Grafi: visita in ampiezza

Una presentazione alternativa (con ulteriori dettagli)

Consideriamo la versione "concreta" dell'algoritmo di visita generica con costruzione del sottografo dei predecessori:

***VISITA* (G, s)**

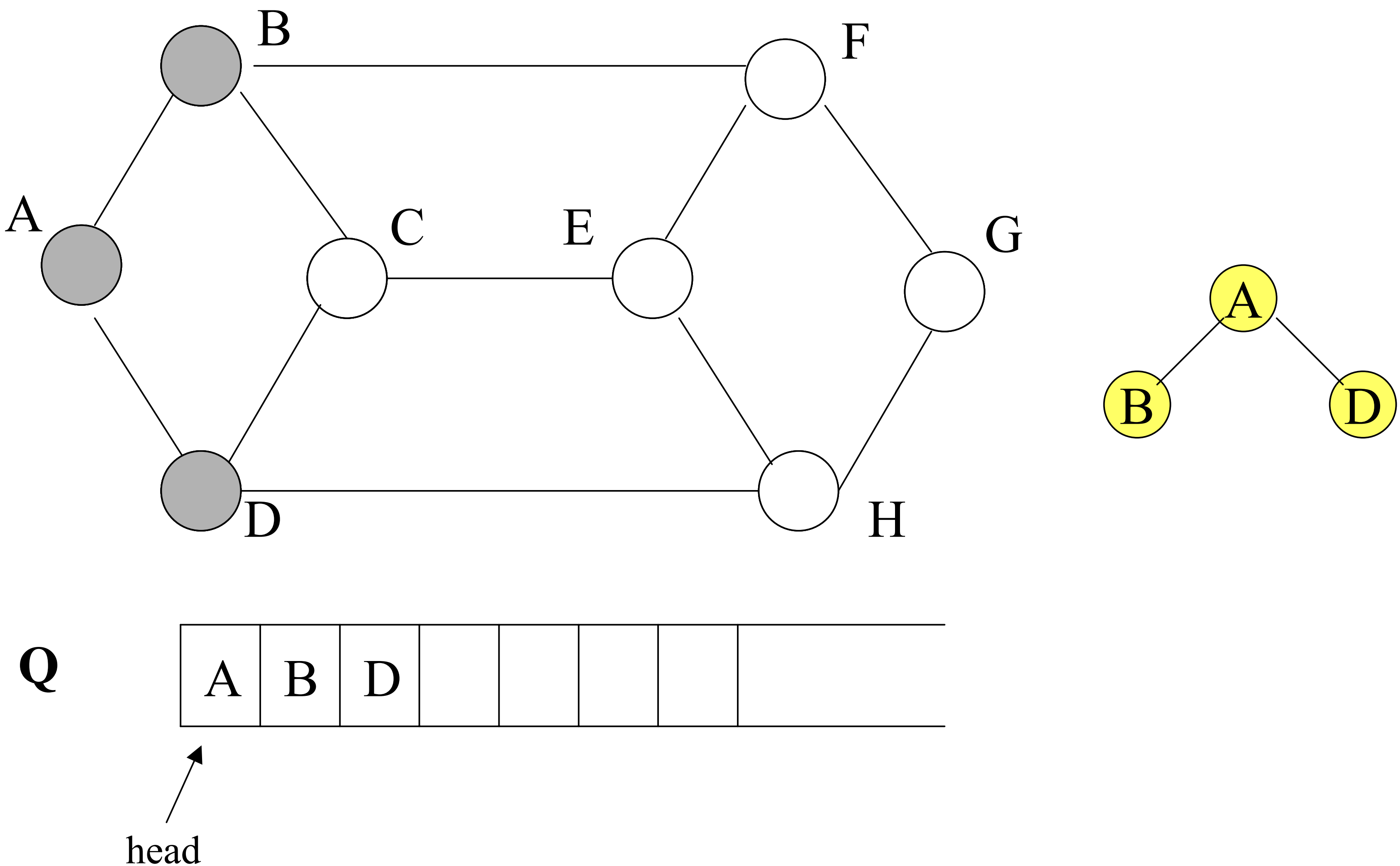
```
D ← make_empty
color [s] ← gray
add (D, s)
while not empty(D)          do
    u ← first (D)
    {if u non visitato then visitalo}
    if c'è v bianco ∈ ADJ [u]
        then color [v] ← gray
             π[v] ← u
             add (D, v)
        else color [u] ← black
             remove_first (D)
```

La struttura dati D è una

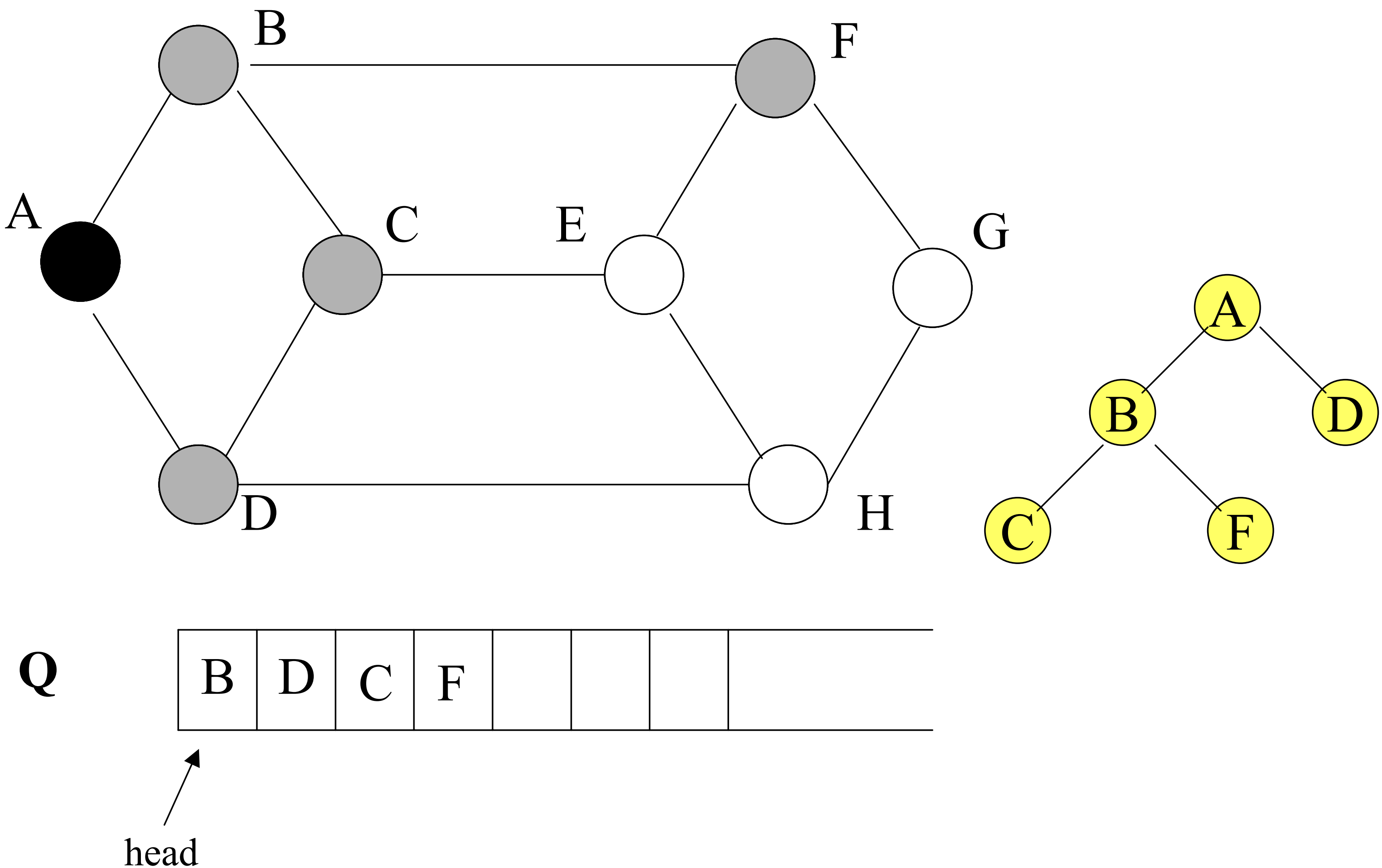
CODA

Visita in ampiezza o Breadth-First-Search (BFS)

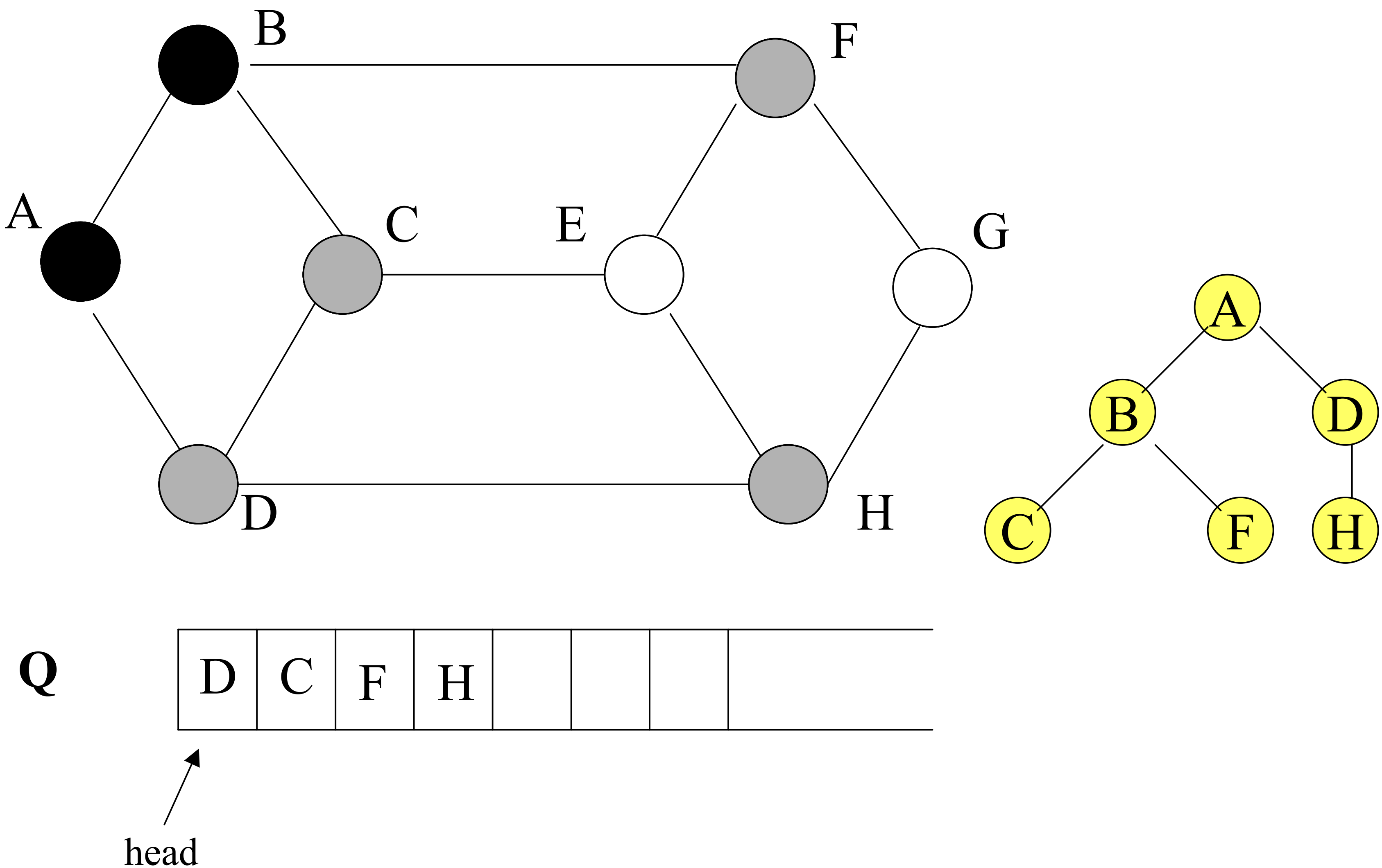
Esempio (1/9)



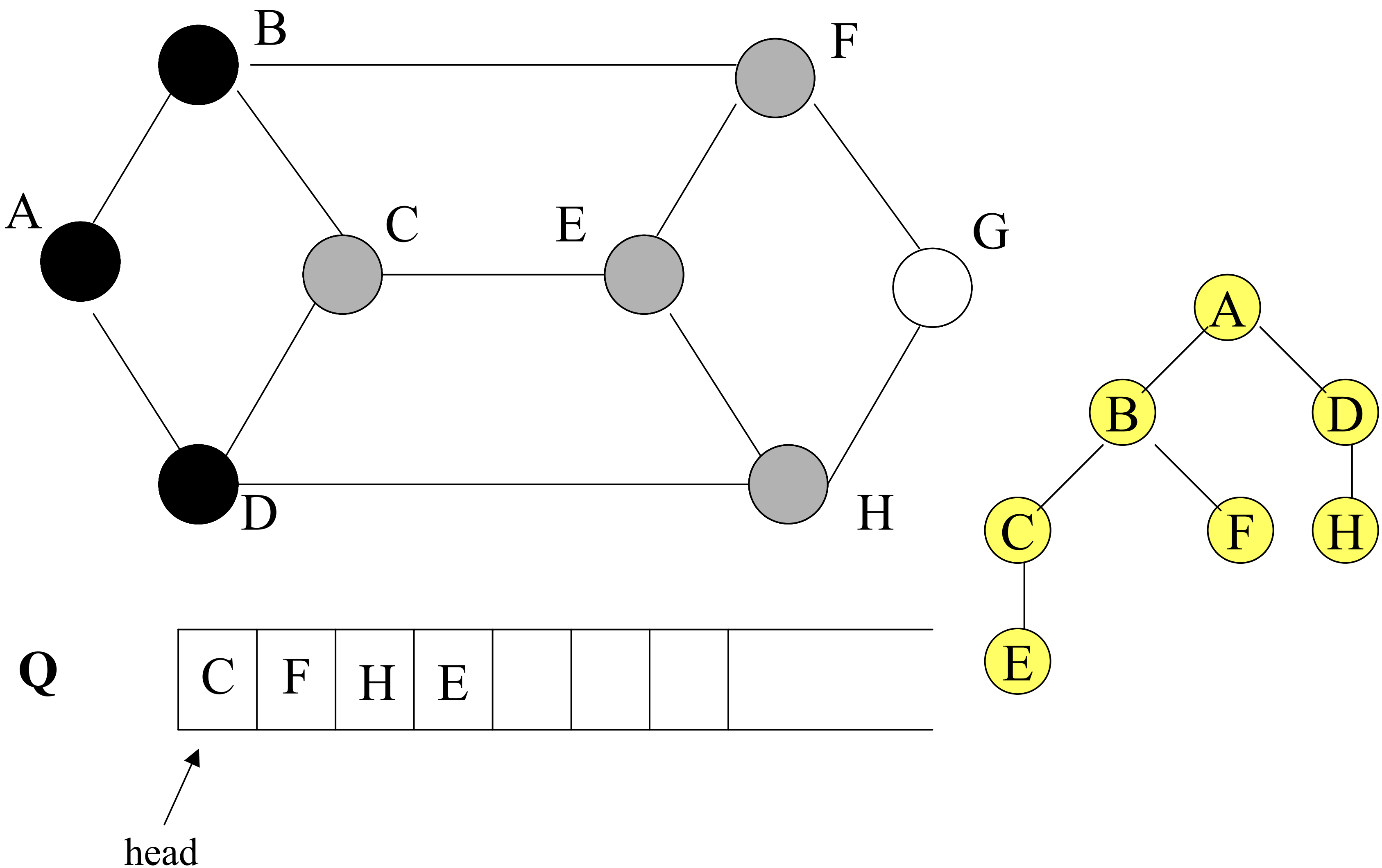
Esempio (2/9)



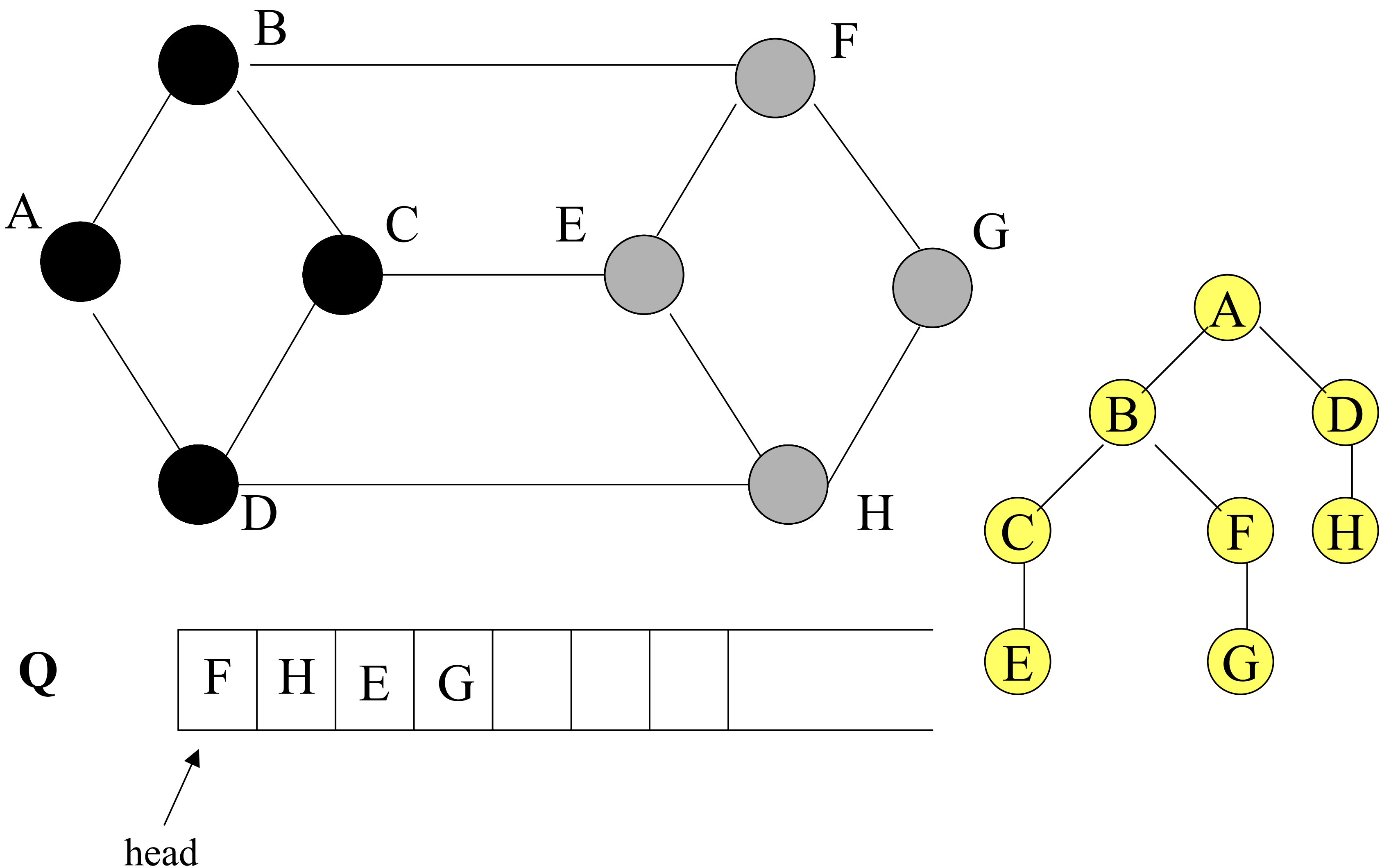
Esempio (3/9)



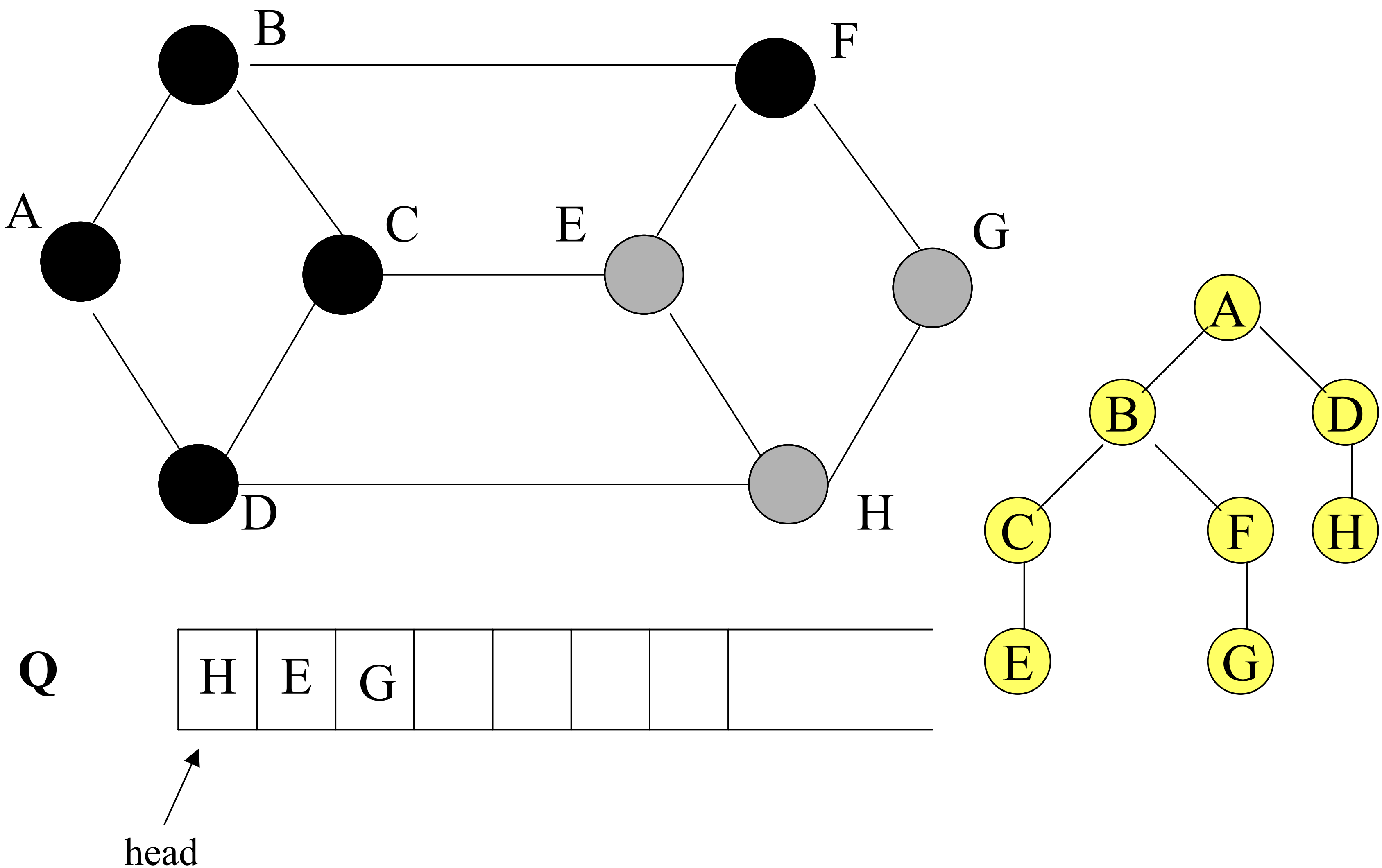
Esempio (4/9)



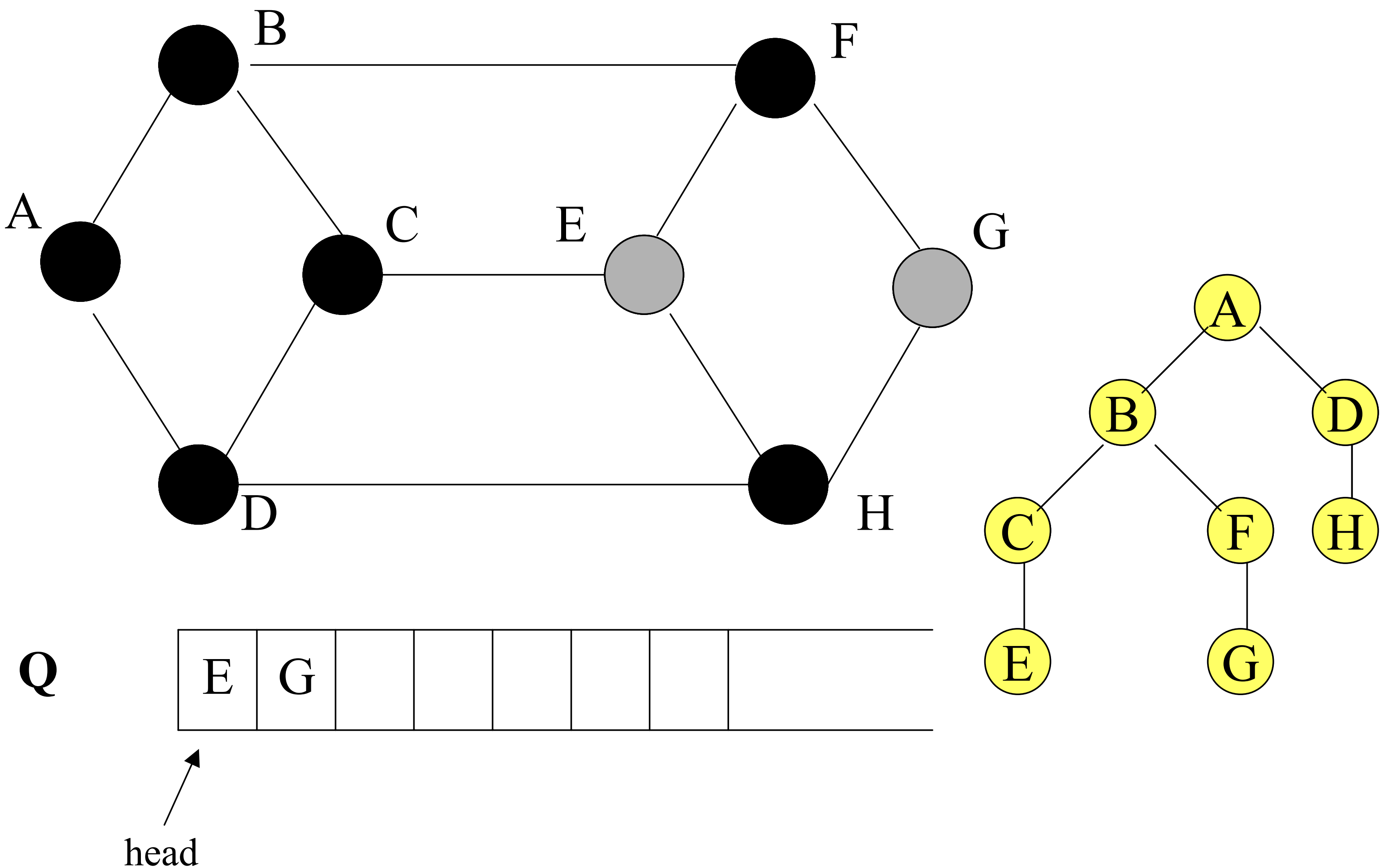
Esempio (5/9)



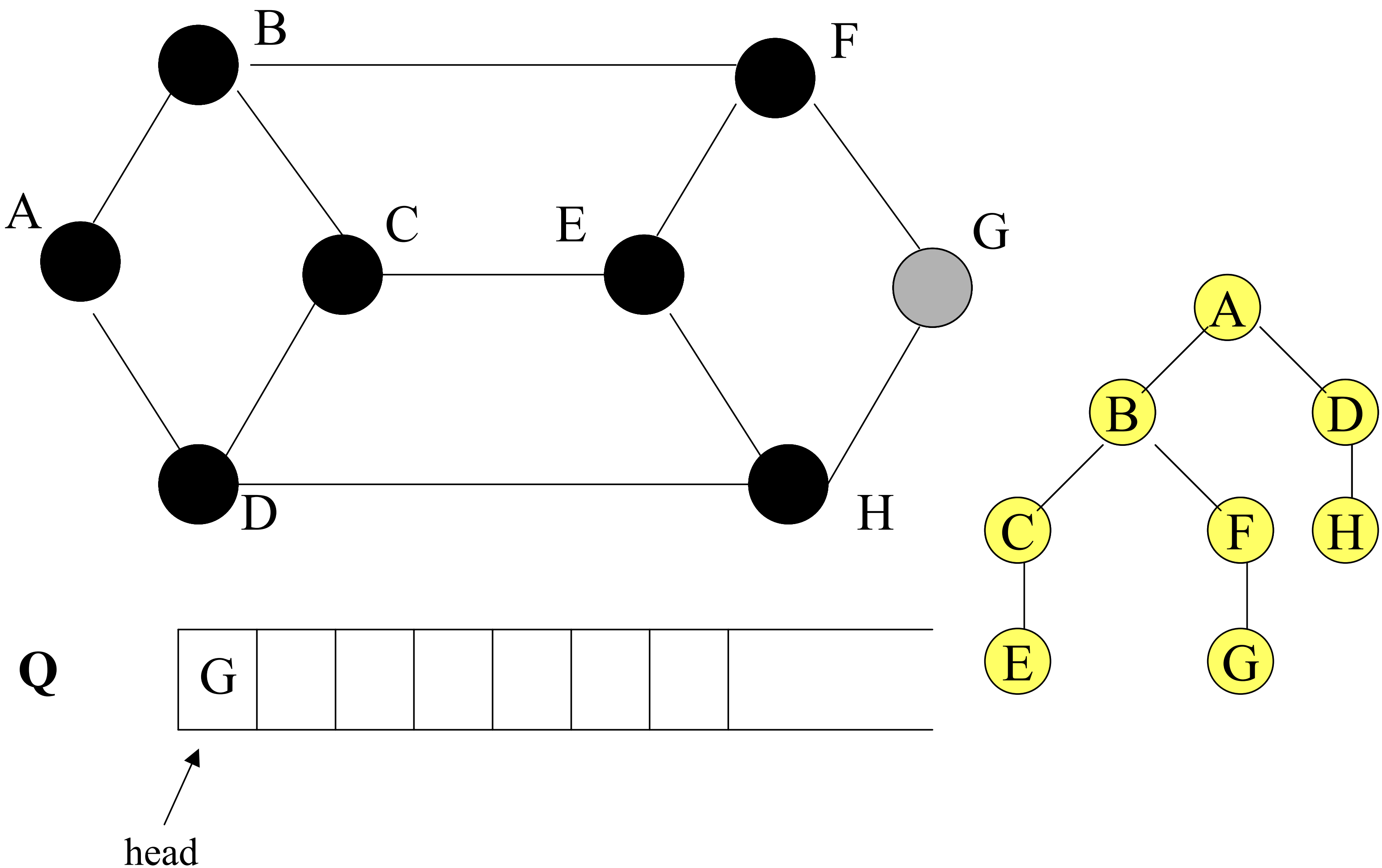
Esempio (6/9)



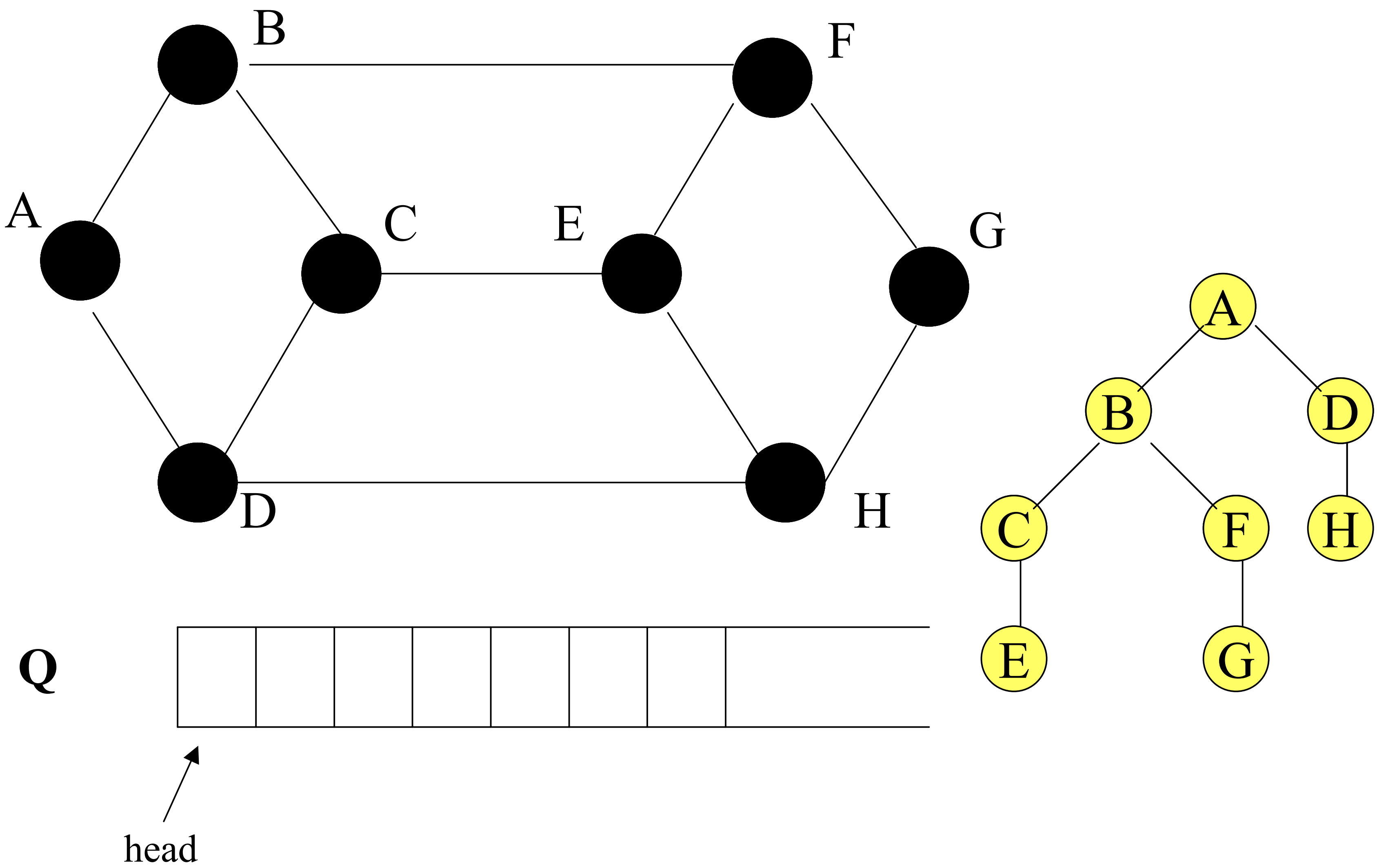
Esempio (7/9)



Esempio (8/9)



Esempio (9/9)



Due versioni dell'algoritmo di visita in ampiezza

BFS-VISITA (G, s)

$Q \leftarrow$ make_empty_queue
color [s] $\leftarrow$ gray
enqueue (Q, s)
while not_empty (Q) **do**
u $\leftarrow$ head (Q)

Poichè la head della coda non cambia finché ci sono adiacenti bianchi, l'algoritmo può essere modificato nel modo seguente:

{**if** u non visitato **then** visitalo}
if c'è v bianco $\in$ ADJ [u]
then color [v] $\leftarrow$ gray
π[v] $\leftarrow$ u
enqueue (Q, v)
else color [u] $\leftarrow$ black
dequeue (Q)

{visita u}
for ogni v $\in$ ADJ [u] **do**
if color [v] = white
then color [v] $\leftarrow$ gray
π[v] $\leftarrow$ u
enqueue (Q, v)
color [u] $\leftarrow$ black
dequeue (Q)

na terza versione “ancora più concreta” dell’algoritmo

FS-VISITA (G, s)

```
Q ← make_empty_queue
color [s] ← gray
enqueue (Q, s)
while not_empty (Q) do
    u ← head (Q)
    {visita u}
```

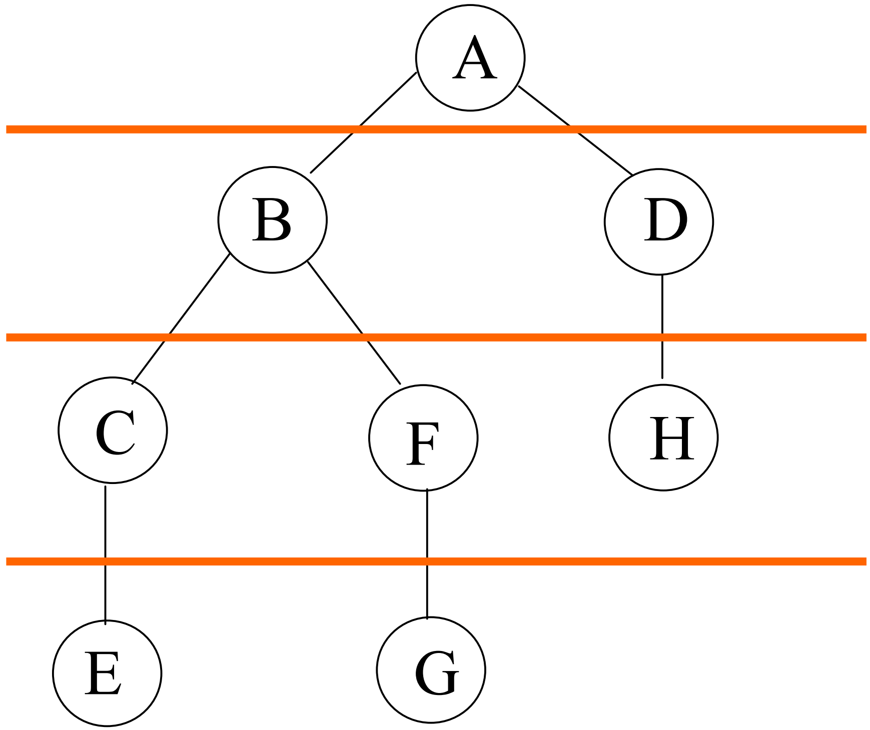
```
    for ogni v ∈ ADJ [u] do
        if color [v] = white
            then color [v] ← gray
                π[v] ← u
                enqueue (Q, v)
```

```
    Ptr ← ADJ [u]
    while Ptr ≠ nil do
        if color [Ptr.vtx] = white
            then color [Ptr.vtx] ← gra
                π[Ptr.vtx] ← u
                enqueue (Q, Ptr.vtx)
        Ptr ← Ptr.link
```

```
    color [u] ← black
    dequeue (Q)
```

Modifichiamo la (2a versione del)l’algoritmo per “sintetizzare” altre informazion

N.B. Nella visita in ampiezza Breadth First Search
l’albero viene costruito a livelli



Riscriviamo l’algoritmo associando ad ogni vertice ‘v’ un attributo ‘d[v]’, che ricorda il suo livello nell’albero ottenuto con la visita.

BFS-VISITA (G, s)

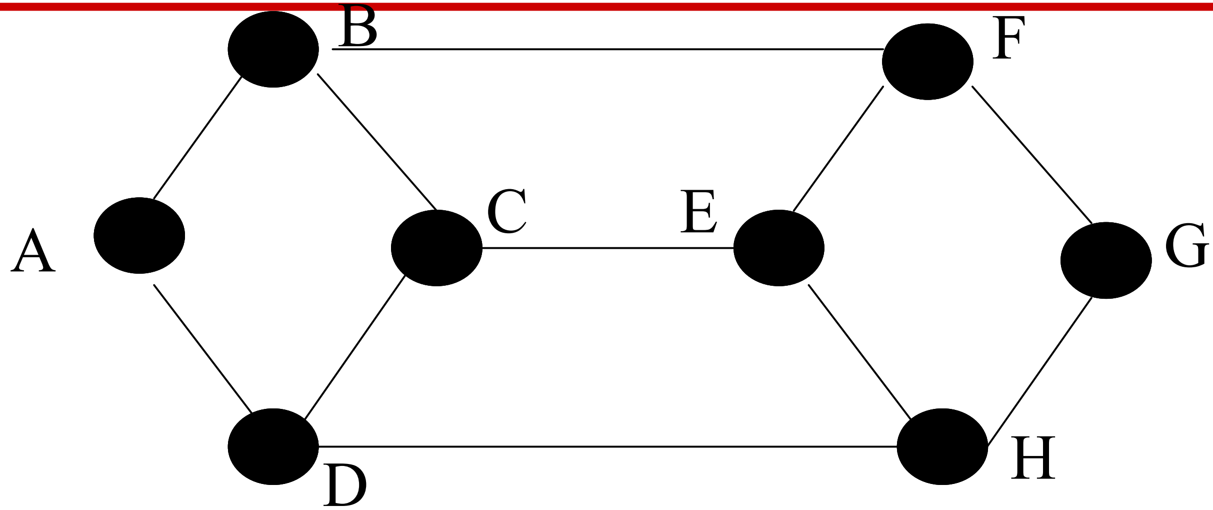
```
Q ← make_empty_queue
color [s] ← gray
d[s] ← 0
enqueue (Q, s)
while not_empty (Q) do
    u ← head (Q)
    {visita u}
    for ogni v ∈ ADJ [u] do
        if color [v] = white
            then color [v] ← gray
                 π[v] ← u
                 d[v] ← d[u] + 1
                 enqueue (Q, v)
    color [u] ← black
    dequeue (Q)
```

Bisogna inizializzare l'attributo d

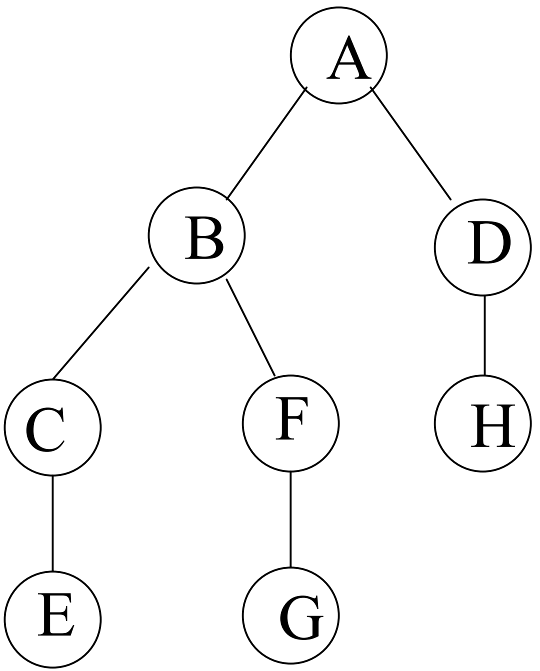
INIZIALIZZA (G)

```
for ogni u ∈ V do
    color [u] ← white
    π[u] ← nil
    d[u] ← ∞
```

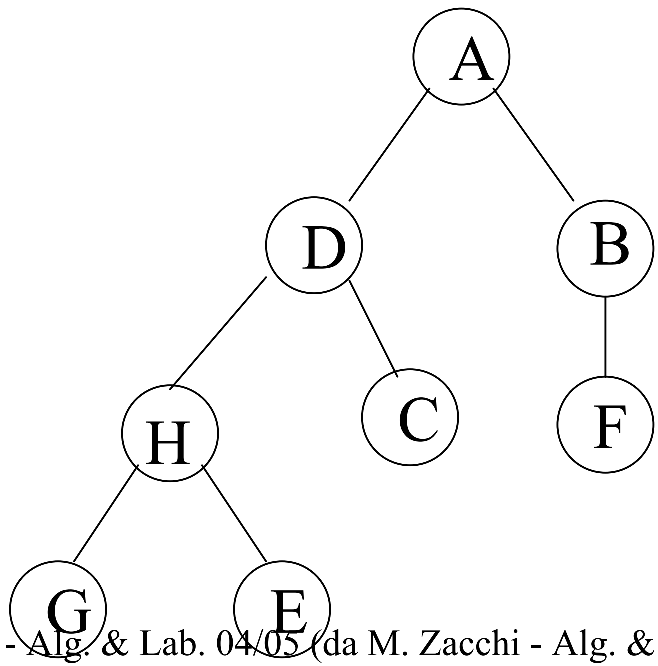
Riprendiamo l'esempio (1/2)

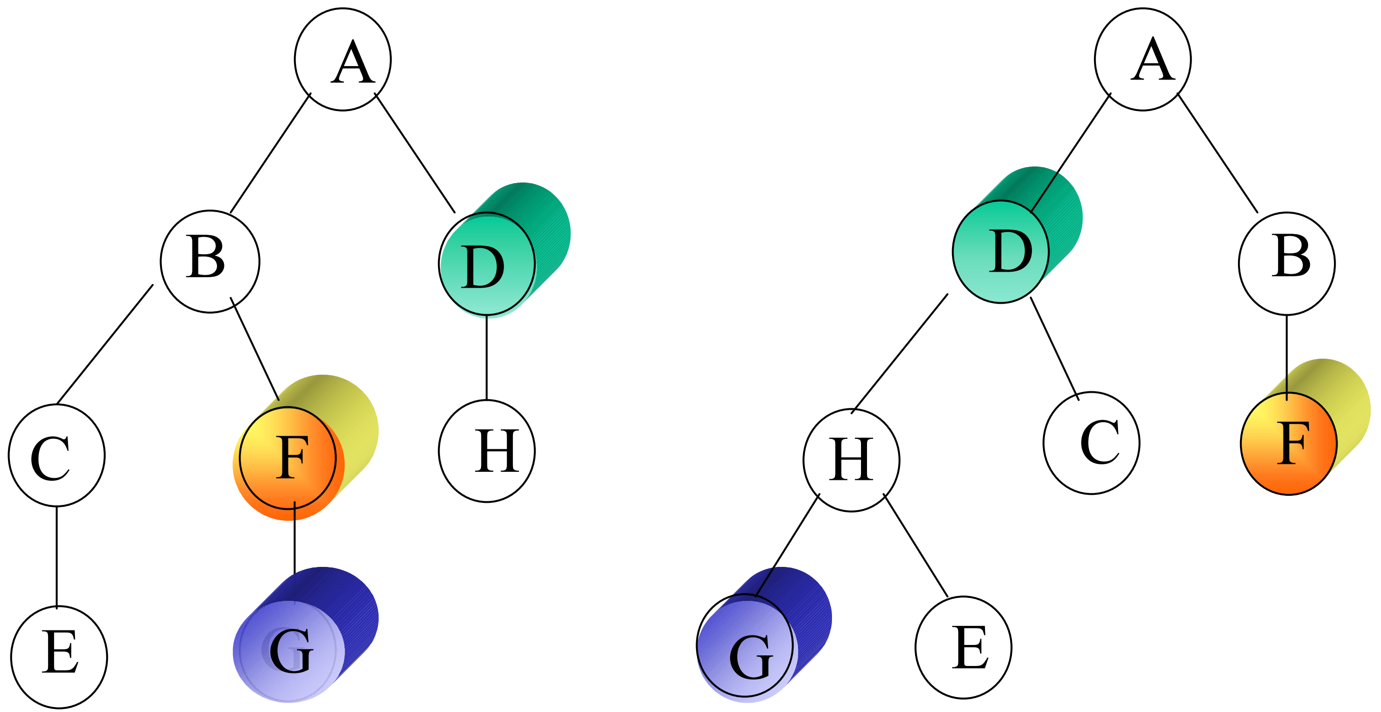


Vertici memorizzati nelle liste
di adiacenza in ordine alfabetico



Vertici memorizzati nelle liste
di adiacenza in ordine inverso





Teorema

Al termine di una BFS-VISITA si ha: $\forall v \in V[G] : d[v] = \delta(s, v)$.

$\delta(s, v)$ indica la distanza di v dal vertice sorgente (lunghezza di un cammino minimo da s a v).

Proprietà

1. In Q ci sono tutti e soli i vertici grigi
2. Se $<v_1, v_2, \ldots, v_n>$ è il contenuto di Q, allora:
   $d[v_n] \leq d[v_1] + 1$
   $d[v_i] \leq d[v_{i+1}]$ $\quad 1 \leq i \leq n-1$

Lemma

La seguente proprietà è un invariante dei due **while** di BFS-VISITA:
(INV4) $d[v] = \delta(s,v)$ per tutti i vertici v grigi o neri.

Dimostrazione

È sufficiente dimostrare che l'assegnazione $d[v] \leftarrow d[u] + 1$ rende $d[v] = \delta(s,v)$; la dimostrazione del lemma è una ovvia conseguenza di questo fatto.

Proprietà della visita in ampiezza (3/4)

Consideriamo in G un cammino minimo da s a v.
Per l'invariante INV3 tale cammino contiene almeno un vertice t grigio. Il cammino minimo può allora essere visto come concatenazione dei due cammini minimi da s a t e da t a v.

$\delta(s,v) = \delta(s,t) + \delta(t,v) \geq \delta(s,t) + 1$ $\quad$ ($\delta(t,v) \geq 1$ poichè $t \neq v$).

Ma il vertice t è in Q (t è grigio) e per l'invariante INV4, essendo $u = head(Q)$: $\quad d[u] \leq d[t] = \delta(s,t)$.
Si ottiene pertanto: $\delta(s,v) \geq \delta(s,t) + 1 = d[t] + 1 \geq d[u] + 1$.

Poichè $s \rightsquigarrow u \rightarrow v$ è un cammino in G da s a v, esso non può avere lunghezza minore di quella di un cammino minimo, allora $\delta(s,v) = d[u] + 1$ e l'assegnazione $d[v] \leftarrow d[u] + 1$ rende $d[v] = \delta(s,v)$.

Teorema.
Al termine dell'esecuzione di BFS si ha $d[v] = \delta(s,v)$ per tutti i vertici $v \in V$.

Dimostrazione.
Se v non è raggiungible da s allora $d[v]$ rimane $\infty = \delta(s,v)$.
Altrimenti v è nero e il teorema vale per il Lemma precedente.

- per ogni vertice v raggiungibile da s, il cammino da s a v sull'albero ottenuto con la visita è un cammino minimo.
- Il livello di un vertice nell'albero è indipendente dall'ordine in cui sono memorizzati i vertici nelle liste di adiacenza.

Esercizio

La correttezza della seconda (e della terza) versione dell'algoritmo di visita in ampiezza segue dalla correttezza della prima versione dell'algoritmo (che, a sua volta, è conseguenza immediata della correttezza dell'algoritmo di visita generica, che stata dimostrata con il metodo delle asserzioni). ANNOTATE (SCRIVENDO SULLA STAMPA DEI LUCIDI) IL CODICE DELLA PRIMA, DELLA SECONDA, E DELLA TERZA VERSIONE DELL'ALGORITMO CON LE ASSERZIONI CHE NE DIMOSTRANO LA CORRETTEZZA.