di u
- $A(u) = \{v \in V \mid v \rightsquigarrow u \text{ in } G\}$ antecedenti di u

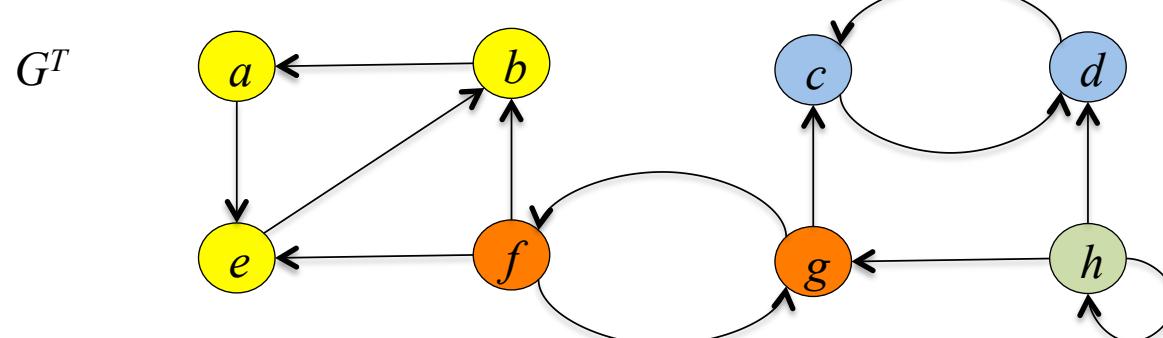
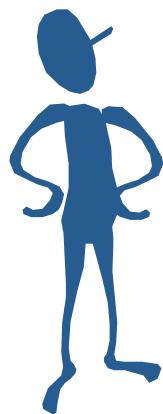
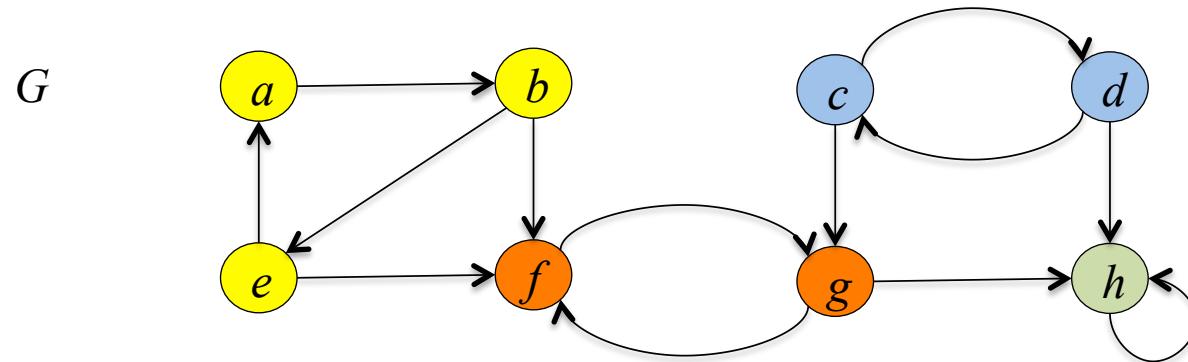
$$C(u) = \text{la cfc cui appartiene } u = D(u) \cap A(u)$$



Considero una DFS su $G^T = (V, E^T)$ dove

$$E^T = \{(v, u) \mid (u, v) \in E\}$$

Un algoritmo basato sulla DFS



Un algoritmo basato sulla DFS

STRONGLY-CONNECTED-COMPONENTS-NAIVE($G, = (V, E)$)

$G^T \leftarrow (V, E^T)$

for all $u \in V$ **do**

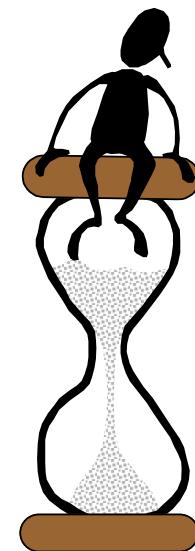
$D(u) \leftarrow \text{DFS}(G, u)$

$A(u) \leftarrow \text{DFS}(G^T, u)$

$C(u) \leftarrow D(u) \cap A(u)$

end for

Questo algoritmo richiede tempo
 $O(|V|) \cdot O(|V| + |E|) = O(|V|^2 + |V| \cdot |E|)$



Un algoritmo basato sulla DFS

STRONGLY-CONNECTED-COMPONENTS($G, = (V, E)$)

DFS-GRAF(G)

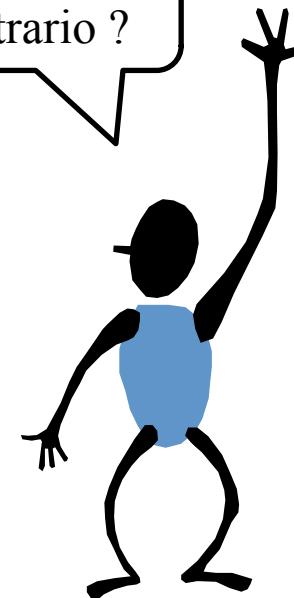
calcola $G^T = (V, E^T)$

DFS-GRAF(G^T)

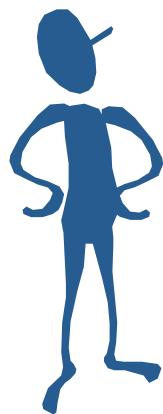
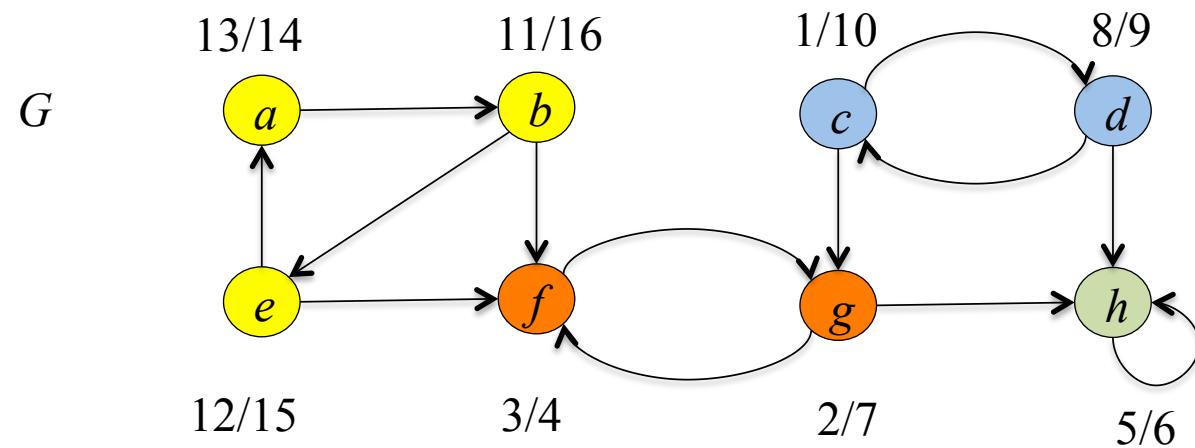


Se u e v appartengono allo stesso albero in $\text{DF}(G)$ e in $\text{DF}(G^T)$ allora $C(u) = C(v)$

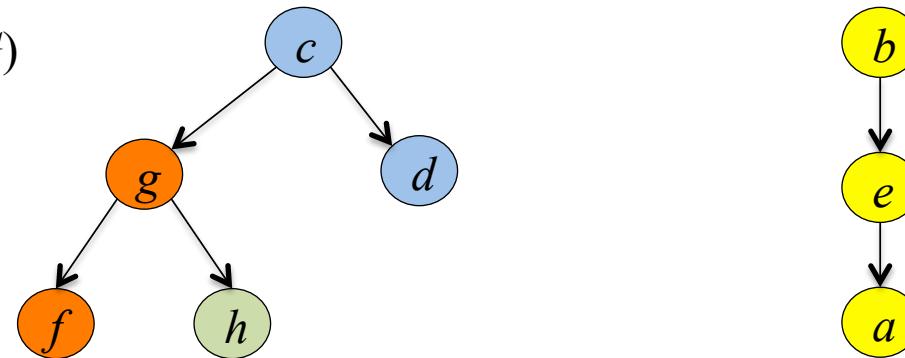
E' vero anche il contrario ?



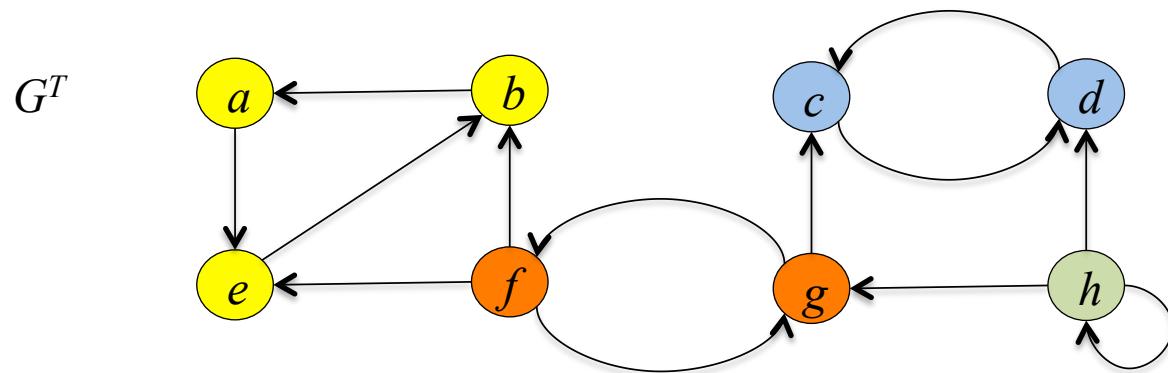
Un algoritmo basato sulla DFS



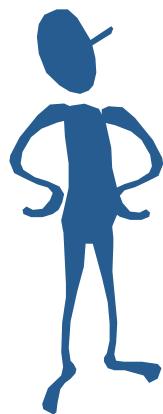
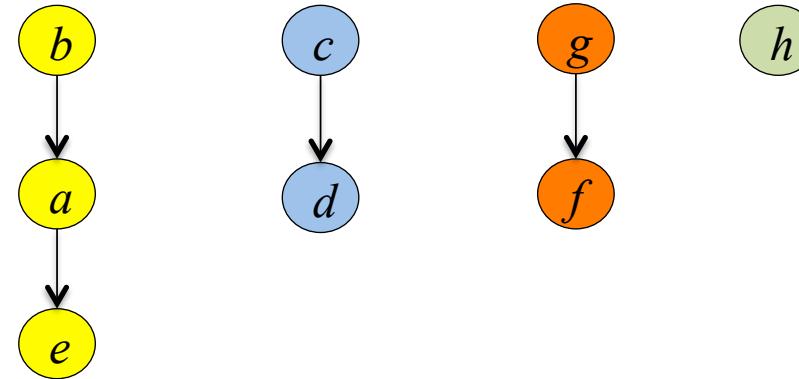
$\text{DF}(G)$



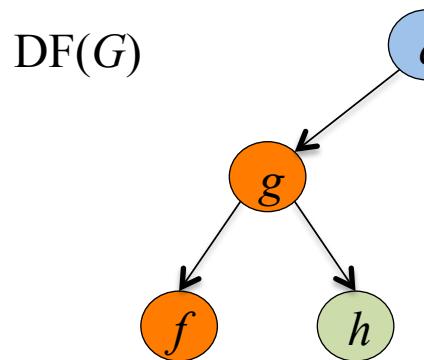
Un algoritmo basato sulla DFS



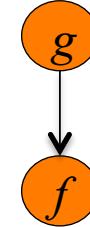
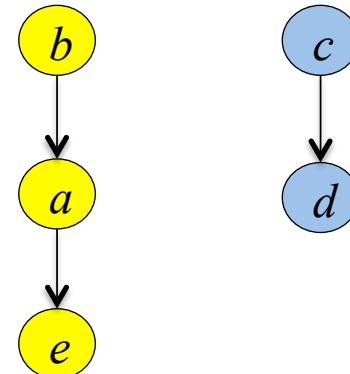
$\text{DF}(G^T)$



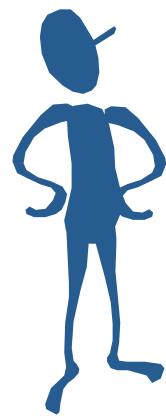
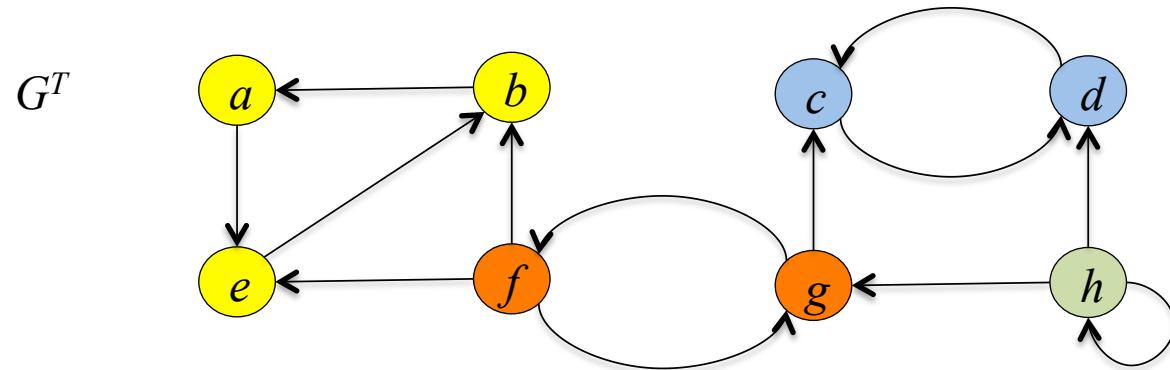
Un algoritmo basato sulla DFS



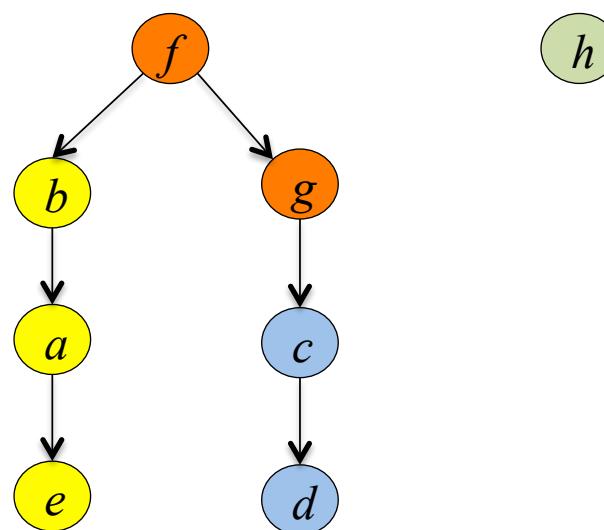
DF(G^T)



Un algoritmo basato sulla DFS



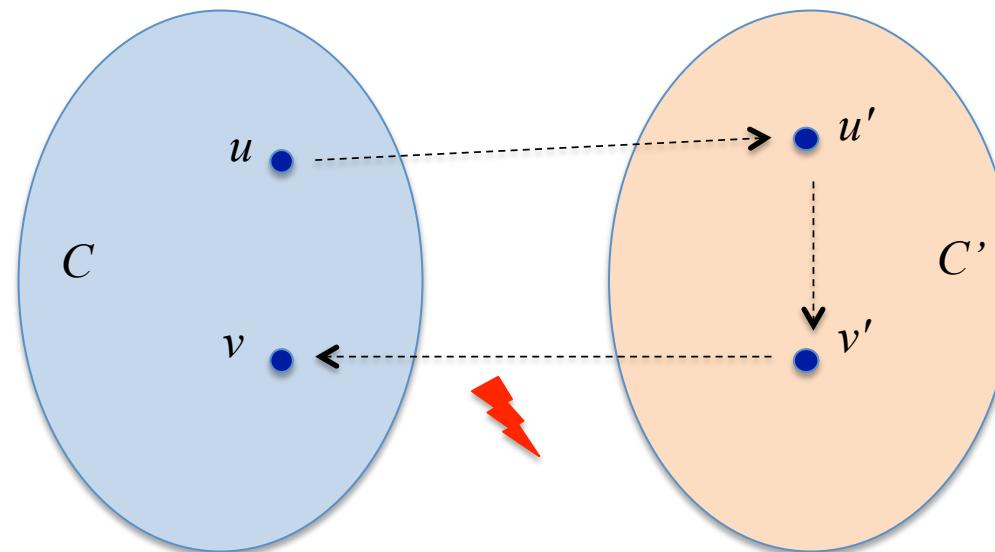
$\text{DF}'(G^T)$



DFS versus cfc

Lemma 1. Siano C, C' due cfc distinte e $u, v \in C, u', v' \in C'$.

Se $u \rightsquigarrow u'$ allora non può essere $v' \rightsquigarrow v$.



DFS versus cfc

Fissata $\text{DF}(G)$ ed i relativi tempi di scoperta/terminazione, sia C una cfc:

- $C.d = \min\{u.d \mid u \in C\}$
- $C.f = \max\{u.f \mid u \in C\}$

Lemma 2. Siano C, C' due cfc distinte e $u \in C$ e $v \in C'$:

$$(u, v) \in E \Rightarrow C.f > C'.f$$

DFS versus cfc

Lemma 2. Siano C, C' due cfc distinte e $u \in C$ e $v \in C'$:

$$(u, v) \in E \Rightarrow C.f > C'.f$$

Caso $C.d < C'.d$: sia $x \in C$ con $x.d$ minimo allora

- x è la radice dell'albero in $\text{DF}(G)$ che include C
- al tempo $x.d$ tutti i vertici in $C \cup C'$ sono bianchi (salvo x)
- $(u, v) \in E$ implica per qualche $w \in C'$

$$x \rightsquigarrow u \rightarrow v \rightsquigarrow w$$

- tutti i vertici in $C \cup C'$ sono discendenti di x in $\text{DF}(G)$ dunque

$$x.f = C.f > C'.f$$

DFS versus cfc

Lemma 2. Siano C, C' due cfc distinte e $u \in C$ e $v \in C'$:

$$(u, v) \in E \Rightarrow C.f > C'.f$$

Caso $C.d > C'.d$: sia $y \in C'$ con $y.d$ minimo allora

- y è la radice dell'albero in $\text{DF}(G)$ che include C'
- al tempo $y.d$ tutti i vertici in $C \cup C'$ sono bianchi (salvo y)
- $(u, v) \in E$ implica NON esiste $x \in C$ t.c. $y \rightsquigarrow x$ (lemma 1)
- per ogni $x \in C$ dunque

$$C.f \geq x.f > y.f = C'.f$$

DFS versus cfc

Lemma 2. Siano C, C' due cfc distinte e $u \in C$ e $v \in C'$:

$$(u, v) \in E \Rightarrow C.f > C'.f$$

Corollario. Siano C, C' due cfc e $u \in C$ e $v \in C'$:

$$(u, v) \in E^T \Rightarrow C.f \leq C'.f$$

dove si ha $C.f < C'.f$ se C, C' sono distinte.

Teorema della visita “giusta”

Teorema. Fissata $\text{DF}(G)$ con i relativi tempi, se $\text{DF}(G^T)$ è generata considerando i vertici in ordine decrescente rispetto ai tempi di terminazione, allora gli alberi di $\text{DF}(G^T)$ sono le cfc di G .

Sia C la cfc con tempo massimo in $\text{DF}(G)$, allora per ogni u in C :

$$(u, v) \in E^T \Rightarrow Cf \leq C(v)f \quad \text{per il corollario}$$

Quindi deve essere $C = C(v)$.

Se $u \in C$ è t.c. $u.f$ sia massimo allora l’albero T in $\text{DF}(G^T)$ radicato in u è t.c.

$$T = A_G(u) = C$$

dove $A_G(u)$ sono gli antecedenti di u in G .

Di qui il teorema segue per
induzione sul numero degli alberi
in $\text{DF}(G^T)$

Correttezza di Strongly-Connected-Components

Corollario. L'algoritmo Strongly-Connected-Components è corretto ed ha complessità $\Theta(|V| + |E|)$

STRONGLY-CONNECTED-COMPONENTS($G = (V, E)$)

DFS-GRAF(G)

$G^T \leftarrow (V, E^T)$

DFS-GRAF(G^T)

scegliendo i vertici in ordine decrescente rispetto ai tempi f della prima visita



Algoritmi greedy

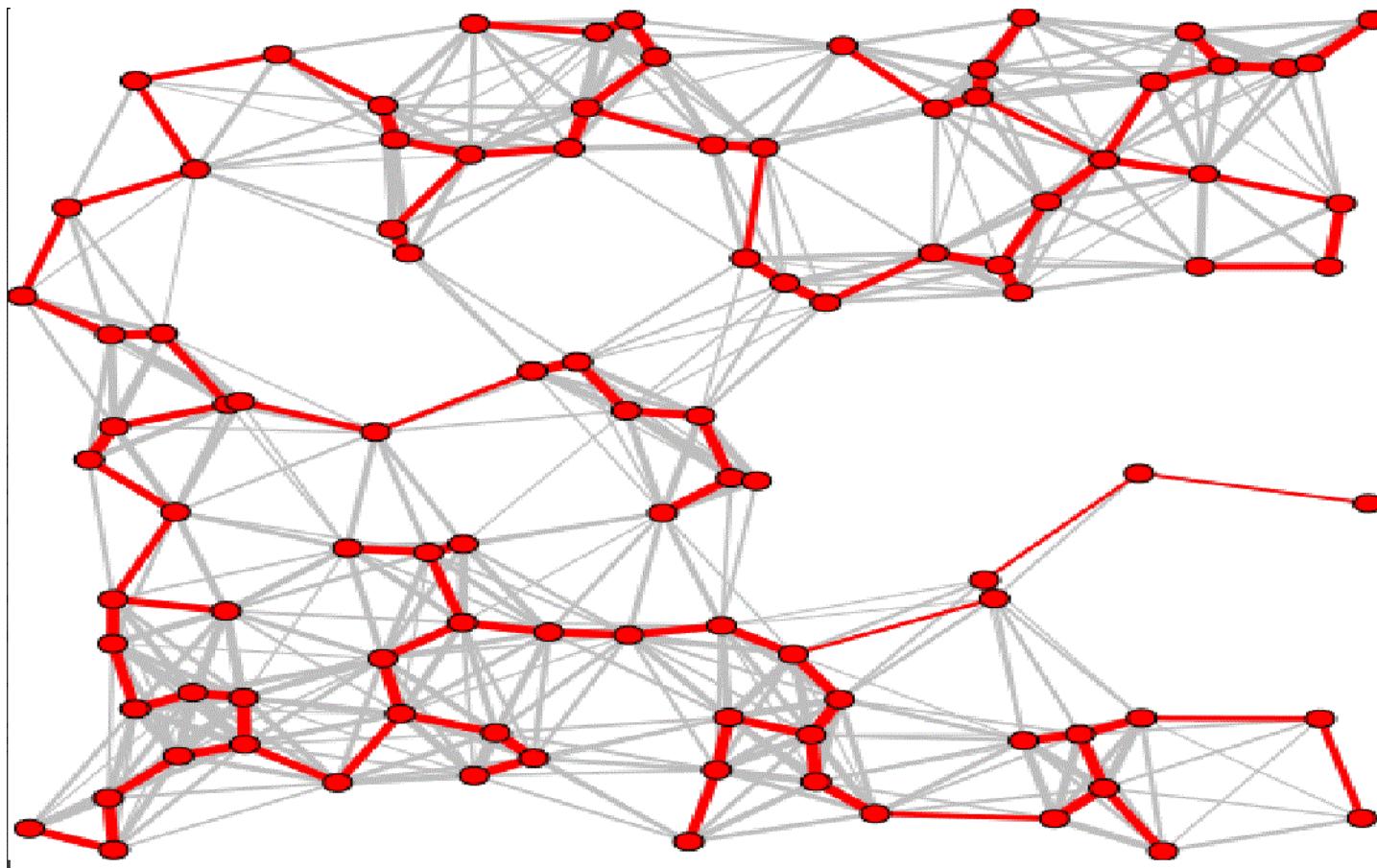
Alberi di copertura minima

Algoritmi e strutture dati
Lezione 17, a.a. 2016-17

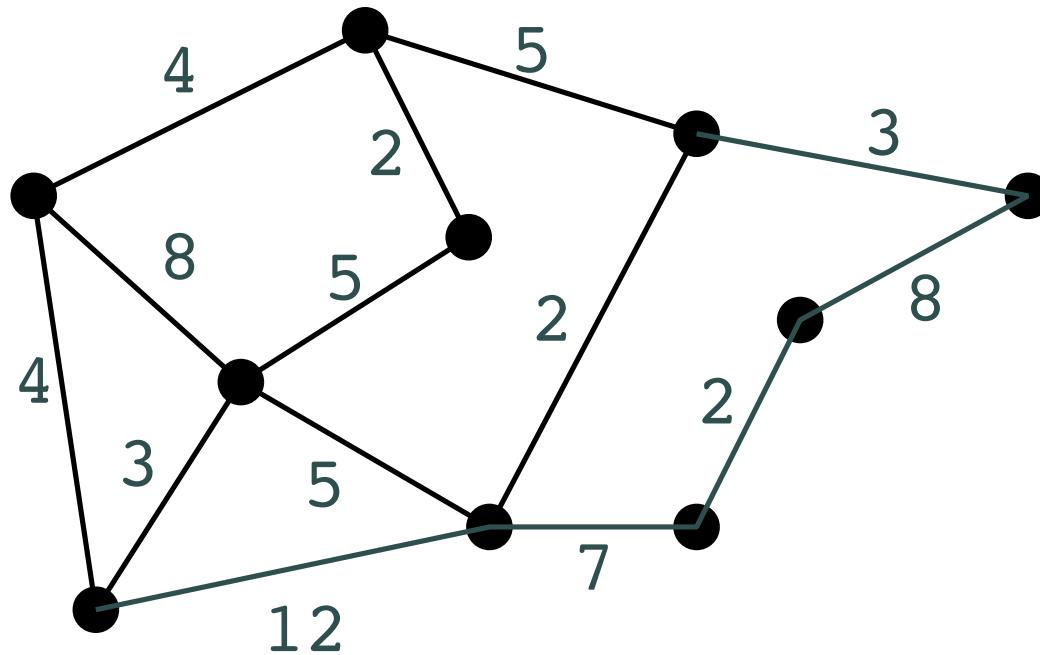
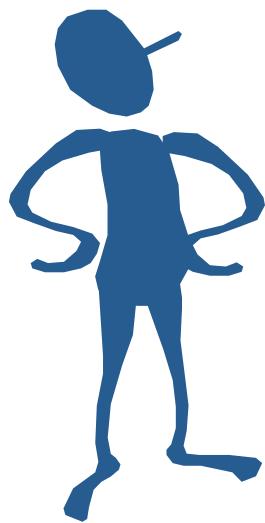
Obiettivi

- Proprietà di grafi pesati
- Algoritmi greedy

Un problema di rete



Albero di copertura minima



Albero di copertura minima

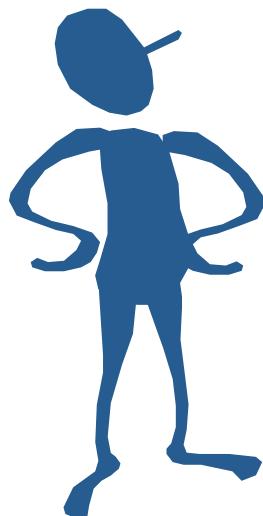
Definizione. Sia $G = (V, E)$ un grafo non orientato e $w:E \rightarrow \mathbb{R}$ una funzione peso:

- $T \subseteq E$ è un *albero di copertura* di G se T è connesso aciclico (albero libero) e

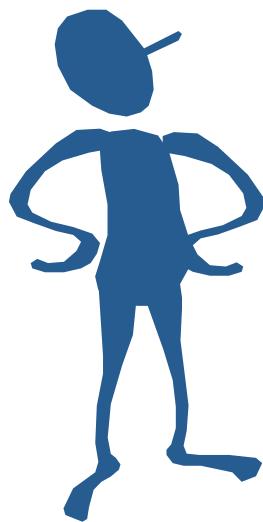
$$V[T] = \{u \mid \exists v. (u, v) \in T\} = V$$

- T è un albero di copertura minima (MST) se $w(T)$ è minimo dove

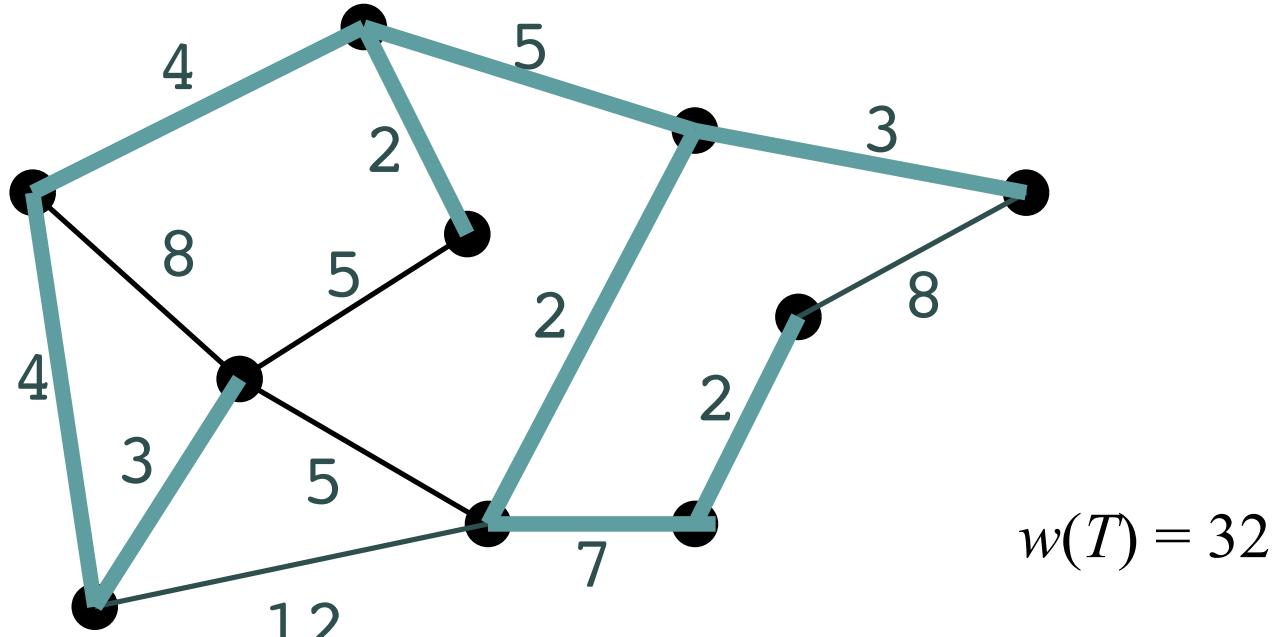
$$w(T) = \sum_{(u,v) \in T} w(u, v)$$



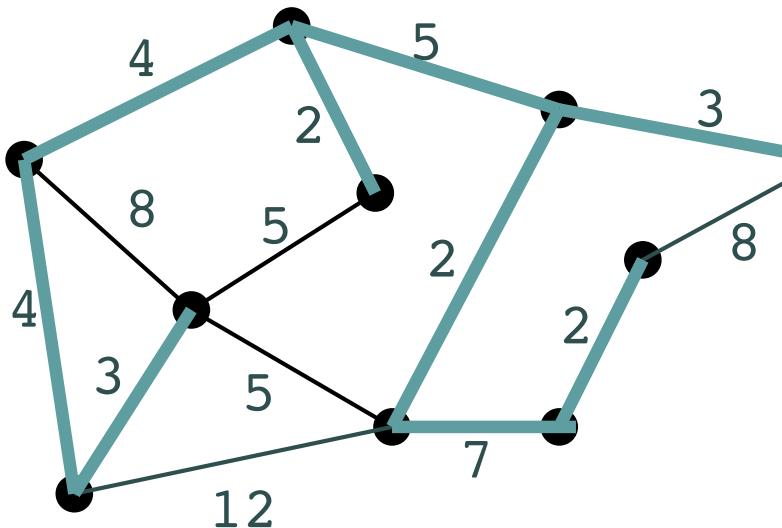
Albero di copertura minima



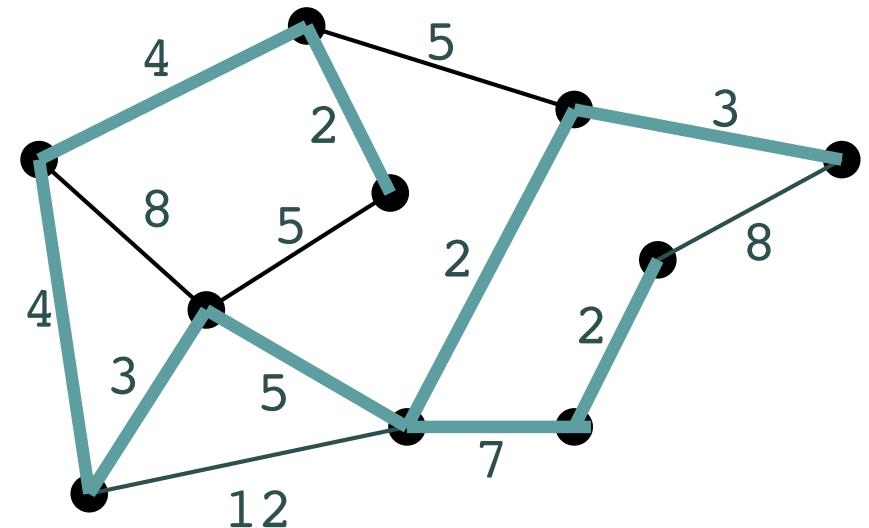
L'albero di copertura
minima è unico?



Albero di copertura minima

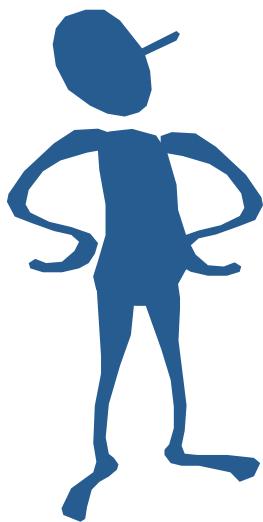


T



T'

$$w(T) = 32 = w(T')$$



Un algoritmo generico

Definizione. Sia $A \subseteq E$: (u, v) è sicuro per A se

$$\exists T \text{ MST di } G. A \cup \{(u, v)\} \subseteq T$$

GENERIC-MST(G, w)

▷ pre: $G = (V, E)$ non orientato e连通的

▷ post: $A \subseteq E$ è un MST di G

$A \leftarrow \emptyset$

while $V[A] \neq V$ **do** ▷ inv. $\exists T \text{ MST di } G. A \subseteq T$

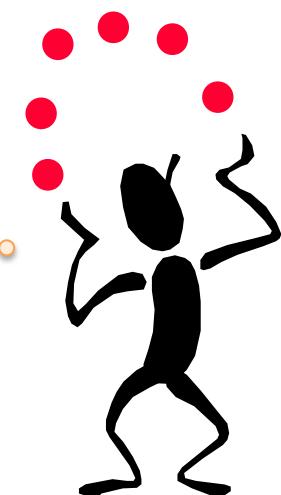
$(u, v) \leftarrow$ un arco sicuro per A in $E \setminus A$

$A \leftarrow A \cup \{(u, v)\}$

end while

return A

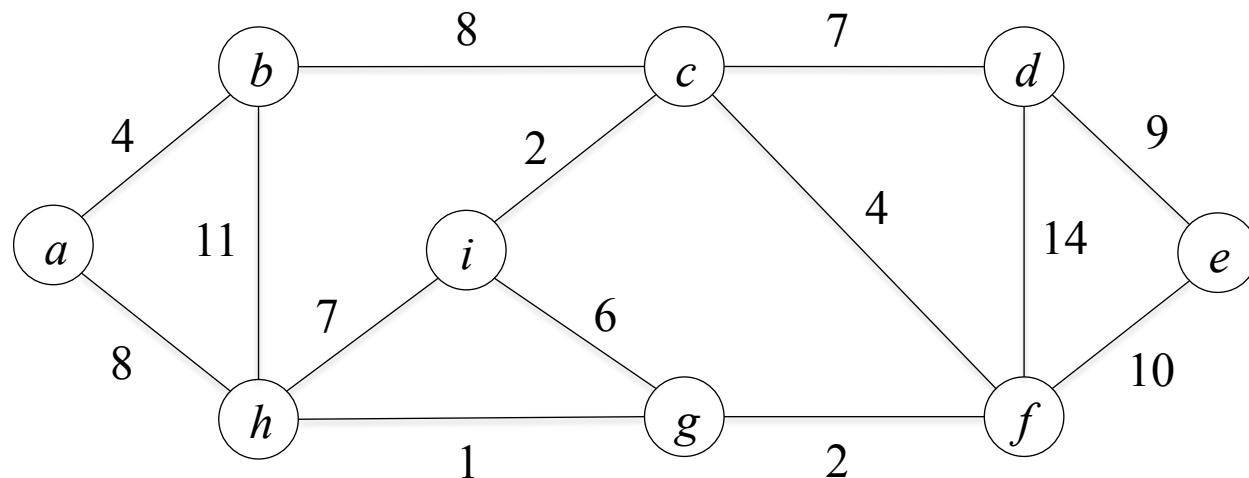
Se non so trovare un arco
“sicuro” allora questo è un
gioco si prestigio



Taglio ed arco “sicuro”

Definizione. Una coppia $(S, V \setminus S)$ è un *taglio* di $G = (V, E)$:

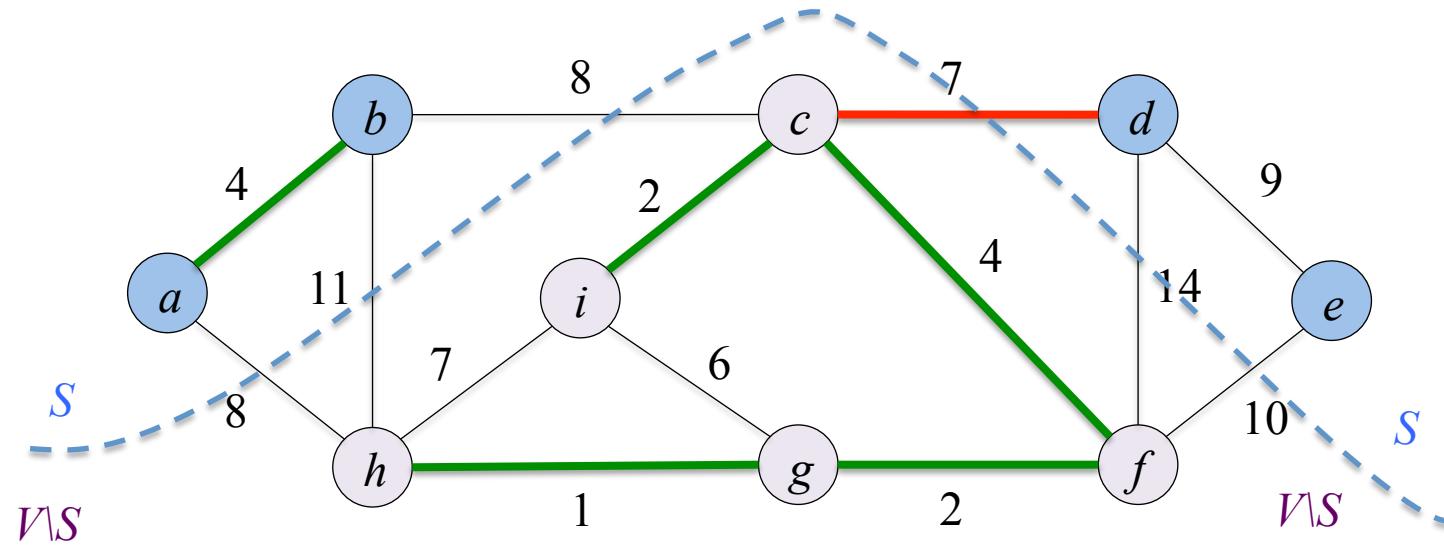
- $(u, v) \in E$ *attraversa* il taglio $(S, V \setminus S)$ se $u \in S$ e $v \in V \setminus S$
- $(S, V \setminus S)$ *rispetta* $A \subseteq E$ se nessun arco di A attraversa $(S, V \setminus S)$
- (u, v) è *leggero* se $w(u, v)$ è minimo tra gli archi che attraversano $(S, V \setminus S)$



Taglio ed arco “sicuro”

Definizione. Una coppia $(S, V \setminus S)$ è un *taglio* di $G = (V, E)$:

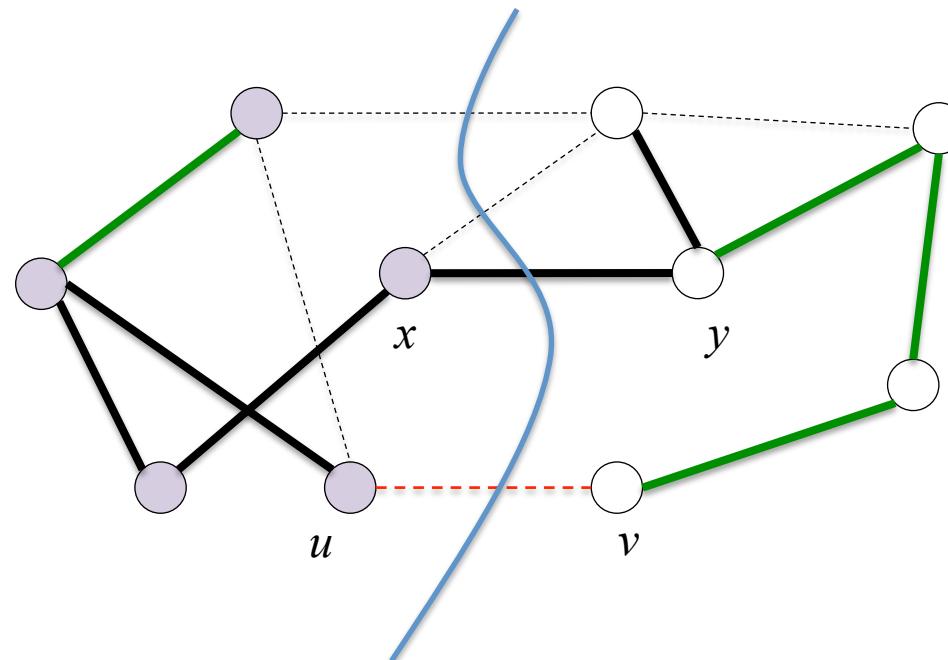
- $(u, v) \in E$ *attraversa* il taglio $(S, V \setminus S)$ se $u \in S$ e $v \in V \setminus S$
- $(S, V \setminus S)$ *rispetta* $A \subseteq E$ se nessun arco di A attraversa $(S, V \setminus S)$
- (u, v) è *leggero* se $w(u, v)$ è minimo tra gli archi che attraversano $(S, V \setminus S)$



Teorema del taglio

Teorema. In un grafo $G = (V, E)$ non orientato e connesso con pesi w , se $(S, V \setminus S)$ è un taglio che rispetta $A \subseteq E$ incluso in qualche MST di G allora

$$(u, v) \text{ è leggero} \Rightarrow (u, v) \text{ è sicuro per } A$$



$S :$

$V \setminus S :$

$A :$

$T :$

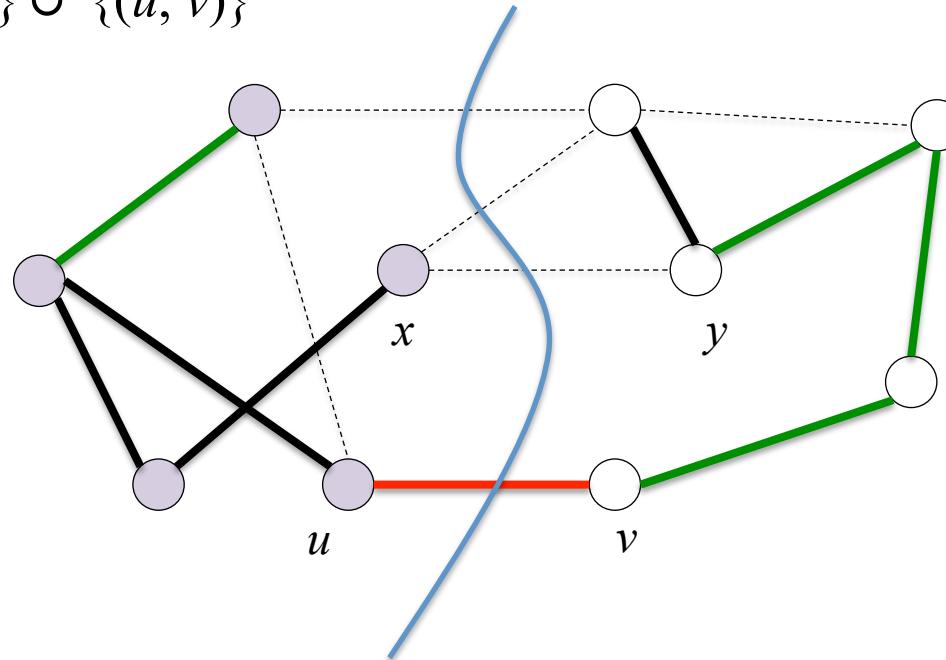
(u, v) leggero

Teorema del taglio

Teorema. In un grafo $G = (V, E)$ non orientato e connesso con pesi w , se $(S, V \setminus S)$ è un taglio che rispetta $A \subseteq E$ incluso in qualche MST di G allora

$$(u, v) \text{ è leggero} \Rightarrow (u, v) \text{ è sicuro per } A$$

$$T' = T \setminus \{(x, y)\} \cup \{(u, v)\}$$



$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T) \leq w(T')$$

Teorema del taglio

Corollario. In un grafo $G = (V, E)$ non orientato e connesso con pesi w , se $A \subseteq E$ incluso in qualche MST di G e $C = (V_C, E_C)$ una componente连通的 (albero) della foresta $G_A = (V, A)$:

(u, v) è leggero per $(V_C, V \setminus V_C)$ e collega C ad un'altra componente连通的 in G_A
 $\Rightarrow (u, v)$ è sicuro per A

Dim. Il taglio $(V_C, V \setminus V_C)$ rispetta A ed (u, v) è leggero per $(V_C, V \setminus V_C)$.

Il metodo “goloso” (greedy)



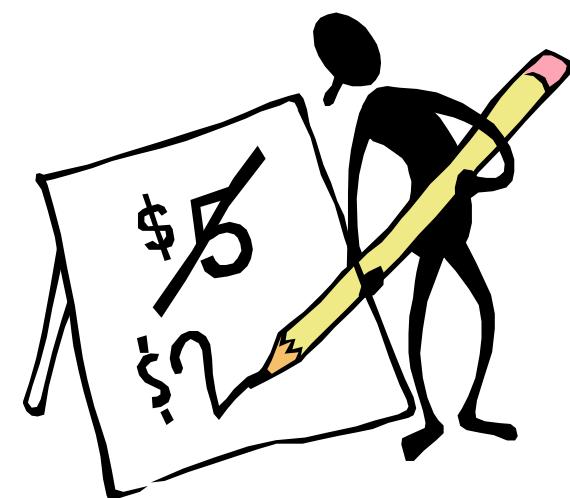
Il problema del resto



Vorrei 72 centesimi di resto
con il minor numero
possibile di monete

Metodo goloso:

- 1 moneta da 50
- 2 monete da 10
- 2 monete da 1



Il problema del resto

11c

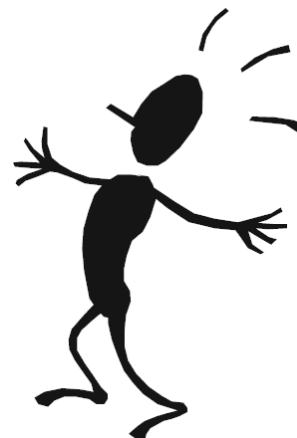


Vorrei 15 centesimi di resto
con il minor numero
possibile di monete

Metodo goloso:

1 moneta da 11

4 monete da 1



Il metodo “goloso” (greedy)

In un problema di ottimizzazione:

- *Scelta greedy*: un criterio di “appetibilità” con cui ogni soluzione ottima può essere modificata in un’altra anch’essa ottima
- *Sottostruttura ottima*: ogni struttura ottima deve contenere una sottostruttura ottima più piccola che si completa con una scelta greedy

L'algoritmo di Kruskal

$P = \{P_1, P_2, \dots, P_k\}$ è una *partizione* di $U = P_1 \cup P_2 \cup \dots \cup P_k$ se

$$i \neq j \Rightarrow P_i \cap P_j = \emptyset$$

MST-KUSKAL(G, w)

▷ pre: $G = (V, E)$ non orientato e connesso

▷ post: $A \subseteq E$ è un MST di G

$A \leftarrow \emptyset$

$P \leftarrow \{\{v_1\}, \dots, \{v_n\}\}$ una partizione di V

$L \leftarrow$ la lista degli archi in E in ordine non decrescente rispetto a w

while $L \neq \text{nil}$ **do**

$(u, w) \leftarrow \text{HEAD}(L)$, $L \leftarrow \text{TAIL}(L)$

$U, W \leftarrow$ gli unici insiemi in P t.c. $u \in U$ e $w \in W$

if $U \neq W$ **then**

$A \leftarrow A \cup \{(u, w)\}$

$P \leftarrow (P \setminus \{U, W\}) \cup \{U \cup W\}$

end if

end while

return A

Correttezza di MST-Kruskal

Se $G = (V, E)$ e $U \subseteq V$, $F \subseteq E$ allora

- $U[F] =$ insieme dei vertici $u, v \in U$ t.c. $(u, v) \in F$
- $F[U] =$ insieme degli archi $(u, v) \in F$ t.c. $u, v \in U$

Invariante del while di MST-Kruskal:

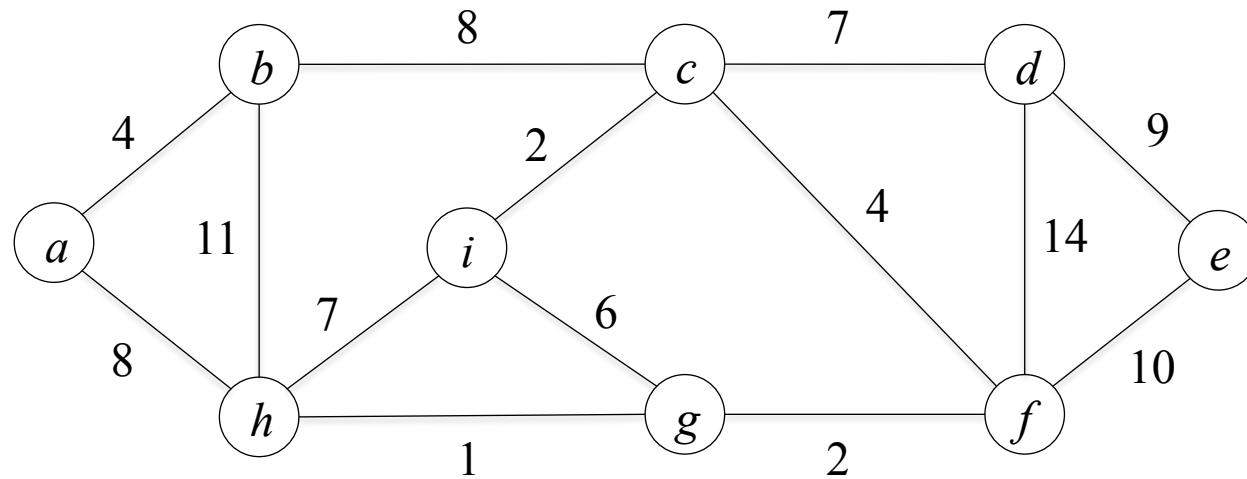
1. $\exists T$ MST di G . $A \subseteq T$ e
2. $(u, w) \in A \wedge u \in U \wedge w \in W \Rightarrow U = W$
3. $\forall U \in P$. $(U, A[U])$ è una parte连通 (albero) di $G_A = (V, A)$ (foresta)
4. $(u, v) \in L \Rightarrow (u, v) \notin A \wedge (u, v) \in T \setminus A \Rightarrow (u, v) \in L$

Supponiamo che l'invariante valga prima del while e che $L \neq \text{nil}$

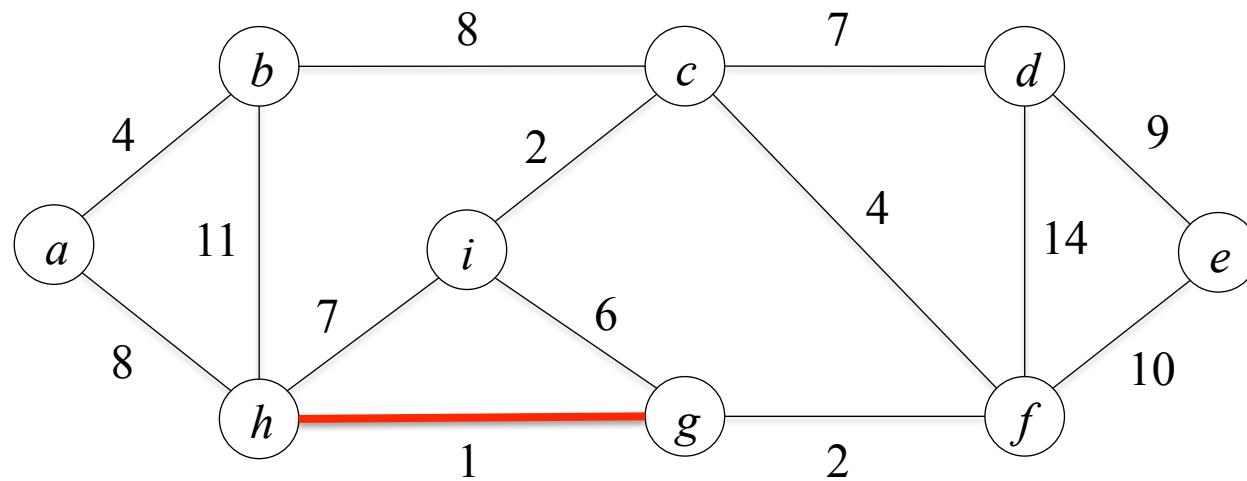
Sia $(u, w) = \text{Head}(L)$ con $u \in U$ e $w \in W$

- se $U = W$ allora $A[U] = A[W]$ è un albero di copertura di U e $(u, w) \notin A$ quindi $A \cup \{(u, w)\}$ contiene un ciclo da cui $A \cup \{(u, w)\} \not\subseteq T$ dunque $(u, w) \notin T$ (e a nessun altro MST che includa A).
- Se $U \neq W$ allora (u, w) attraversa $(U, V \setminus U)$ che rispetta A , ed è leggero per A , dunque è sicuro per il corollario al teorema del taglio.

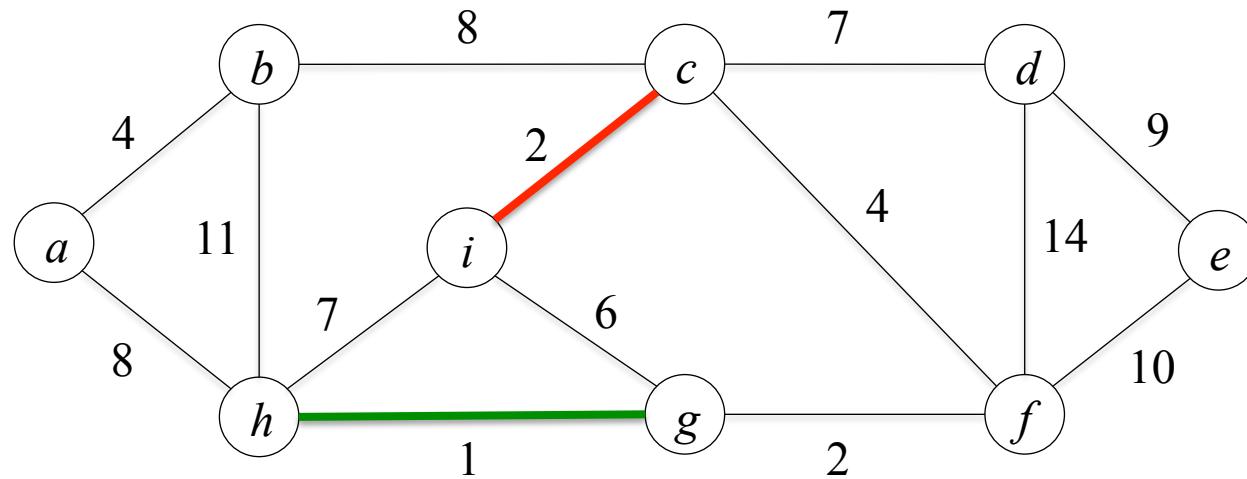
Esempio di MST-Kruskal



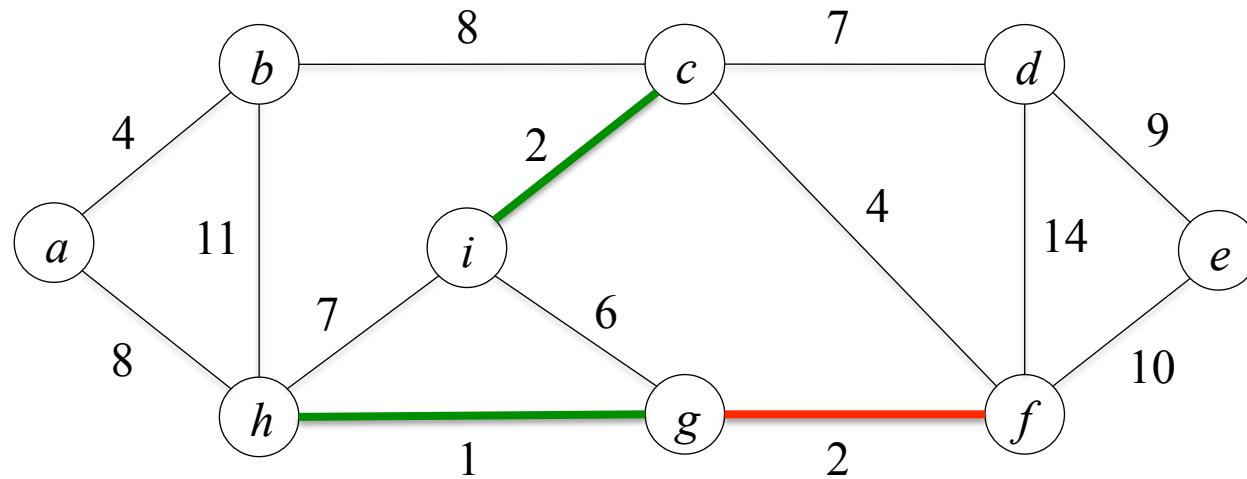
Esempio di MST-Kruskal



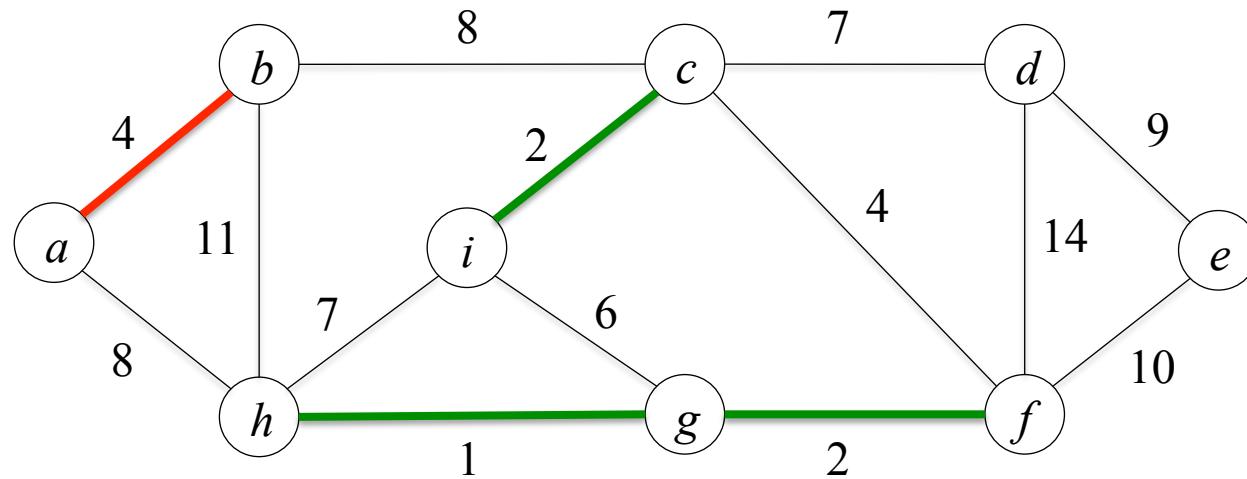
Esempio di MST-Kruskal



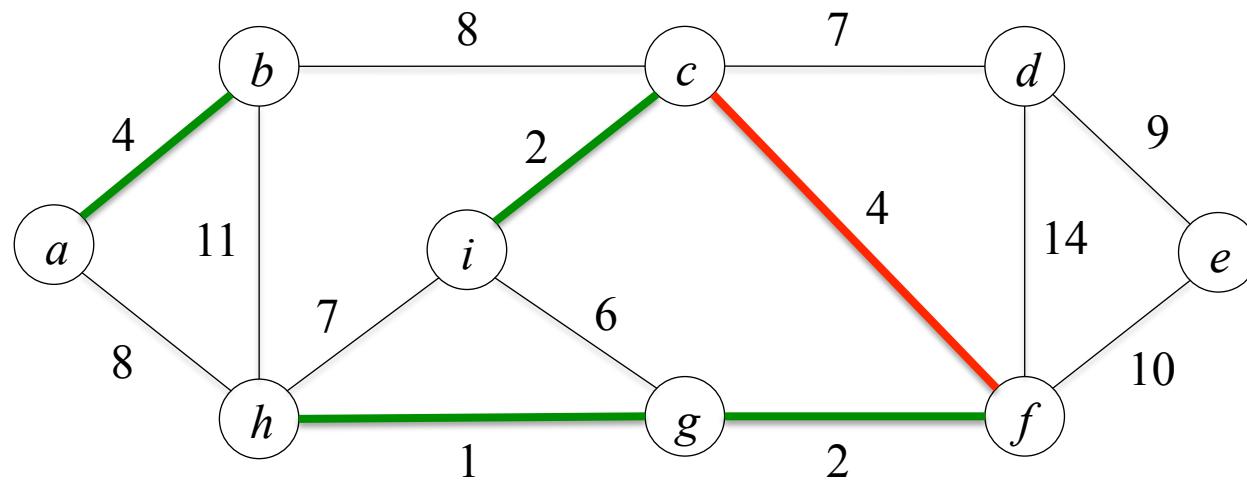
Esempio di MST-Kruskal



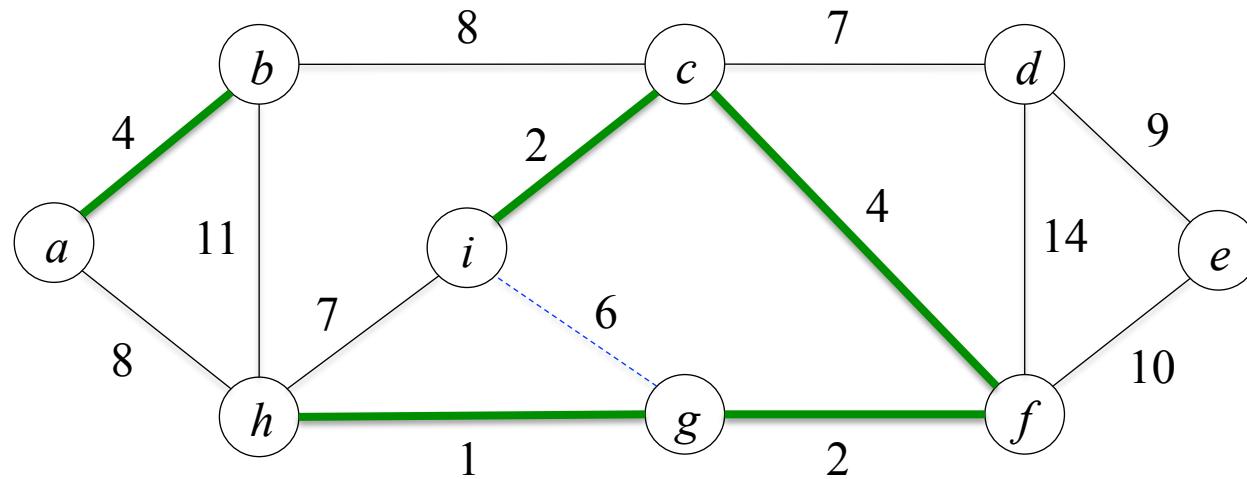
Esempio di MST-Kruskal



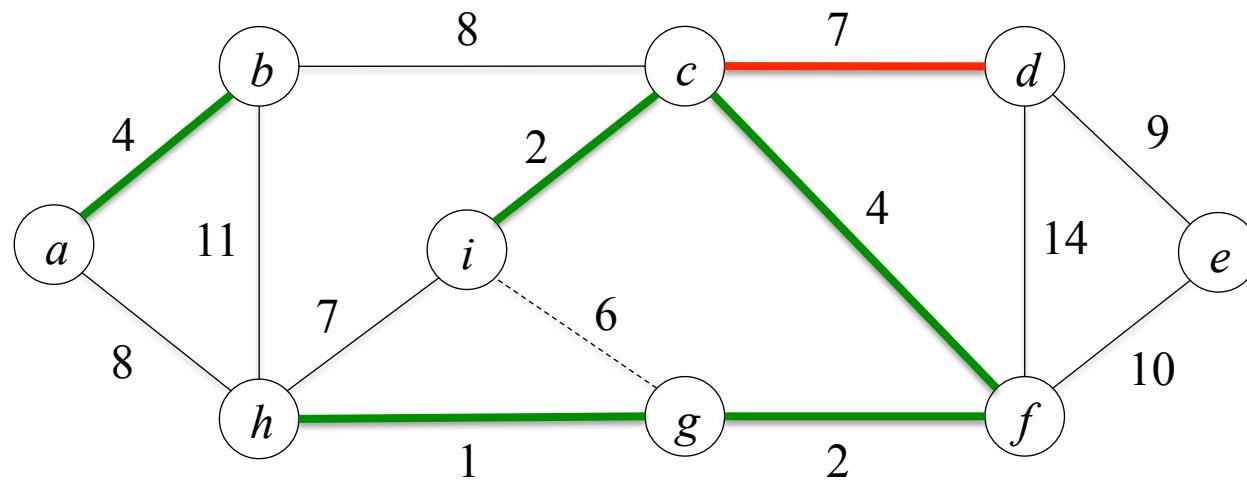
Esempio di MST-Kruskal



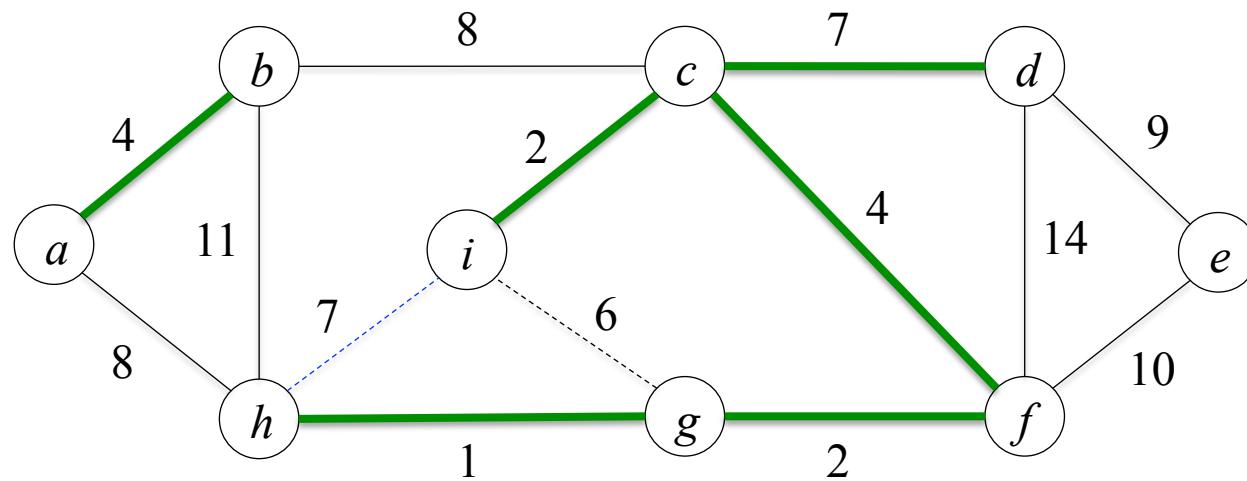
Esempio di MST-Kruskal



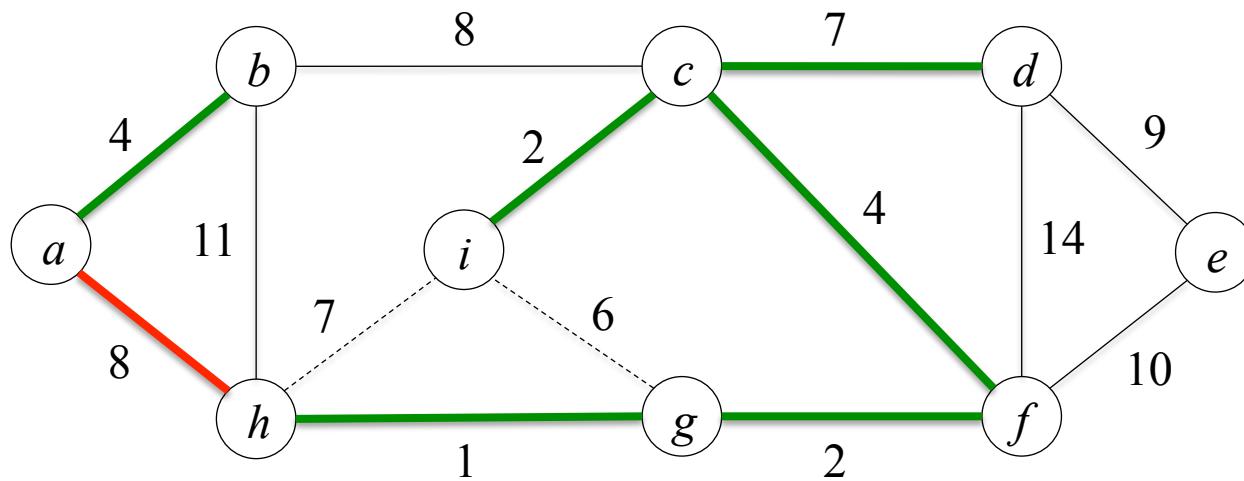
Esempio di MST-Kruskal



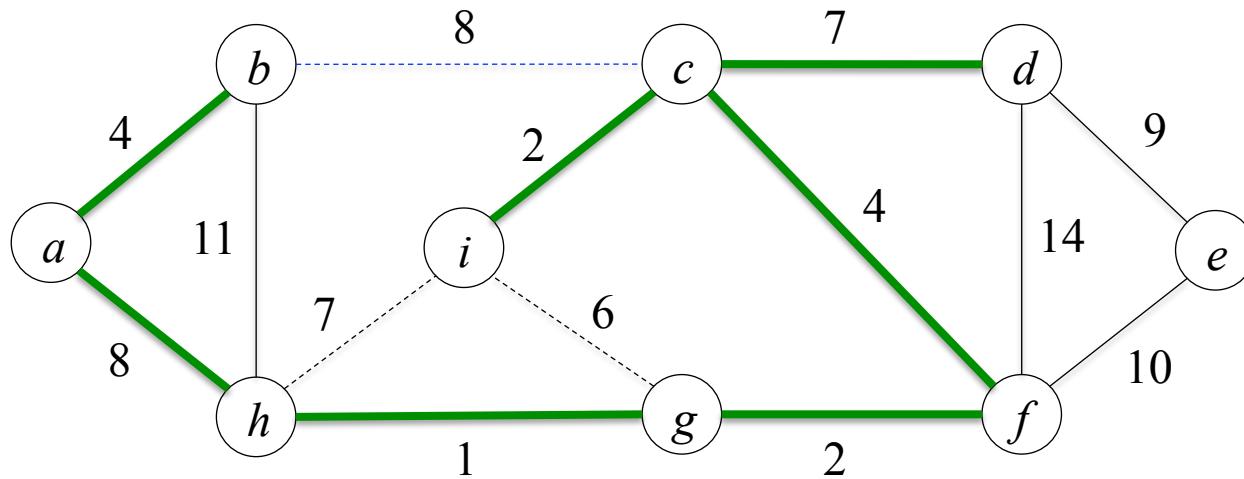
Esempio di MST-Kruskal



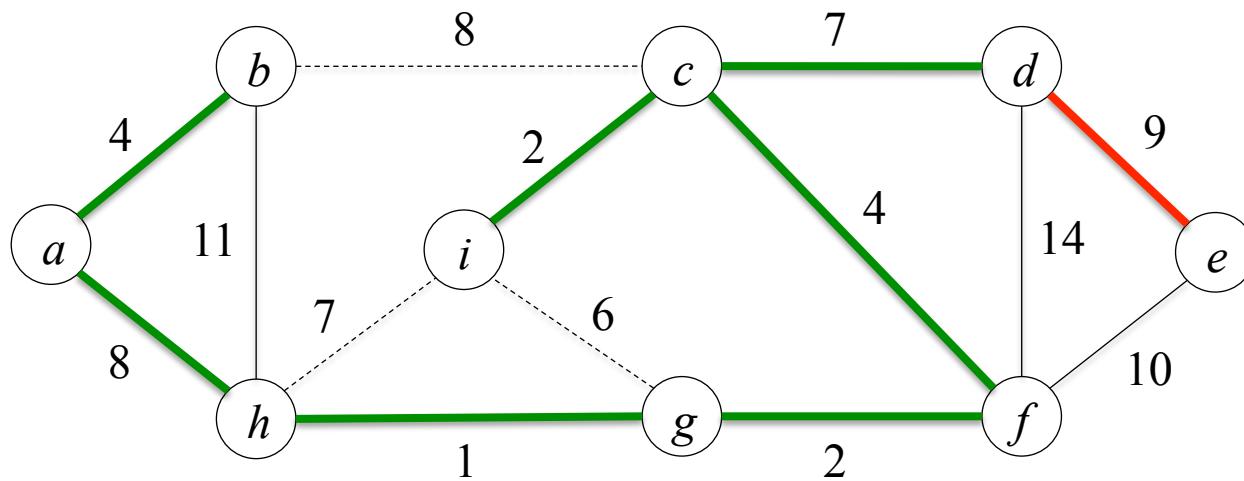
Esempio di MST-Kruskal



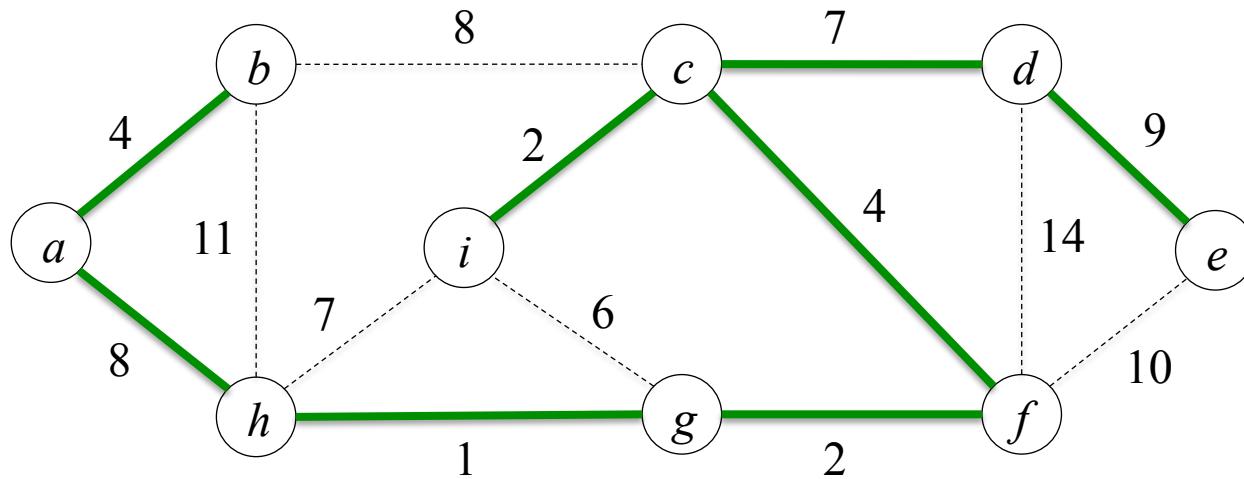
Esempio di MST-Kruskal



Esempio di MST-Kruskal



Esempio di MST-Kruskal



Union-Find Set

$\mathbf{P} = \{P_1, P_2, \dots, P_k\}$ è una *partizione* di $U = P_1 \cup P_2 \cup \dots \cup P_k$ se

$$i \neq j \Rightarrow P_i \cap P_j = \emptyset$$

Se $U = \{u_1, \dots, u_n\}$ allora $\{\{u_1\}, \dots, \{u_n\}\}$ è la *partizione discreta* di U

$r : \mathbf{P} \rightarrow U$ è una *funzione rappresentante* se $\forall P \in \mathbf{P}. r(P) \in P$

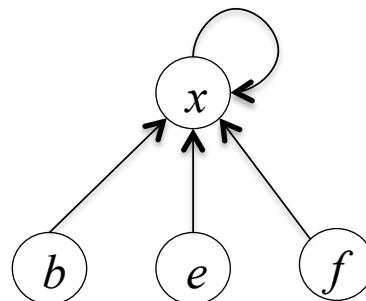
ADT Union-Find-Set

MakeSet(U) // pre: U (rappresenta un) insieme; post: la partizione discreta di U

Union(x, y, P) // pre: $\exists X, Y \in P. x \in X \wedge y \in Y$; post: $P \leftarrow P \setminus \{X, Y\} \cup \{X \cup Y\}$

Find(x, P) // pre: $\exists X \in P. x \in X$; post: ritorna $r(X)$

Union-Find Set

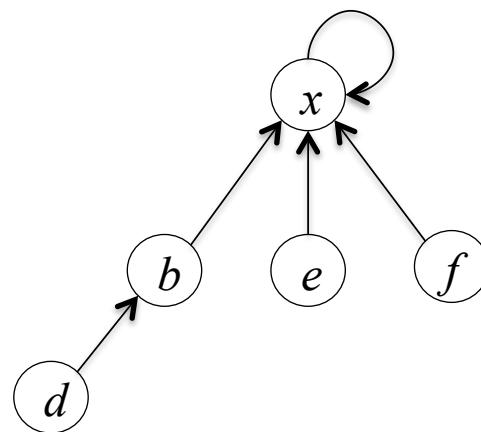


rappresenta $X = \{x, b, e, f\}$
con $r(X) = x$

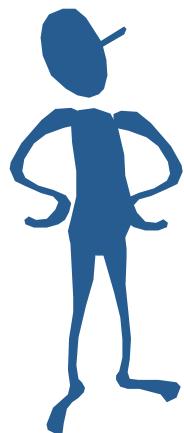
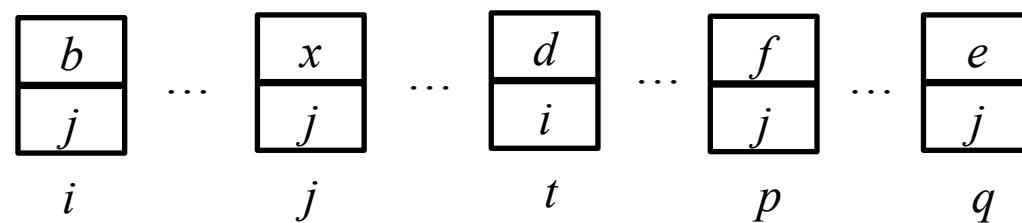


Rappresentiamo una partizione con una foresta di alberi pozzo, ciscuno dei quali è una delle parti con in radice il suo rappresentante

Union-Find Set

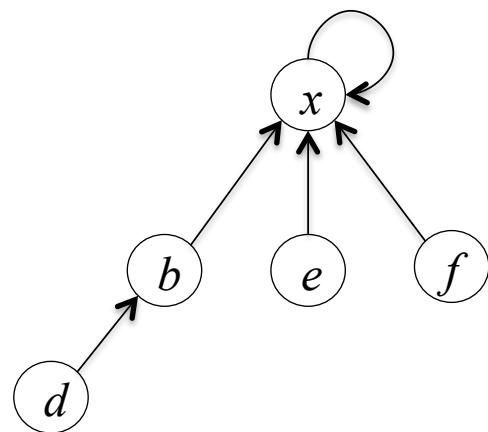


rappresenta $X = \{x, b, e, d, f\}$
con $r(X) = x$

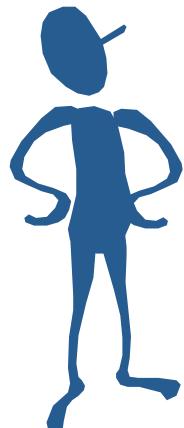


= MakeSet([x, b, e, d, f, \dots])

Union-Find Set



rappresenta $X = \{x, b, e, d, f\}$
con $r(X) = x$

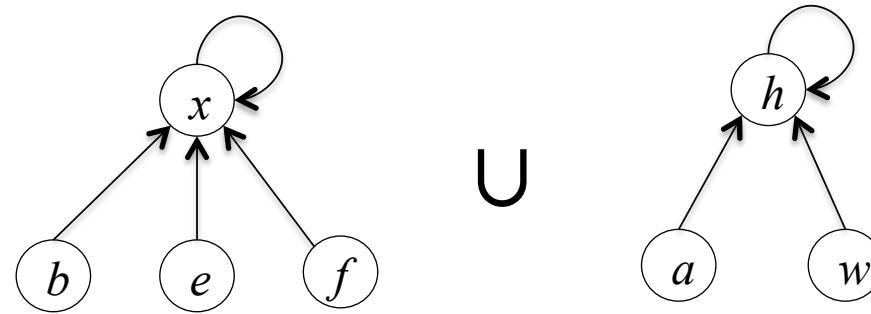


$\text{Find}(d, P) = x$ calcolabile navigando da d alla radice

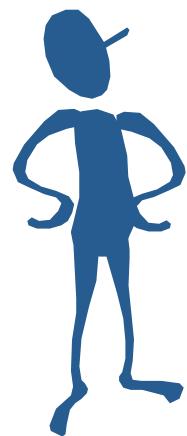
Se con $u \in U$ e $w \in W$
allora

$$U = W \Leftrightarrow \text{Find}(u, P) = \text{Find}(v, P)$$

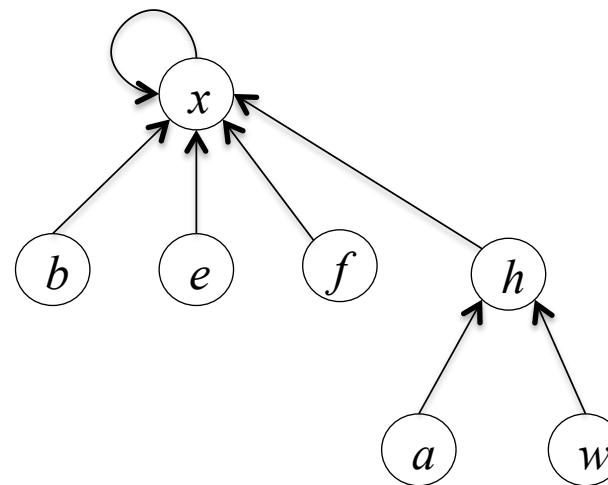
Union-Find Set



U



=



= Union(*f*, *w*, P)

Union-Find Set

Con opportune euristiche* i tempi delle operazioni di una struttura Union-Find realizzata con il vettore dei padri sono:

$$\begin{aligned}\text{Time(MakeSet)} &= O(n) && \text{dove } n = \text{cardinalità dell'insieme dato} \\ \text{Time(Find)} &= \text{Time(Union)} \\ &= O(m \log n) && \text{tempo ammortizzato dopo } m \text{ operazioni}\end{aligned}$$

Time(Find) = Time(Union)
perché Union fonde due alberi
dopo averne trovato le radici



* Vedi Cormen par. 21.3

Complessità MST-Kruskal

MST-KUSKAL(G, w)

▷ pre: $G = (V, E)$ non orientato e连通的

▷ post: $A \subseteq E$ è un MST di G

$A \leftarrow \text{nil}$

$P \leftarrow \text{MAKE-SET}(V)$

$L \leftarrow$ la lista degli archi in E in ordine non decrescente rispetto a w

while $L \neq \text{nil}$ **do**

$(u, w) \leftarrow \text{HEAD}(L)$, $L \leftarrow \text{TAIL}(L)$

if $\text{FIND}(u, P) \neq \text{FIND}(w, P)$ **then**

$A \leftarrow \text{CONS}(\{(u, w)\}, A)$

$\text{UNION}(u, w, P)$

end if

end while

return A

Complessità MST-Kruskal

MST-KUSKAL(G, w)

- ▷ pre: $G = (V, E)$ non orientato e connesso
- ▷ post: $A \subseteq E$ è un MST di G

$A \leftarrow \text{nil}$

$P \leftarrow \text{MAKE-SET}(V)$

$L \leftarrow$ la lista degli archi in E in ordine non decrescente rispetto a w

while $L \neq \text{nil}$ **do**

$(u, w) \leftarrow \text{HEAD}(L)$, $L \leftarrow \text{TAIL}(L)$

if $\text{FIND}(u, P) \neq \text{FIND}(w, P)$ **then**

$A \leftarrow \text{CONS}(\{(u, w)\}, A)$

$\text{UNION}(u, w, P)$

end if

end while

return A

$n = |V|$, $m = |E|$ allora Time(MST-Kruskal) =
Time(Make-Set) = $O(n)$ +
Tempo per ordinare L = $O(m \log m)$ +
Time(Find) + Time(Union) dopo m op. = $O(m \log n)$



G è connesso implica $|V| - 1 \leq |E| \leq |V|^2$
da cui $\log m \leq \log n^2 = O(\log n)$, quindi

Time(MST-Kruskal) = $O(m \log n)$

MST-Prim

MST-PRIM(G, w, r) $\triangleright G = (V, E), n = |V|, m = |E|$

for all $u \in V$ **do**

$v.key \leftarrow \infty, v.\pi \leftarrow nil$

end for

$A \leftarrow \emptyset, r.key \leftarrow 0$

$Q \leftarrow \text{MAKEPRIORITYQUEUE}(V)$

while $Q \neq \emptyset$ **do**

$u \leftarrow \text{EXTRACT-MIN}(Q)$

$A \leftarrow A \cup \{(u.\pi, u)\}$

for all $v \in Adj[u]$ **and** $v \in Q$ **do**

if $v.key > w(u, v)$ **then**

$v.key \leftarrow w(u, v)$

$v.\pi \leftarrow u$

 DECREASE-KEY($v, w(u, v), Q$)

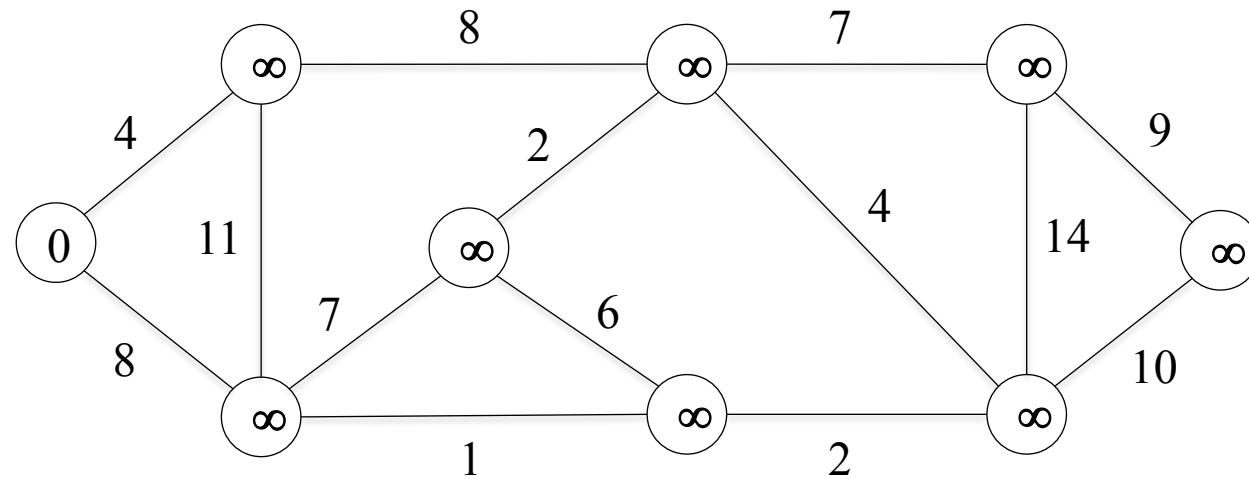
end if

end for

end while

return A

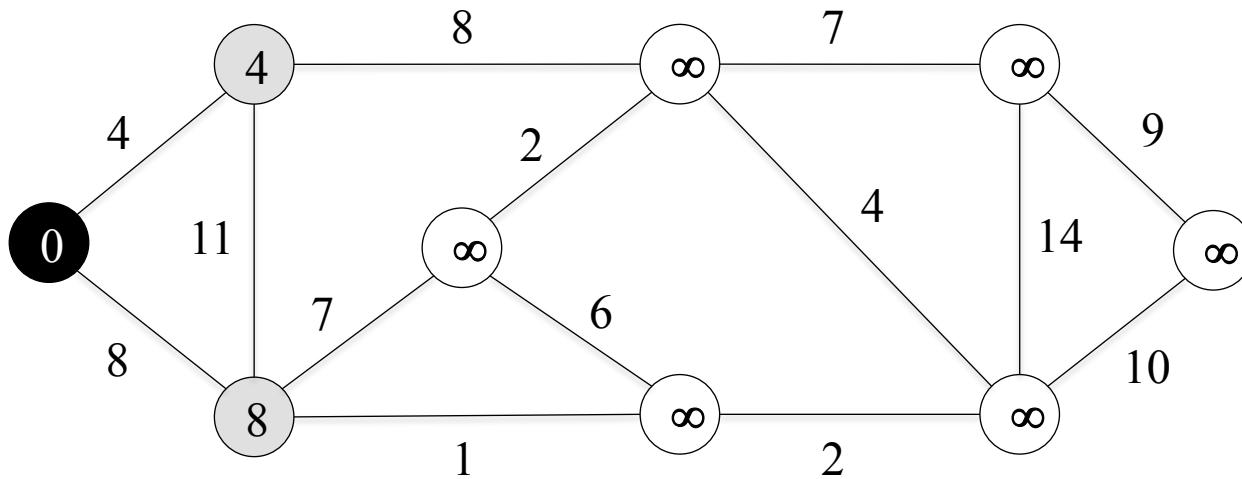
Esempio di MST-Prim



I vertici in Q sono bianchi e contengono il valore di key



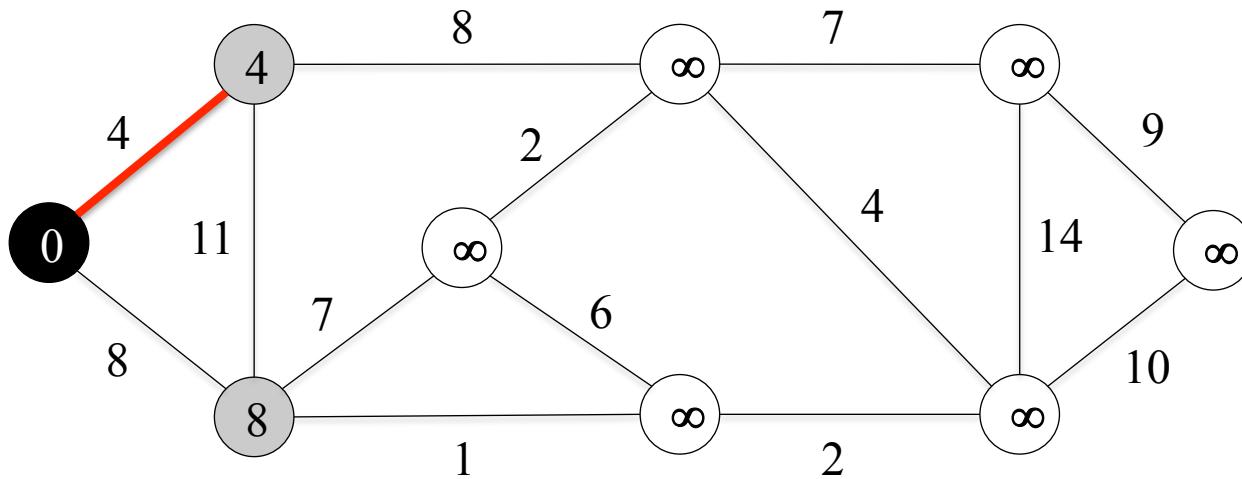
Esempio di MST-Prim



La frontiera consiste dei vertici v in Q con $v.key \neq \infty$ (grigi)



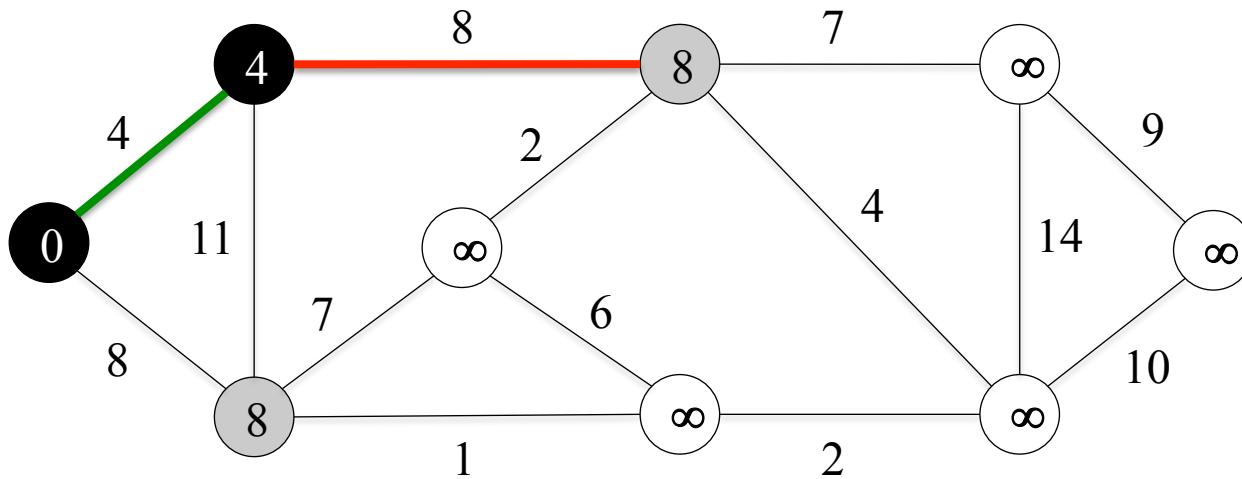
Esempio di MST-Prim



L'arco rosso è leggero per A
rispetto al taglio $(Q, V \setminus Q)$



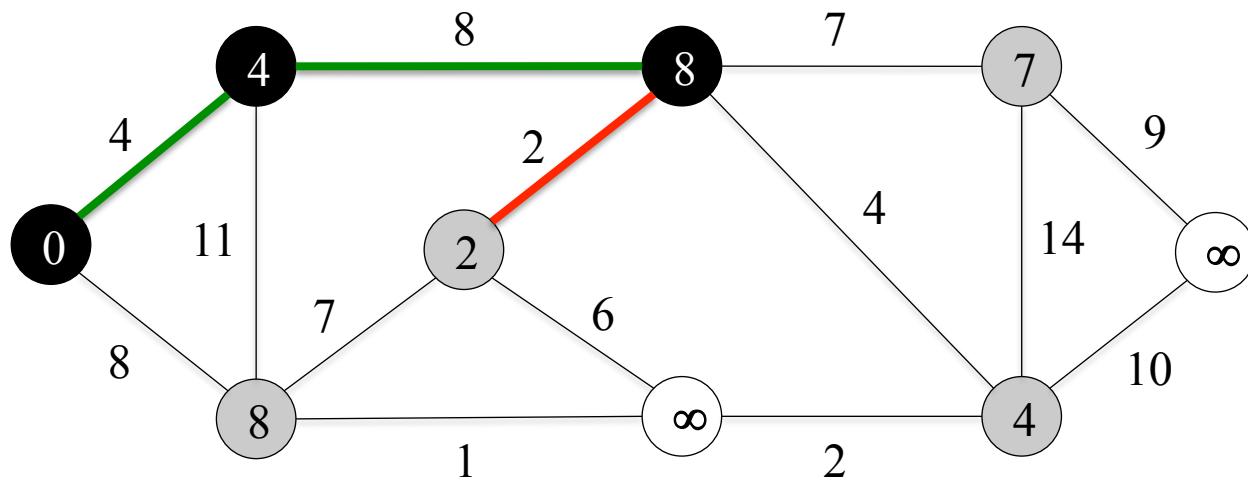
Esempio di MST-Prim



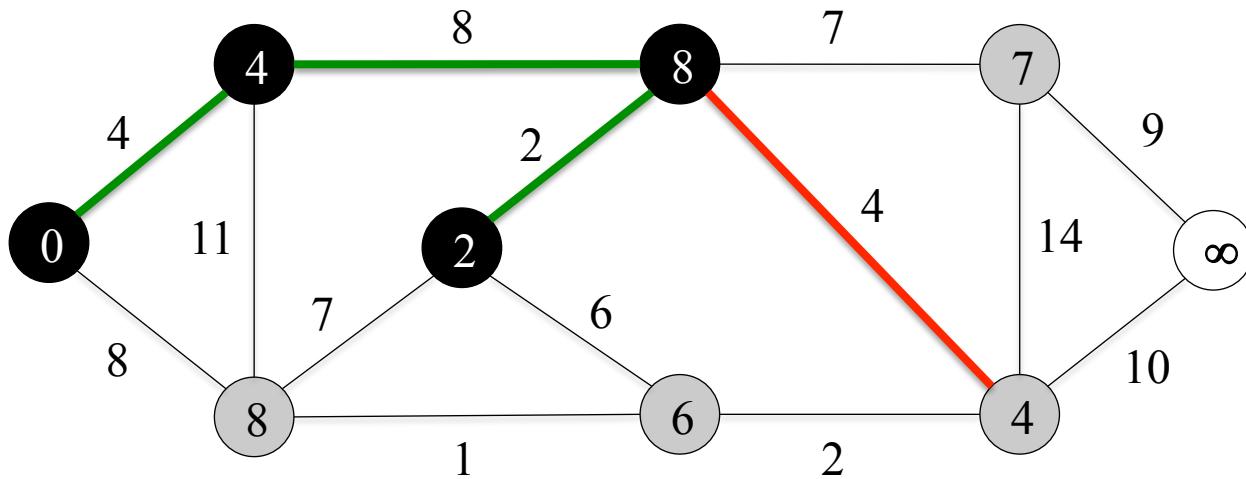
Gli archi verdi sono in A



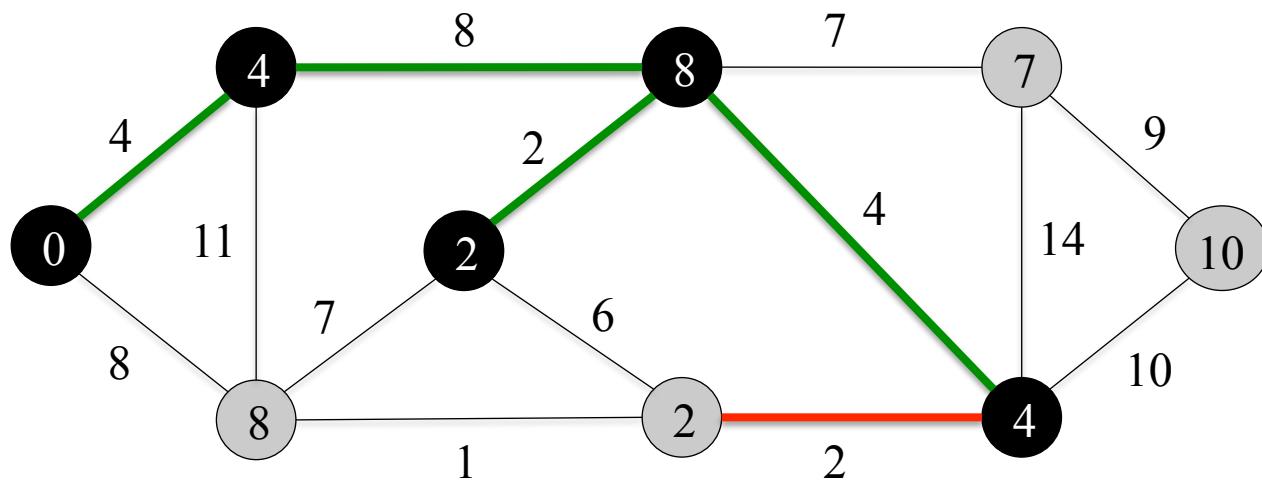
Esempio di MST-Prim



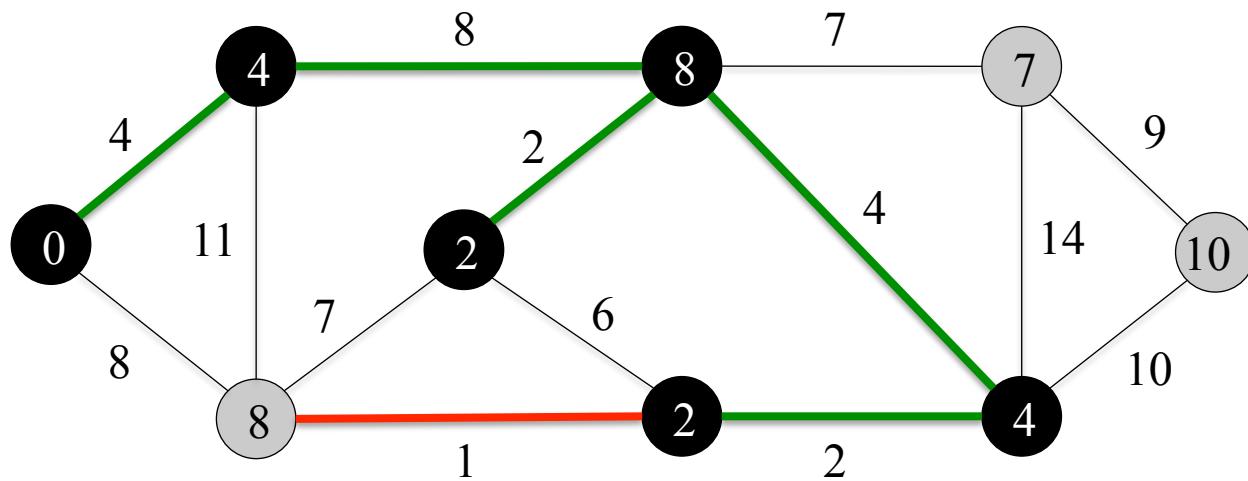
Esempio di MST-Prim



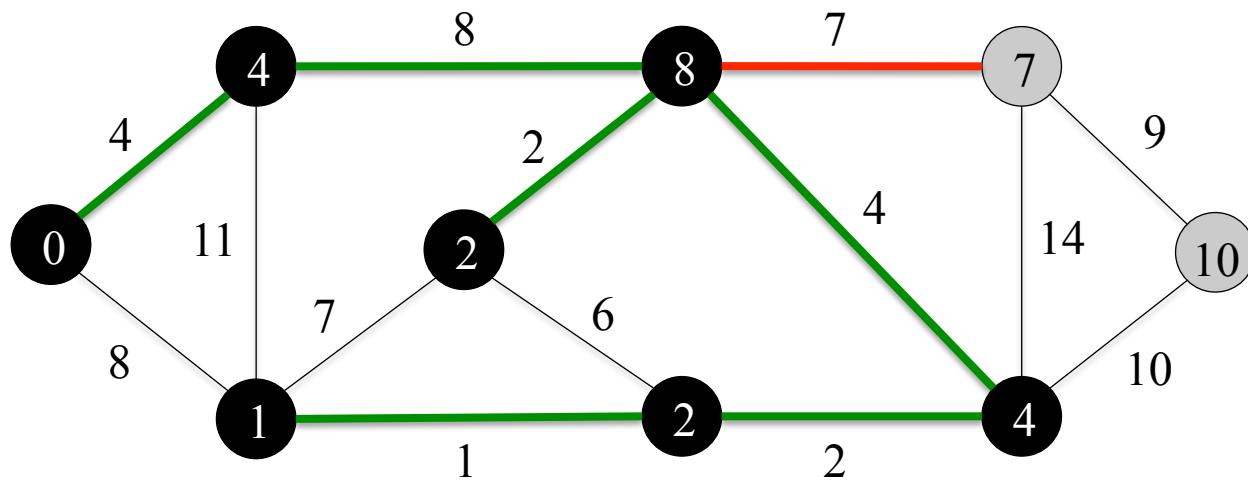
Esempio di MST-Prim



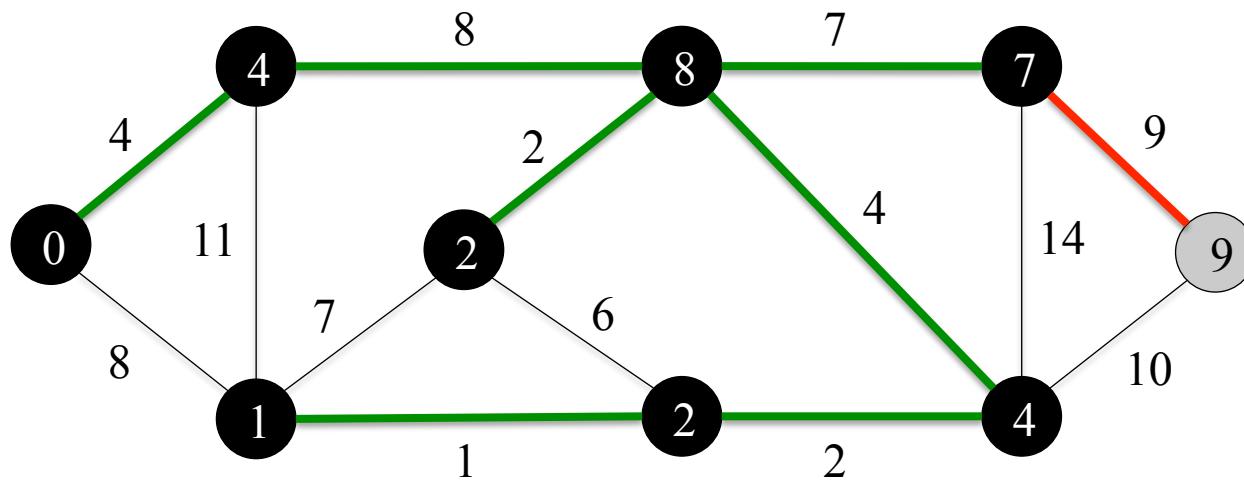
Esempio di MST-Prim



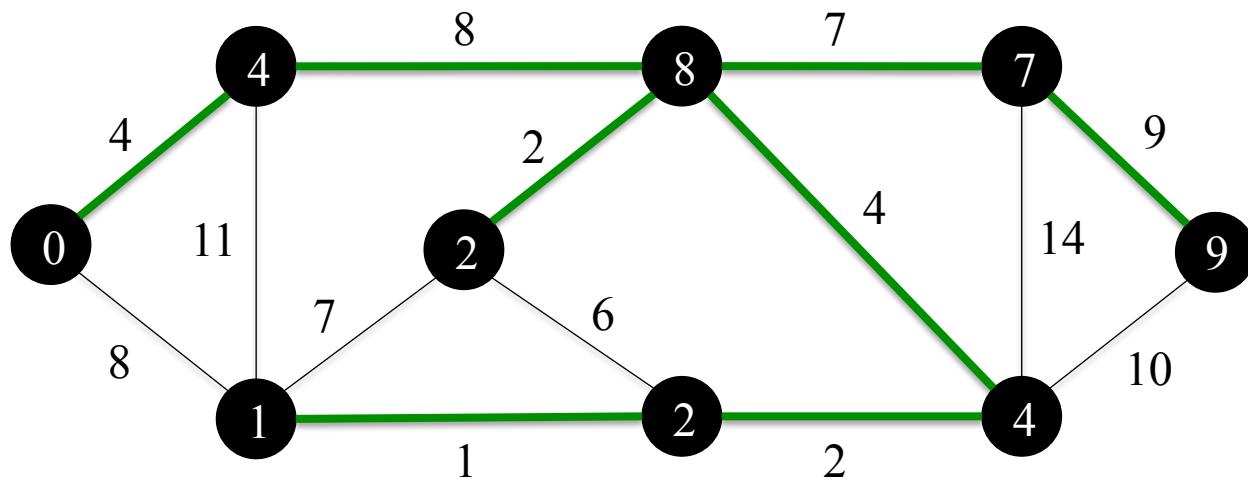
Esempio di MST-Prim



Esempio di MST-Prim



Esempio di MST-Prim



Correttezza di MST-Prim

Invariante del while di MST-Prim:

1. $\exists T \text{ MST di } G. A \subseteq T \text{ e}$
2. $V[A] = \text{vertici neri o grigi} = V \setminus \{v \in Q \mid v.key = \infty\}$
3. se $v \in Q$ e $v.key < \infty$ allora $v.key = w(u, v)$ per qualche $u \in V[A] \setminus Q$

Da 3. segue che l'arco di peso minimo (u, v) con $u \in V[A] \setminus Q$ e $v.key < \infty$ è **leggero** per A



Complessità di MST-Prim

$$n = |V|, m = |E|$$

Q implementata con uno heap minimo

Time(MakePriorityQueue) = $O(n \log n)$ usando Build-Heap

Time(Extract-Min) = $O(\log n)$

Time(Decrease-Key) = $O(\log n)$ facendo risalire la chive nello heap

Complessità di MST-Prim

MST-PRIM(G, w, r) $\triangleright G = (V, E), n = |V|, m = |E|$

for all $u \in V$ do

$v.key \leftarrow \infty, v.\pi \leftarrow \text{nil}$

$O(n)$

end for

$A \leftarrow \emptyset, r.key \leftarrow 0$

$Q \leftarrow \text{MAKEPRIORITYQUEUE}(V)$

$O(n \log n)$

while $Q \neq \emptyset$ do

$u \leftarrow \text{EXTRACT-MIN}(Q)$

$O(\log n)$

$A \leftarrow A \cup \{(u.\pi, u)\}$

for all $v \in \text{Adj}[u]$ and $v \in Q$ do

if $v.key > w(u, v)$ then

$v.key \leftarrow w(u, v)$

$v.\pi \leftarrow u$

$\text{DECREASE-KEY}(v, w(u, v), Q)$

$O(\log n)$

end if

end for

end while

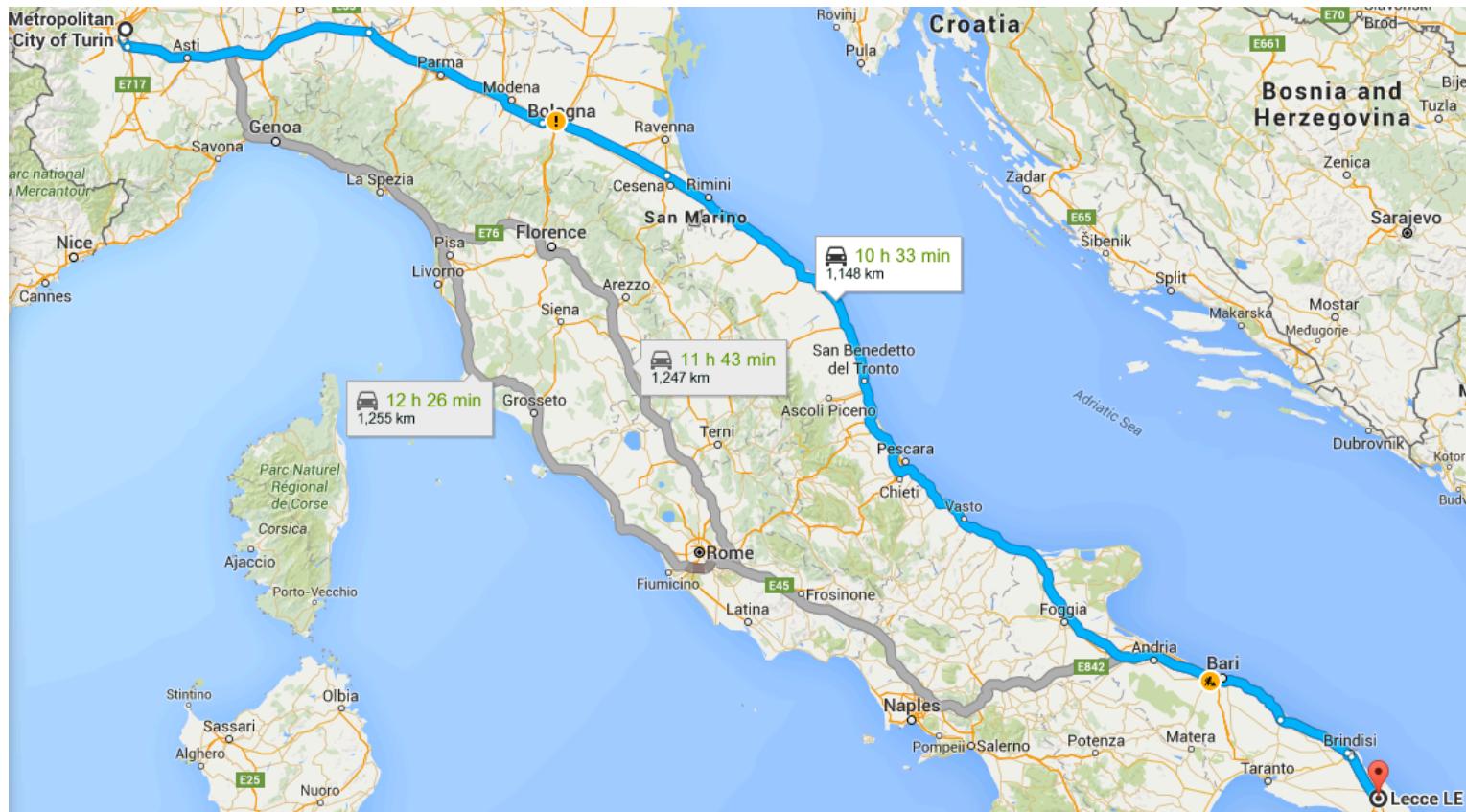
return A

$O(m \log n)$

Poiché G è connesso
 $n = O(m)$ onde il
tempo di Prim è
 $O(m \log n)$

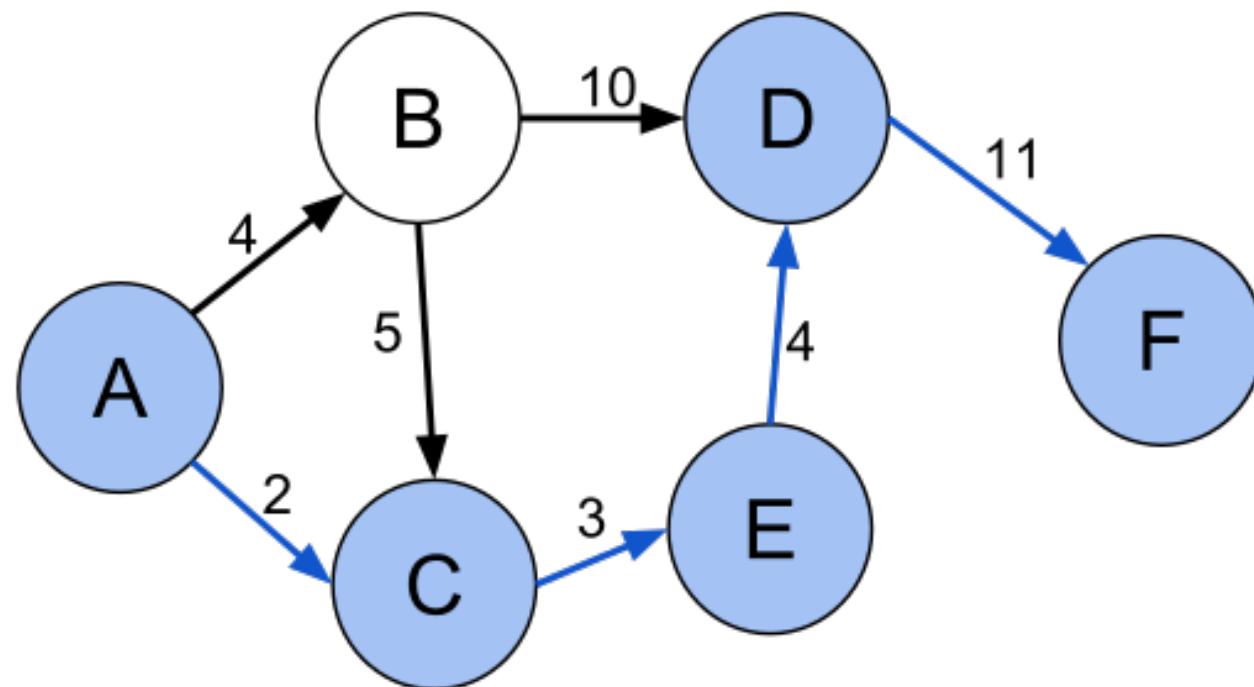


Cammini minimi Algoritmo di Dijkstra



Algoritmi e strutture dati
Lezione 18, a.a. 2016-17

Problema del cammino minimo



Definizioni

$$p = \langle v_0, v_1, \dots, v_k \rangle \quad w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

$$\delta(u, v) = \begin{cases} \min\{w(p) \mid u \xrightarrow{p} v\} & \text{se esiste un cammino da } u \text{ a } v \\ \infty & \text{altrimenti} \end{cases}$$

Ben definito solo se in G non vi sono cicli di peso < 0



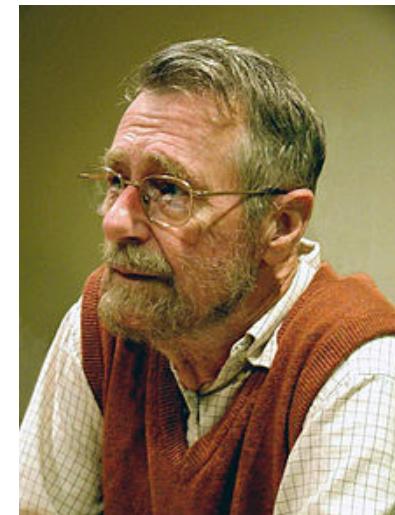
In questa lezione assumeremo
che w sia non negativa

Problema del cammino minimo

- *Input:* un grafo orientato $G = (V, E)$ con pesi w ed una sorgente $s \in V$
- *Output:* un sottografo $G_\pi = (V_\pi, E_\pi)$ t.c.
 - G_π è un albero radicato in s
 - $V_\pi \subseteq V$ è l'insieme dei vertici raggiungibili da s
 - per ogni $v \in V_\pi$ il cammino $s \rightarrow v$ in G_π ha peso minimo tra i cammini $s \rightarrow v$ in G .

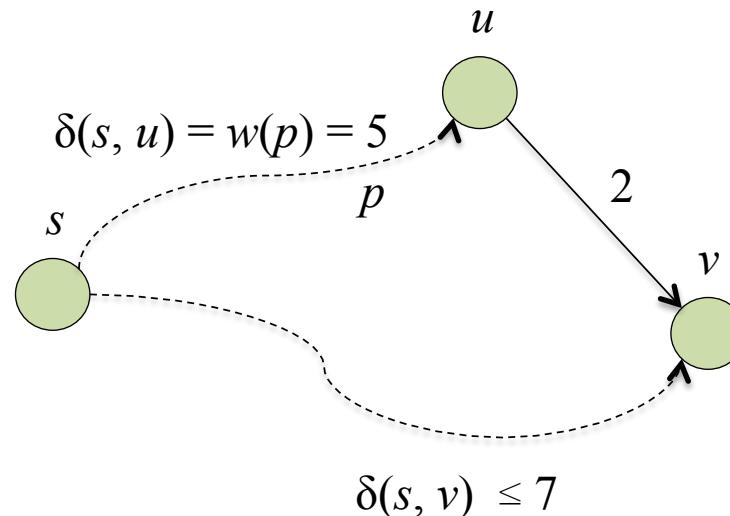
L'algoritmo di Dijkstra

- L'algoritmo di Dijkstra (1959), richiede che w non sia negativa
- Il caso generale si può trattare con l'algoritmo di Bellman-Ford (1957) di complessità $O(|V| \cdot |E|)$
- L'algoritmo di Pape-D'Esopo (1974) ha complessità superpolinomiale nel caso peggiore, ma è quasi lineare in $|V|$ se $|E| = O(|V|)$ ed il grafo è planare (come nelle vere mappe stradali)



E. W. Dijkstra (1930 - 2002)

Condizione di Bellman



$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Conseguenza della
diseguaglianza triangolare

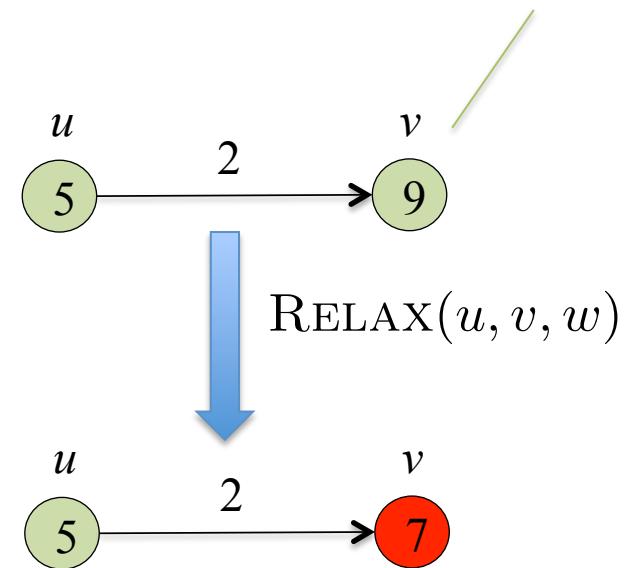


Rilassamento di un arco

$\text{RELAX}(u, v, w)$

```
if  $v.d > u.d + w(u, v)$  then  
   $v.d \leftarrow u.d + w(u, v)$   
   $v.\pi \leftarrow u$   
end if
```

$$v.d = 9 > 5 + 2 = u.d + 2$$



Rilassamento di un arco

$\text{RELAX}(u, v, w)$

if $v.d > u.d + w(u, v)$ **then**

$v.d \leftarrow u.d + w(u, v)$

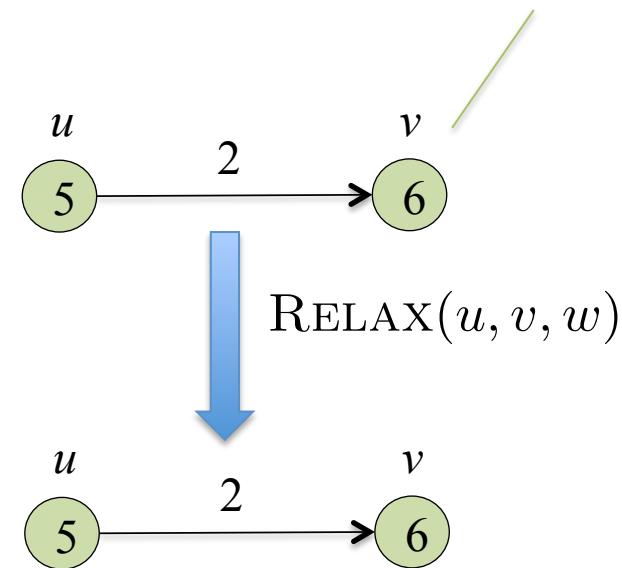
$v.\pi \leftarrow u$

end if

$$v.d = 6 \leq 5 + 2 = u.d + 2$$



Se inizialmente $v.d = \infty$
allora $v.d \geq \delta(s, v)$



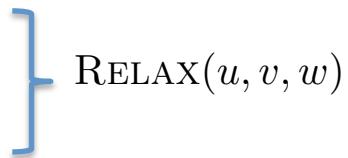
Lemma di convergenza

Lemma. Sia w non negativa e sia $s \rightsquigarrow u \rightarrow v$ un cammino minimo; se inizialmente $u.d = \infty$ e $u.d = \delta(s, u)$ prima della chiamata $\text{Relax}(u, v, w)$ Allora $v.d = \delta(s, v)$ dopo la chiamata.

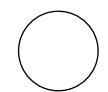
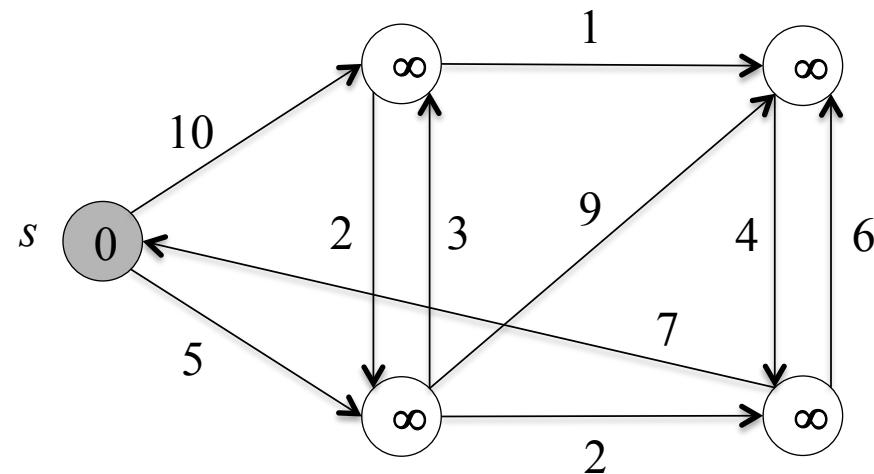
$$\begin{aligned} v.d &\leq u.d + w(u, v) && \text{per definizione di RELAX}(u, v, w) \\ &= \delta(s, u) + w(u, v) && \text{per ipotesi} \\ &= \delta(s, v) && s \rightsquigarrow u \rightarrow v \text{ è minimo} \end{aligned}$$

L'algoritmo di Dijkstra

```
DIJKSTRA( $G, w, s$ )       $\triangleright G = (V, E)$  orientato con pesi non negativi  $w$ 
for all  $v \in V$  do
     $v.d \leftarrow \infty$        $\triangleright$  stima in eccesso di  $\delta(s, v)$ 
     $v.\pi \leftarrow \text{nil}$ 
end for
 $s.d \leftarrow 0$ 
 $Q \leftarrow \text{MAKEPRIORITYQUEUE}(V)$        $\triangleright Q$  è uno heap minimo con  $v.key = v.d$ 
 $S \leftarrow \emptyset$ 
while  $Q \neq \emptyset$  do       $\triangleright$  inv.  $Q = V \setminus S$  e  $\forall v \in S. v.d = \delta(s, v)$ 
     $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
     $S \leftarrow S \cup \{u\}$ 
    for all  $v \in \text{Adj}[u]$  do
        if  $v.d > u.d + w(u, v)$  then
             $v.d \leftarrow u.d + w(u, v)$ 
             $v.\pi \leftarrow u$ 
             $\text{DECREASE-KEY}(v, w(u, v), Q)$ 
        end if
    end for
end while
return  $S$ 
```



Esecuzione dell'alg. di Dijkstra

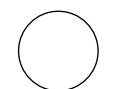
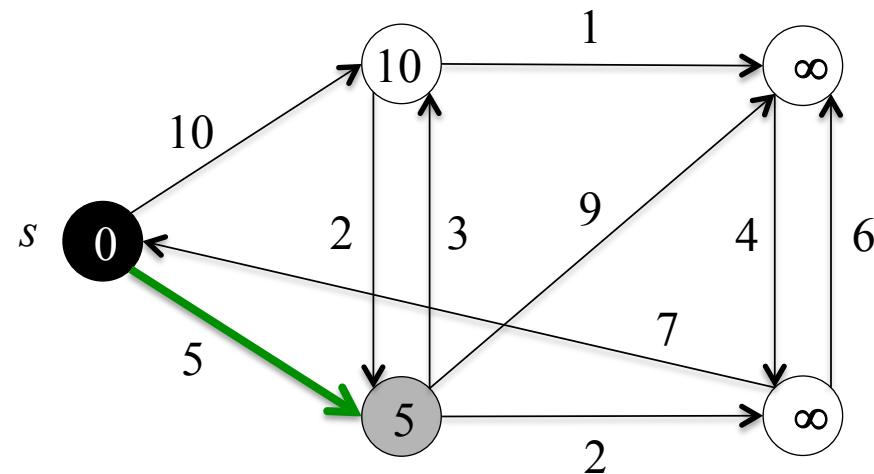


in $Q \setminus \min(Q)$



$\min(Q)$

Esecuzione dell'alg. di Dijkstra



in $Q \setminus \min(Q)$



$\min(Q)$

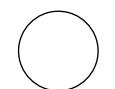
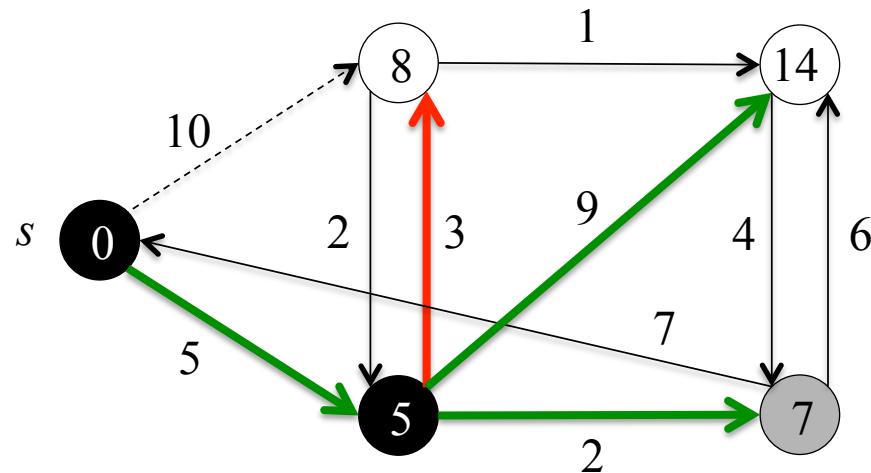


in S



in E_π

Esecuzione dell'alg. di Dijkstra



in $Q \setminus \min(Q)$



$\min(Q)$



in S

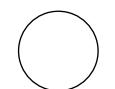
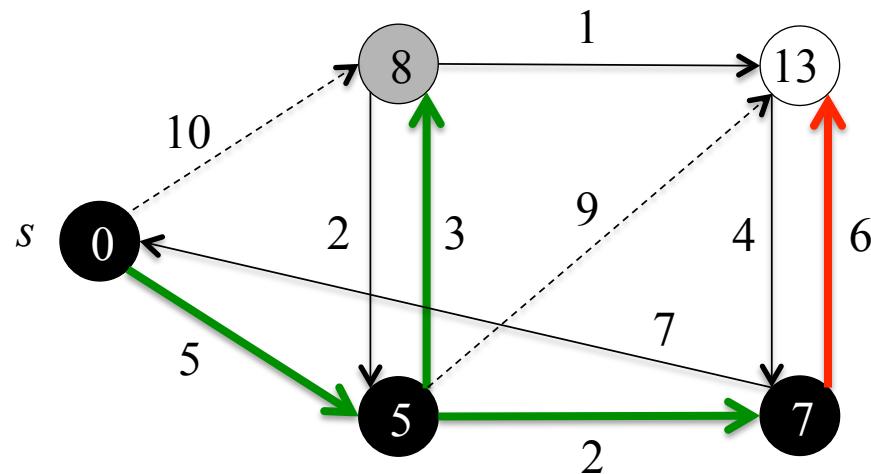


in E_π



in E_π per rilassamento

Esecuzione dell'alg. di Dijkstra



in $Q \setminus \min(Q)$



$\min(Q)$



in S

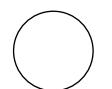
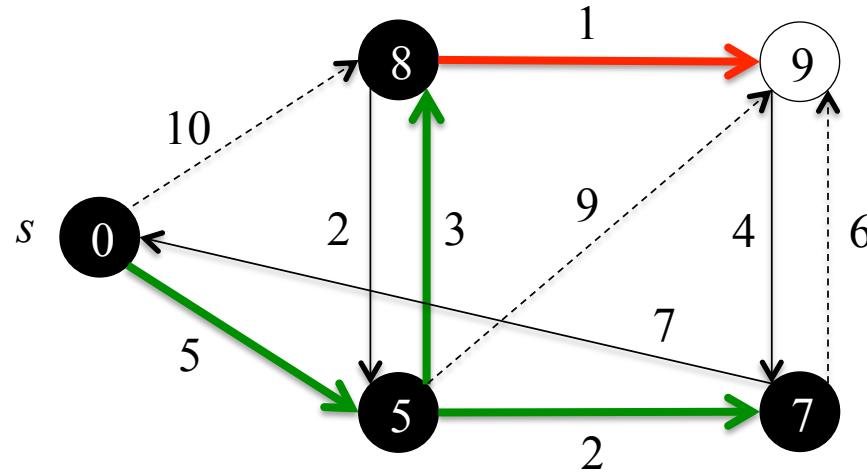


in E_π



in E_π per rilassamento

Esecuzione dell'alg. di Dijkstra



in $Q \setminus \min(Q)$



$\min(Q)$



in S

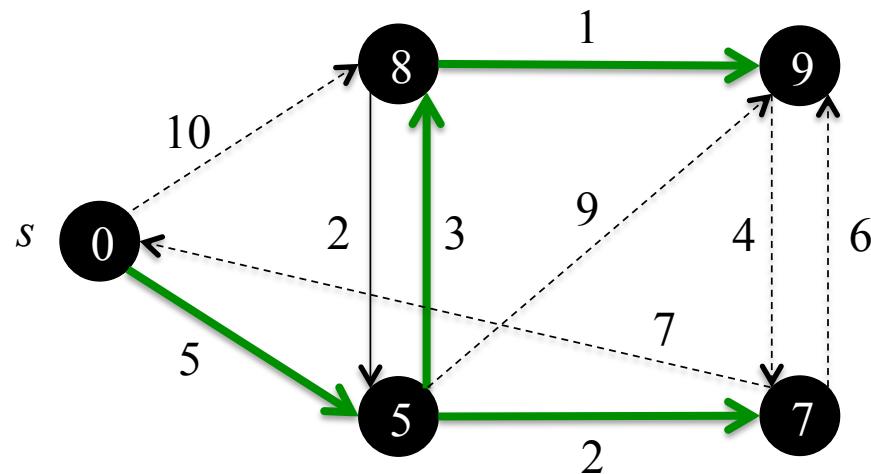


in E_π



in E_π per rilassamento

Esecuzione dell'alg. di Dijkstra



in $S = V_\pi$



in E_π

Correttezza di Dijkstra

Teorema. L'algoritmo di Dijkstra eseguito su un grafo orientato pesato $G = (V, E)$ con pesi non negativi w dalla sorgente s termina con $u.d = \delta(s, u)$ per ogni $u \in V$.

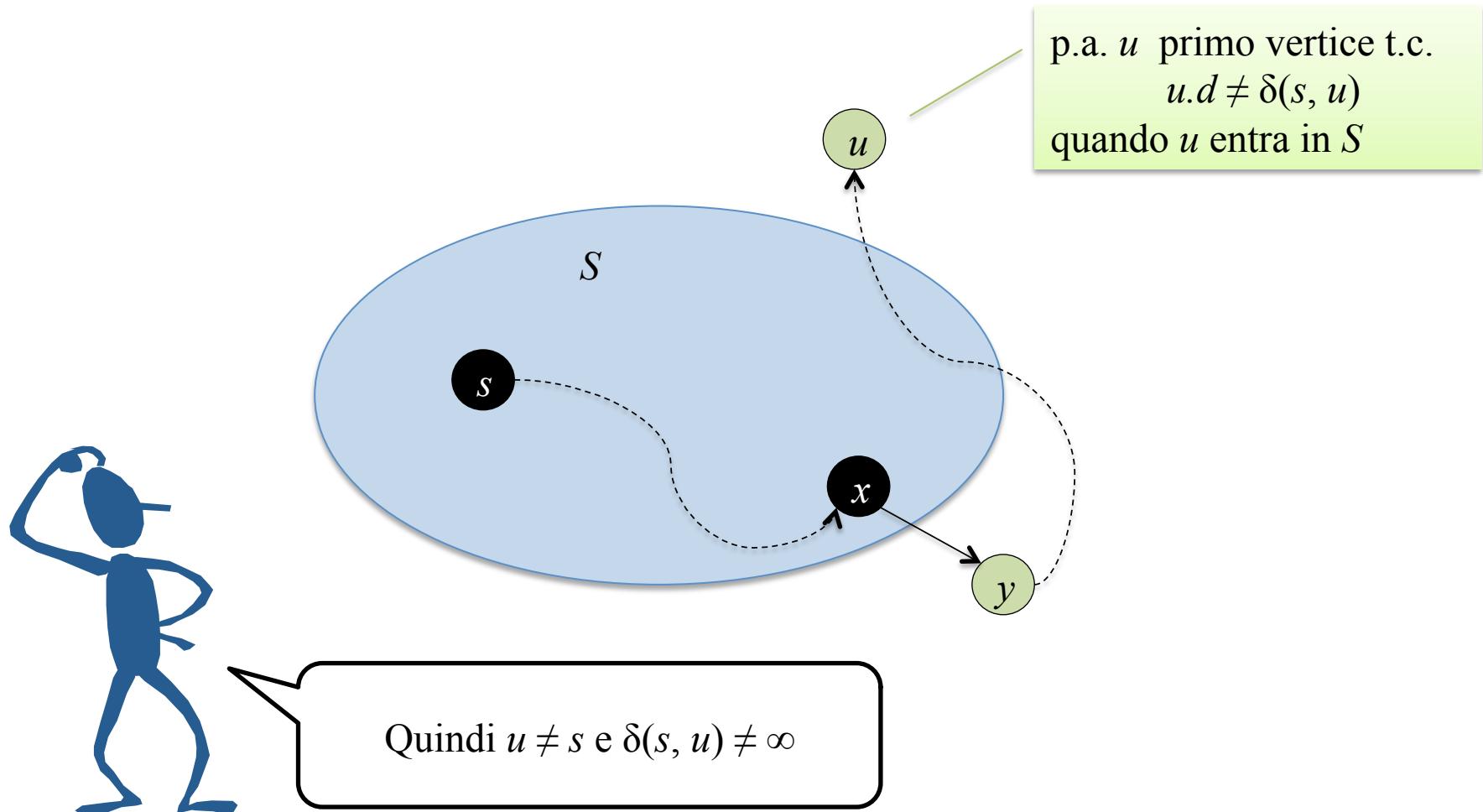
$$Q = V \setminus S \text{ e } \forall v \in S. v.d = \delta(s, v)$$



E' sufficiente provare che
questo è un invariante
dell'algoritmo

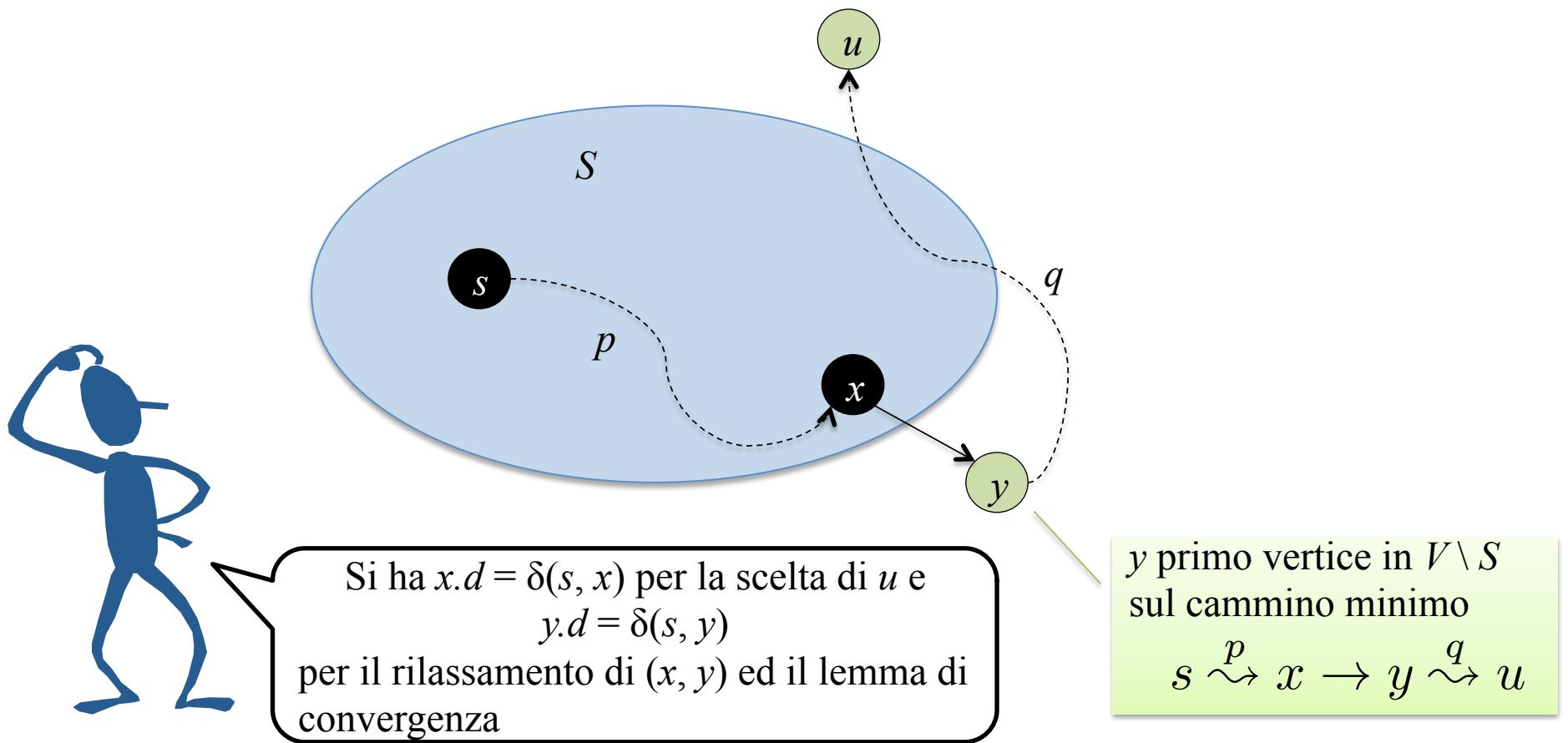
Correttezza di Dijkstra

$$Q = V \setminus S \text{ e } \forall v \in S. v.d = \delta(s, v)$$



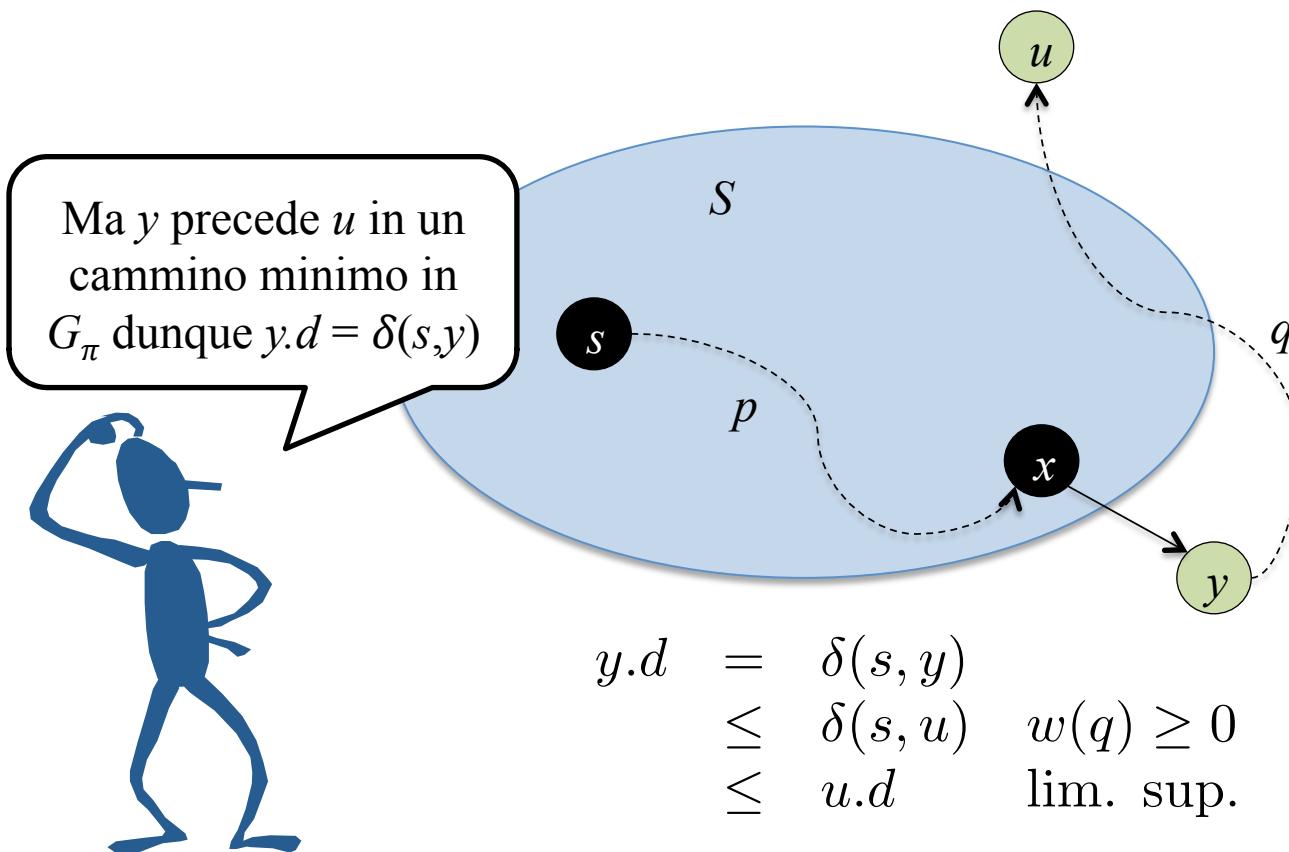
Correttezza di Dijkstra

$$Q = V \setminus S \text{ e } \forall v \in S. v.d = \delta(s, v)$$



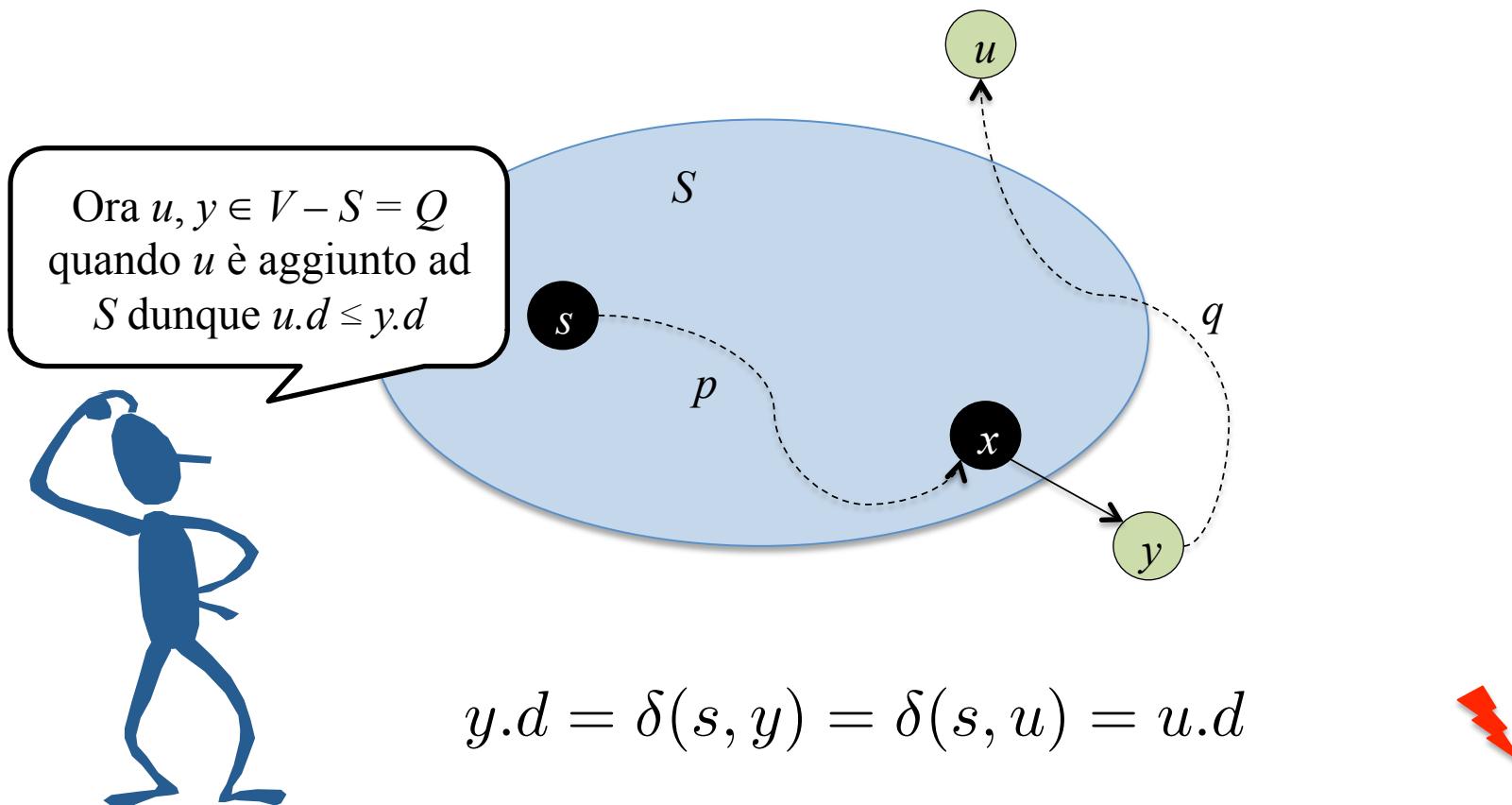
Correttezza di Dijkstra

$$Q = V \setminus S \text{ e } \forall v \in S. v.d = \delta(s, v)$$

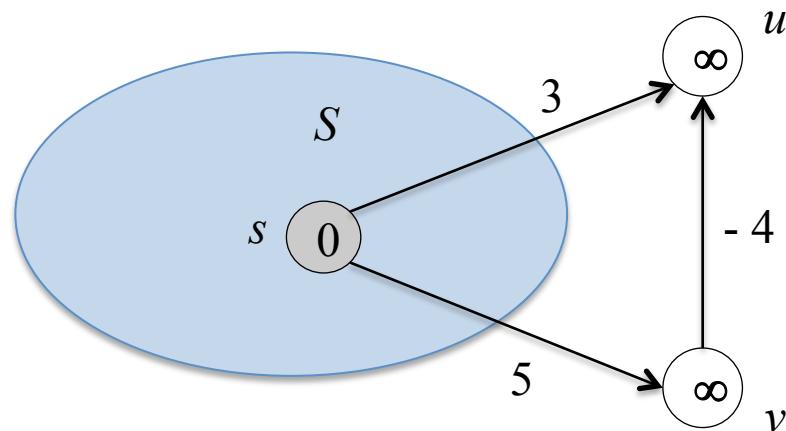


Correttezza di Dijkstra

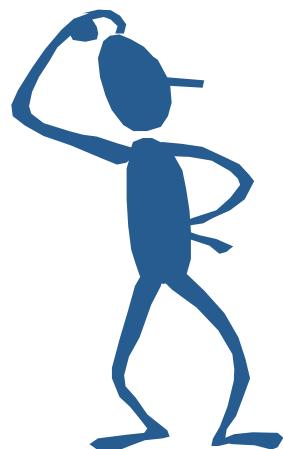
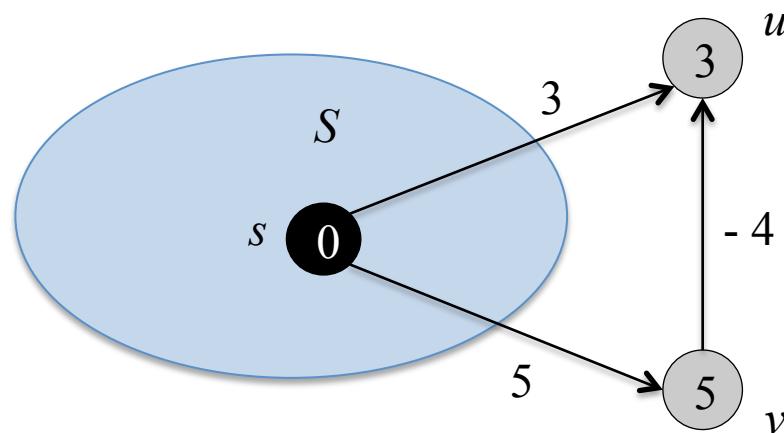
$$Q = V \setminus S \text{ e } \forall v \in S. v.d = \delta(s, v)$$



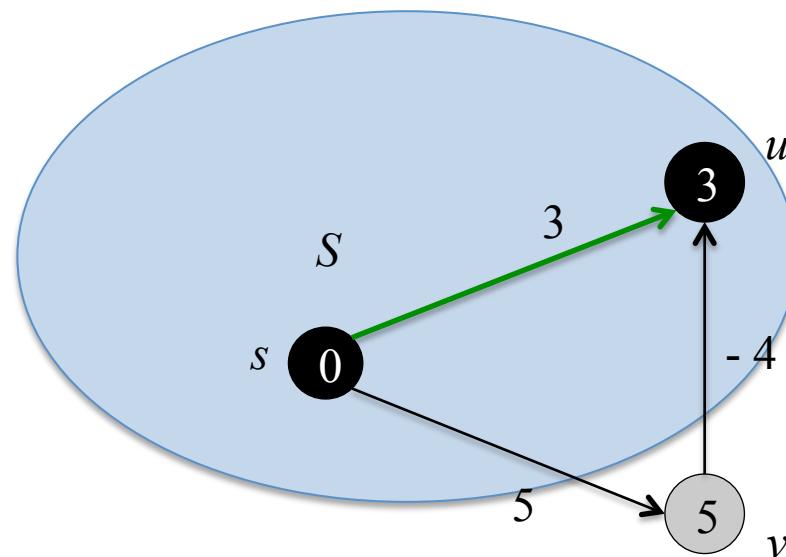
Necessità che w non sia negativa



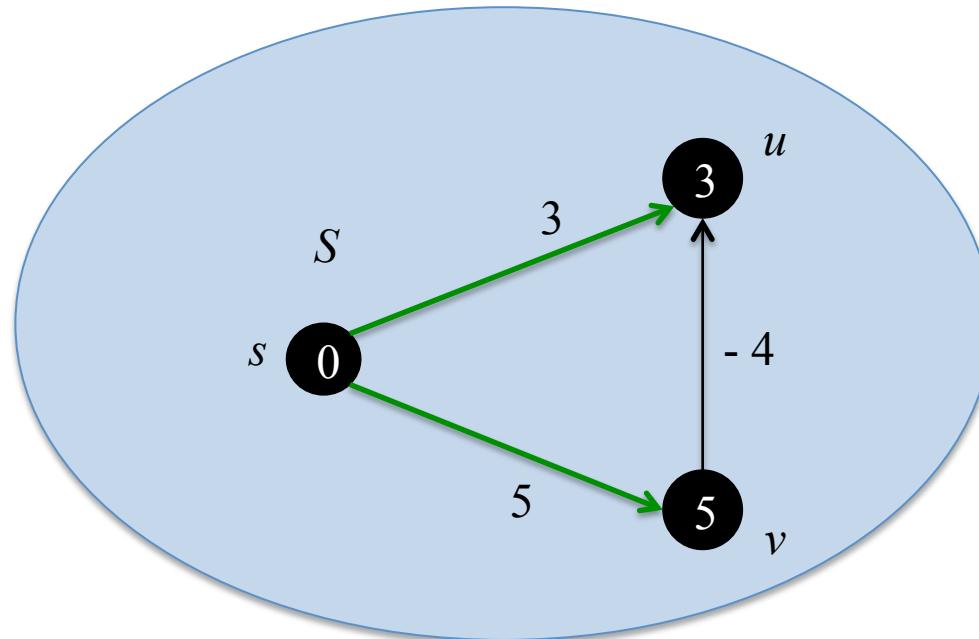
Necessità che w non sia negativa



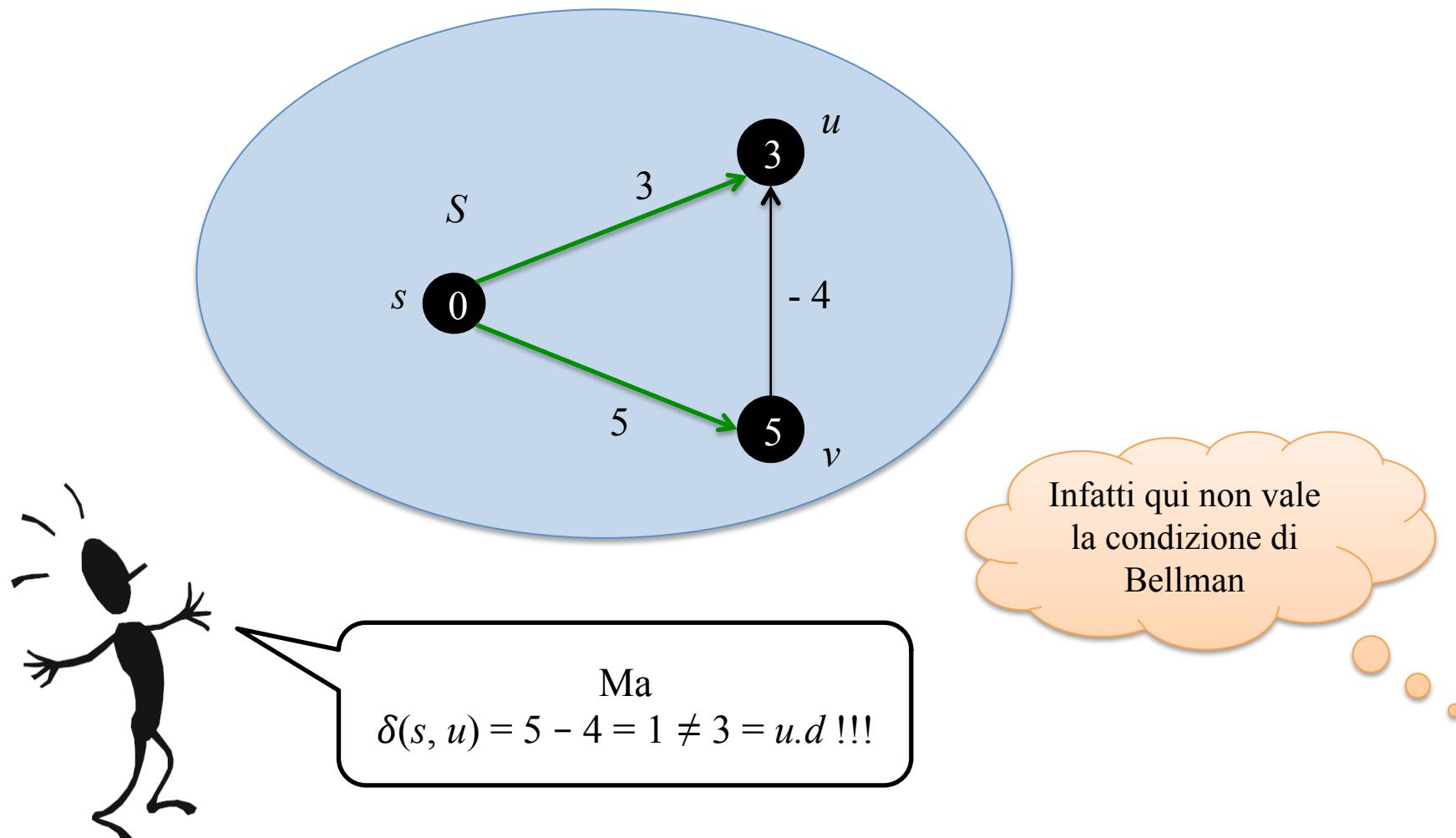
Necessità che w non sia negativa



Necessità che w non sia negativa



Necessità che w non sia negativa



Terminazione e complessità



- Se l'invariante è mantenuto, il sottografo $G_\pi = (V_\pi, E_\pi)$ è un albero di cammini minimi radicato in s
- Ogni vertice passa da $Q = V \setminus S$ ad S
- La struttura di Dijkstra è analoga a quella di Prim, dunque Dijkstra è $O(|E| \log |V|)$

Capitolo 8

Grafi

8.1 Simulazione di algoritmi noti

Esercizio 1.

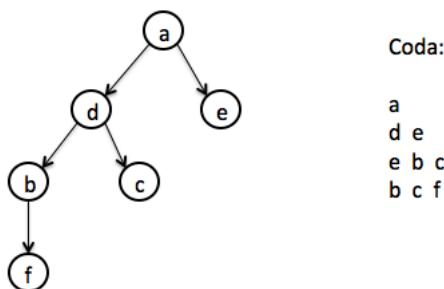
- Si effettui la visita in ampiezza (BFS) del seguente grafo orientato a partire dal nodo a riportando l'albero generato dalla visita e il contenuto della coda utilizzata durante la visita dopo gli inserimenti degli adiacenti non ancora visitati di ciascun vertice:

$$\begin{aligned} a &: d, e \\ b &: a, c, e, f \\ c &: a, e \\ d &: b, c, e \\ e &: a, d \\ f &: d, e \end{aligned} \tag{8.1}$$

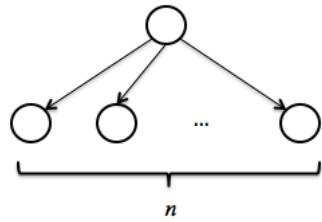
- Ricordando che un grafo si dice *completo* se ogni coppia di vertici è collegata da un arco, si disegni l'albero generato da una visita in ampiezza effettuata su un grafo completo con $n + 1$ vertici.

Soluzione 1.

- L'albero generato dalla BFS del grafo a partire dal vertice a (a sinistra) e lo stato della coda dopo ciascun inserimento (a destra) risultano:



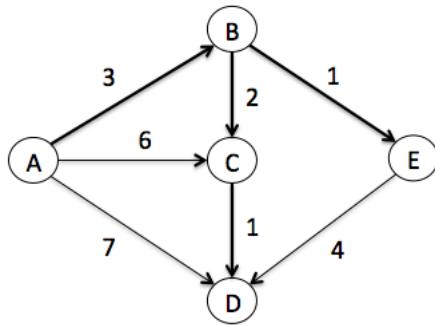
- Poiché il grafo è completo, dato un vertice qualunque r esistono gli archi $(r, v_1), \dots, (r, v_n)$ dove v_1, \dots, v_n sono i rimanenti n archi. Pertanto una BFS che comincia da un qualsiasi r genera un unico albero in cui tutti gli altri vertici (se ve ne sono, ossia se $n > 1$) sono figli del nodo r :



Esercizio 2. Si applichi l'algoritmo di Dijkstra al grafo riportato sotto con le liste di adiacenti a partire dal nodo A . Riportare il contenuto della coda di priorità prima di ogni estrazione.

$A : B(3), C(6), D(7)$
 $B : C(2), E(1)$
 $C : D(1)$
 $D :$
 $E : D(4)$

Soluzione 2. Il grafo, in cui gli archi di maggior spessore sono quelli che compongono i cammini minimi da A scelti dall'algoritmo di Dijkstra, risulta:



Quelli che seguono sono gli stati della coda (rappresentata come uno heap minimo in forma di array) prima dell'estrazione di ciascun vertice:

$(A, 0), (B, \infty), (D, \infty), (C, \infty), (E, \infty)$
 $(B, 3), (D, 7), (C, 6), (E, \infty)$
 $(E, 1), (D, 7), (C, 5)$
 $(C, 5), (D, 7)$
 $(D, 6)$

Esercizio 3. Si applichi l'algoritmo di Dijkstra al grafo orientato pesato rappresentato con le liste di adiacenza qui sotto riportato (il numero tra parentesi è il peso dell'arco), a partire dal nodo A :

- A: B(3), C(2), D(7)
- B: E(4)
- C: B(5), D(4), E(6), F(6), G(9)
- D: E(7), G(3)
- E: F(4), G(4)

- F: G(2)
- G:

Si spieghi a parole l'uso della coda di priorità nell'esecuzione dell'algoritmo di Dijkstra.

Soluzione 3. La seguente tabella riporta il contenuto della coda di priorità prima di ogni estrazione (omettendo i nodi con distanza infinita e quelli già estratti).

B	C	D	E	F	G
3,A	2,A	7,A			
3,A		6,C	8,C	8,C	11,C
		6,C	7,B	8,C	11,C
			7,B	8,C	9,D
				8,C	9,D
					9,D

I cammini minimi con relativo peso sono

$$\begin{aligned}
 & A \rightarrow C : 2 \\
 & A \rightarrow B : 3 \\
 & A \rightarrow C \rightarrow D : 6 \\
 & A \rightarrow B \rightarrow E : 7 \\
 & A \rightarrow C \rightarrow F : 8 \\
 & A \rightarrow C \rightarrow D \rightarrow G : 9
 \end{aligned} \tag{8.2}$$

La coda di priorità contiene i nodi per i quali non si conosce il cammino minimo. La priorità associata a ciascun nodo è la stima del cammino minimo verso il nodo stesso. L'estrazione del nodo con stima minima corrisponde ad aver trovato il cammino minimo verso il nodo. Quando un nodo viene estratto, bisogna verificare se si può migliorare il cammino verso i nodi che sono ancora presenti nella coda.

8.2 Problemi

Esercizio 4. Un grafo non orientato $G = (V, E)$ si dice *bipartito* se esiste un insieme $S \subseteq V$ tale che per ogni $(u, v) \in E$ si abbia $u \in S$ e $v \in V \setminus S$ oppure $v \in S$ e $u \in V \setminus S$.

Si trovi un algoritmo di complessità $O(|V| + |E|)$ per decidere se G sia bipartito.

Soluzione 4. La soluzione proposta si basa su di una BFS modificata, iterata su tutti i vertici non ancora visitati, di cui ci limitiamo a mostrare l'algoritmo di visita da un vertice BIPARTITE-COMP, così chiamata perché esplora una delle componenti connesse di G a partire da un vertice nella componente. Assumiamo che il campo $v.color$ assuma uno tra i tre valori *bianco*, *rosso*, *noncolorato* e che all'inizio tutti i vertici abbiano $v.color = \text{noncolorato}$:

```

BIPARTITE-COMP( $G, r$ )       $\triangleright G = (V, E)$  non orientato,  $r \in V$ 
 $Q \leftarrow \text{EMPTYQUEUE}$ 
 $r.color \leftarrow \text{bianco}$ 
 $\text{ENQUEUE}(r, Q)$ 
 $bipartito \leftarrow \text{true}$ 
 $\text{while } Q \neq \emptyset \text{ do}$ 
     $u \leftarrow \text{DEQUEUE}(Q)$ 
     $\text{for all } v \in \text{Adj}[u] \text{ do}$ 
         $\text{if } v.color = \text{noncolorato} \text{ then}$ 
             $\text{if } u.color = \text{bianco} \text{ then}$ 

```

```

    v.color = rosso
else
    v.color = bianco
    ENQUEUE(v, Q)
else
    if v.color = u.color then
        bipartito  $\leftarrow$  false
return bipartito

```

A rigore BIPARTITE-COMP(G, r) decide se la componente连通的 cui appartiene r è 2-colorabile, ossia se i suoi vertici possono essere colorati in modo tale che i vertici di ogni arco $(u, v) \in E$ abbiano colore diverso.

Esercizio 5. Si dimostri che il test di aciclicità per un grafo orientato $G = (V, E)$ può essere realizzato con un algoritmo di complessità $O(|V| + |E|)$.

Soluzione 5. Dimostriamo dapprima che G è ciclico se e solo se in ogni foresta generata da una visita DFS c'è almeno un arco all'indietro.

Se in una qualsiasi DF F di G c'è un arco all'indietro (v, u) allora esiste un albero $T \in F$ tale che v sia un discendente di u in T ; perciò $u \rightsquigarrow v \rightarrow u$ è un ciclo.

Supponiamo che G sia ciclico. Siano F una DF di G e $C \subseteq V$ un ciclo di G tale che $C.d = \min\{u.d \mid u \in C\}$ sia minimo tra i tempi di scoperta in F di vertici in V appartenenti a qualche ciclo. Allora esistono $u, v \in C$ t.c. $u.d = C.d$ e $u \rightsquigarrow v$ e $(v, u) \in E$. Per la scelta di C abbiamo che $u.d < v.d$, dunque u, v appartengono allo stesso albero T che include C ; ne segue che v è un discendente di u in T e (v, u) è un arco all'indietro di F .

Ora in una DFS la condizione per cui (v, u) è un arco all'indietro coincide con la scoperta di $u \in adj[v]$ prima che u esca dalla pila (la frontiera) ossia quando $u.color = grigio$; dunque è sufficiente modificare la DFS in modo che ritorni **false** non appena si scopre un arco all'indietro, e **true** se la visita di tutto il grafo termina senza scoprire un tale arco. La complessità dunque è quella della DFS ossia $O(|V| + |E|)$.

Esercizio 6. Si dimostri che se G è un grafo non orientato connesso con archi pesati, allora esiste un MST di G , senza usare la correttezza di Kruskal o Prim.

Soluzione 6. Dimostriamo dapprima che esiste un albero T che sia una copertura di $G = (V, E)$. Se G è connesso ogni visita, non importa se BFS o DFS, genererà una foresta con un unico albero, visto che tutti i vertici sono raggiungibili gli uni dagli altri. Dunque quest'albero è una copertura di G . Poiché E è finito, anche l'insieme $\wp(E)$ delle parti di E è finito; ma un albero di copertura T (inteso come insieme di archi) è un sottoinsieme di E , dunque l'insieme degli alberi di copertura di G è un insieme $\{T_1, \dots, T_n\} \subseteq \wp(E)$ finito non vuoto. Allora l'insieme di numeri reali $r = w(T)$ per qualche albero T di copertura di G è anch'esso finito e non vuoto; dunque ammette un minimo, corrispondente ad un MST di G .