GRAFI: VISITA IN AMPIEZZA

[Deme, seconda edizione] cap. 12

Sezione 12.3.1

Visita in ampiezza - caratteristiche

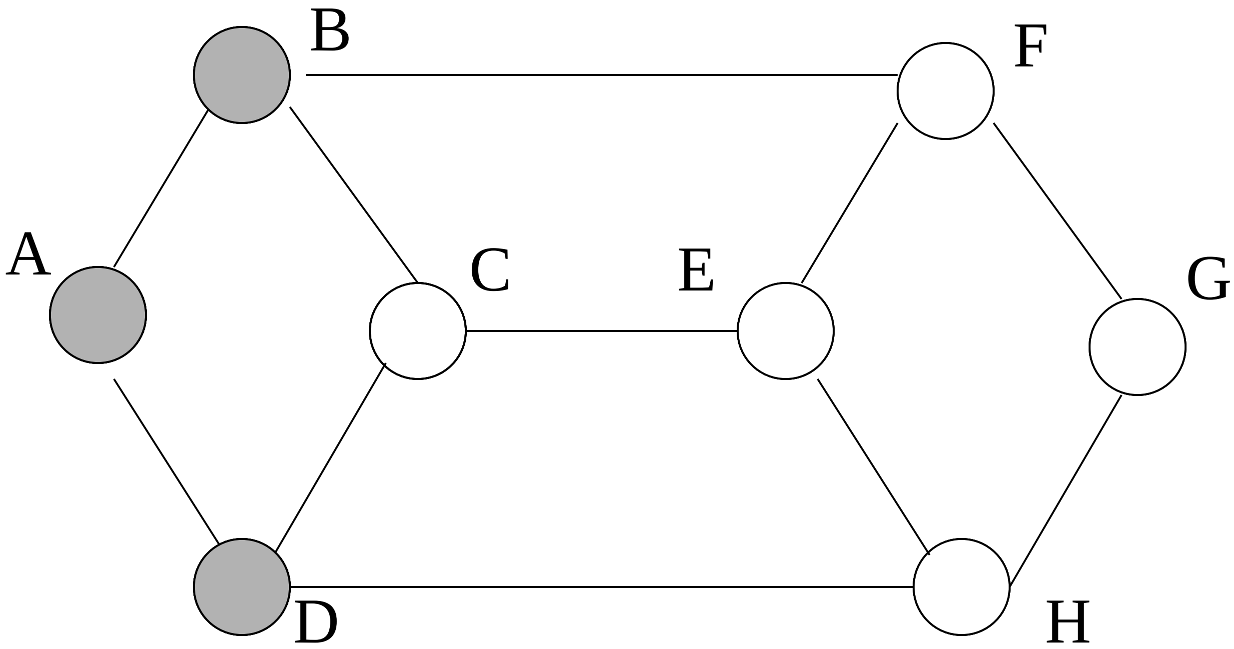
La visita in **ampiezza** (**BFS = breadth first search**) esamina i vertici del grafo in un ordine ben preciso, costruendo un albero di visita chiamato **albero BFS**.

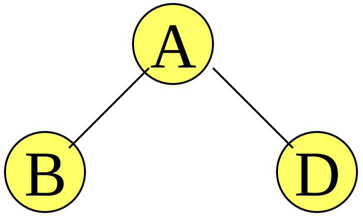
Nell'albero BFS, ogni vertice si trova il più vicino possibile alla radice.

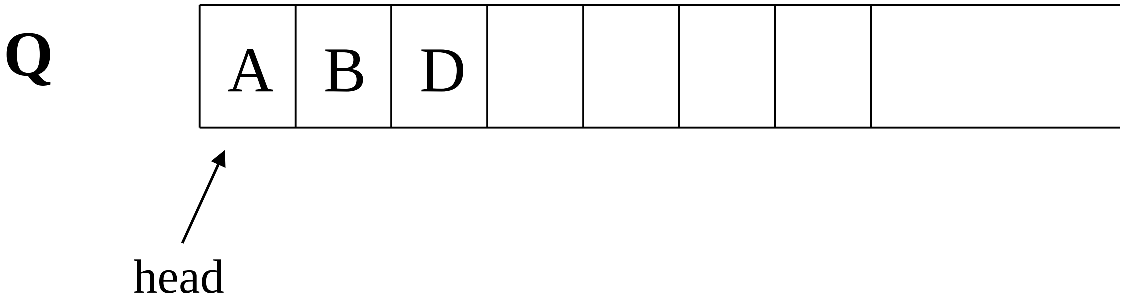
VisitaBFS si ottiene dall'algoritmo generico VISITA implementando la struttura dati D con una **coda**.

La coda (come già saprete) ha una politica di tipo **FIFO** (First In First Out) -> ciò vuol dire che sarà sempre il nodo **da più tempo nella coda** ad essere esaminato per primo.

Esempio (1/9)

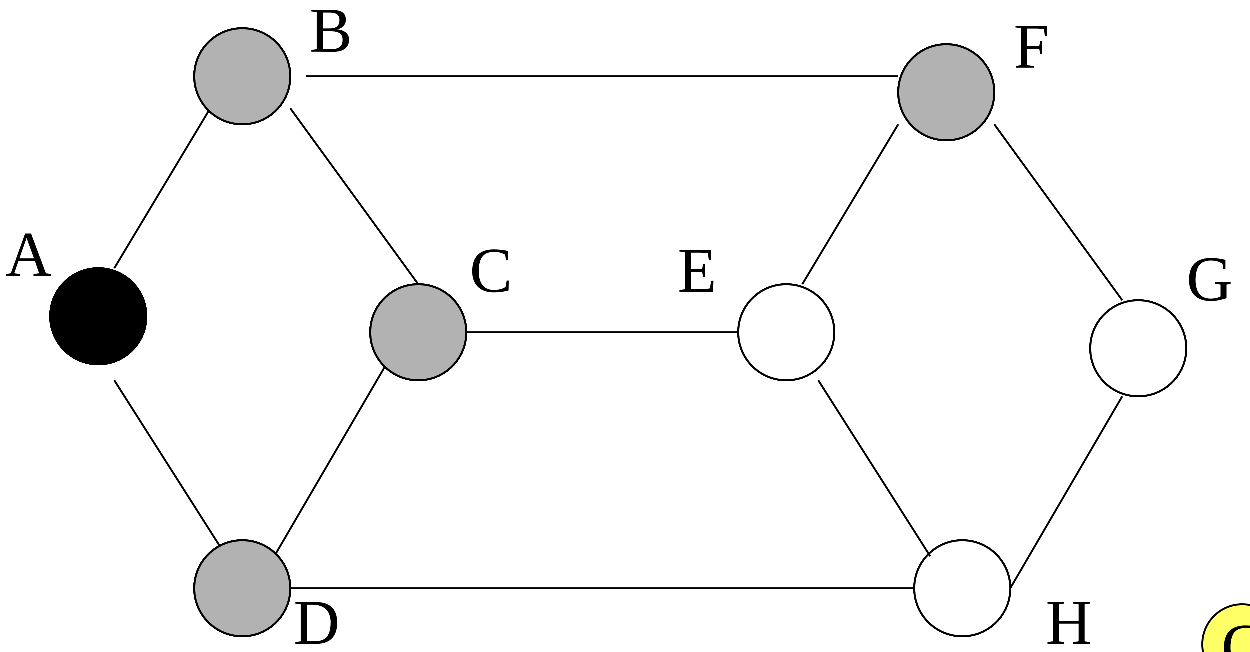


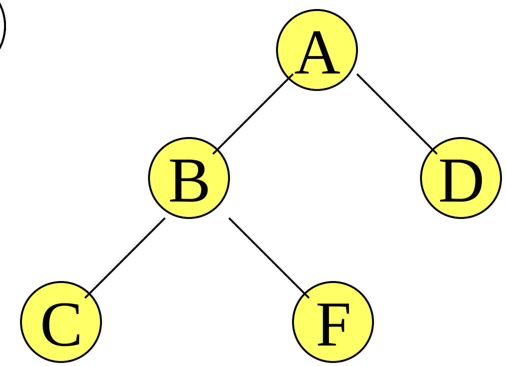


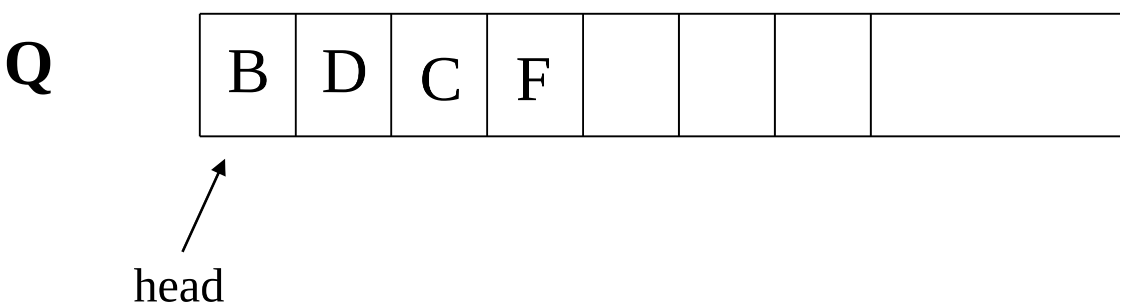


(da F. Damiani - Alg. & Sper. 06/07)

Esempio (2/9)

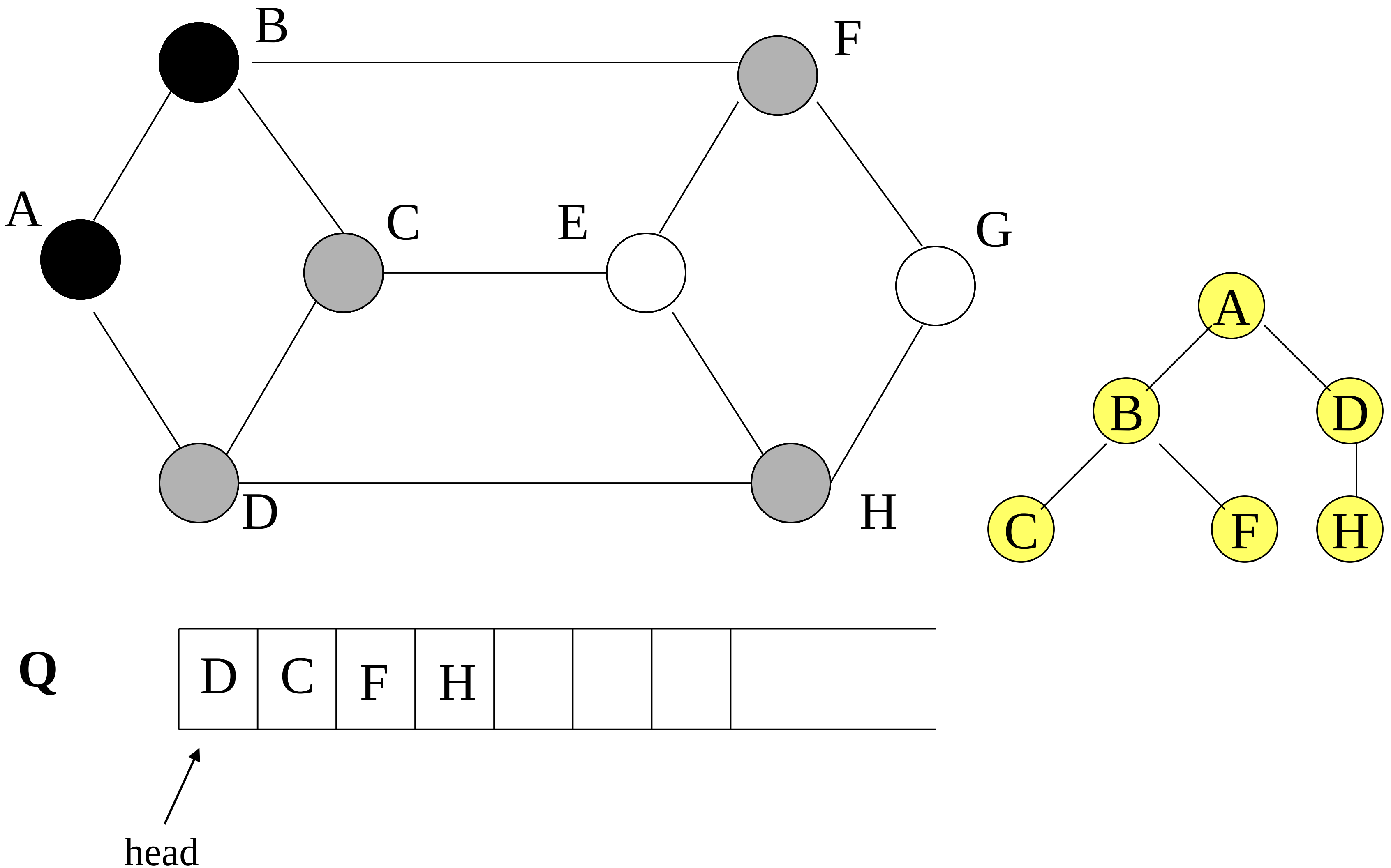






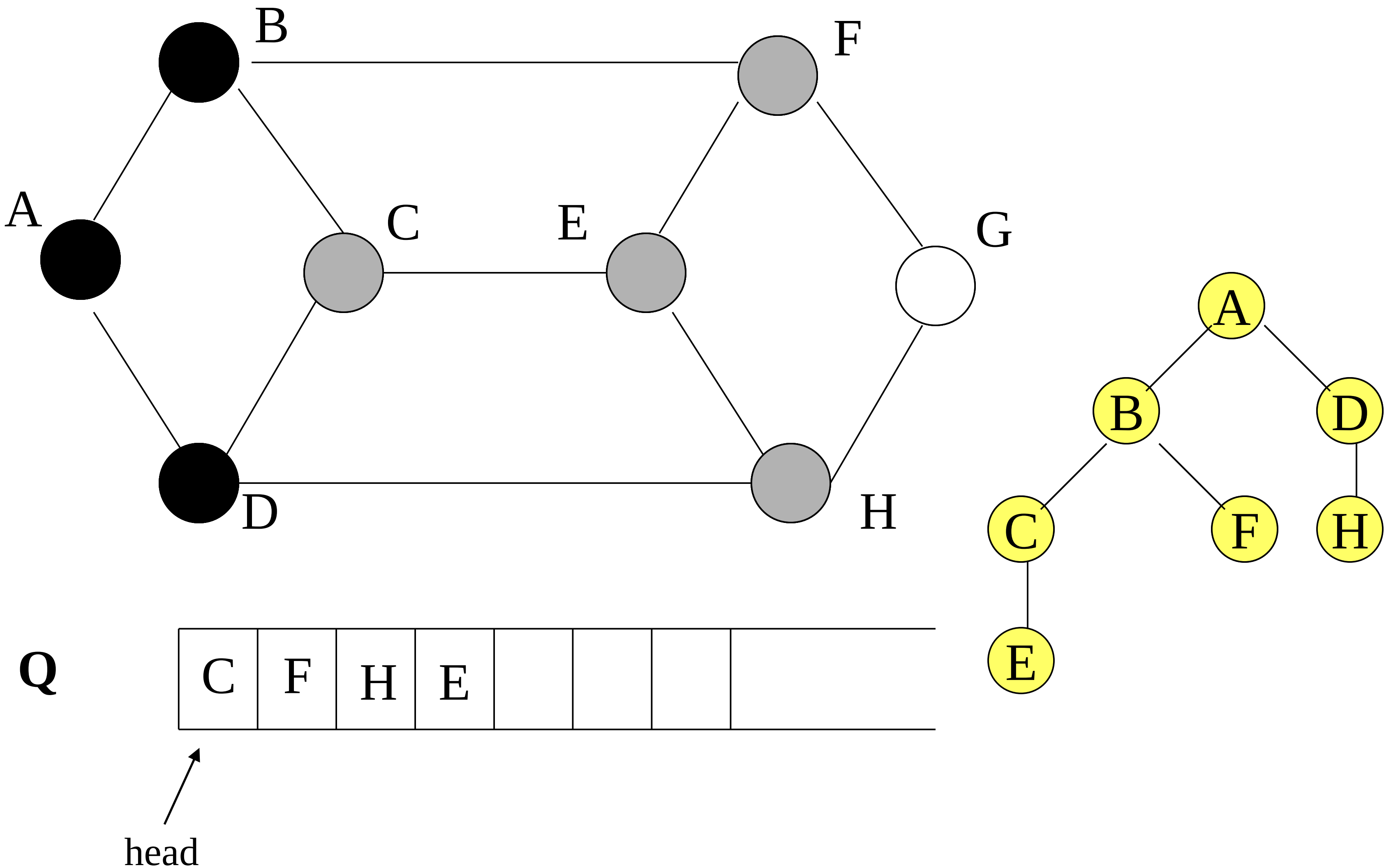
(da F. Damiani - Alg. & Sper. 06/07)

Esempio (3/9)



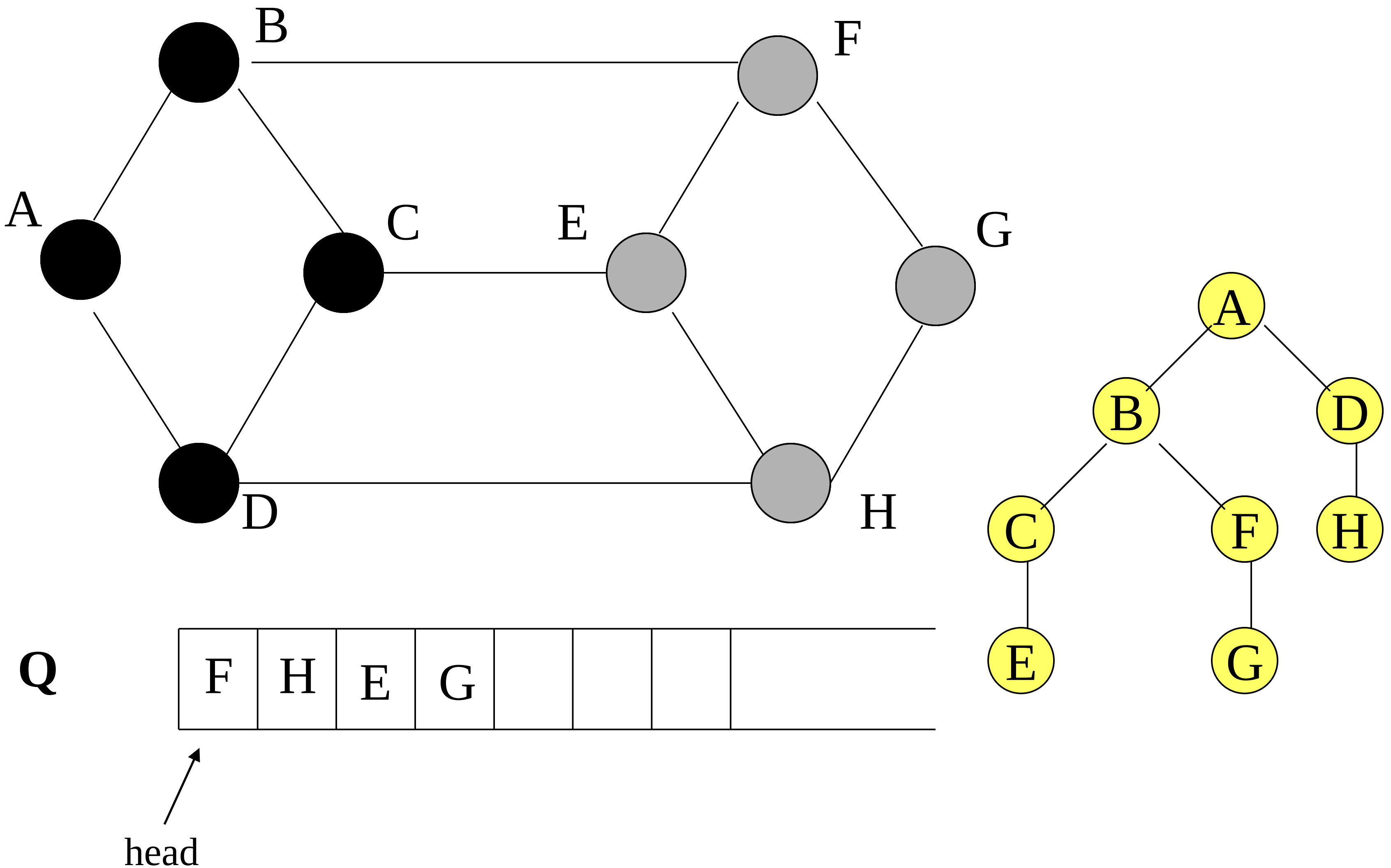
(da F. Damiani - Alg. & Sper. 06/07)

Esempio (4/9)



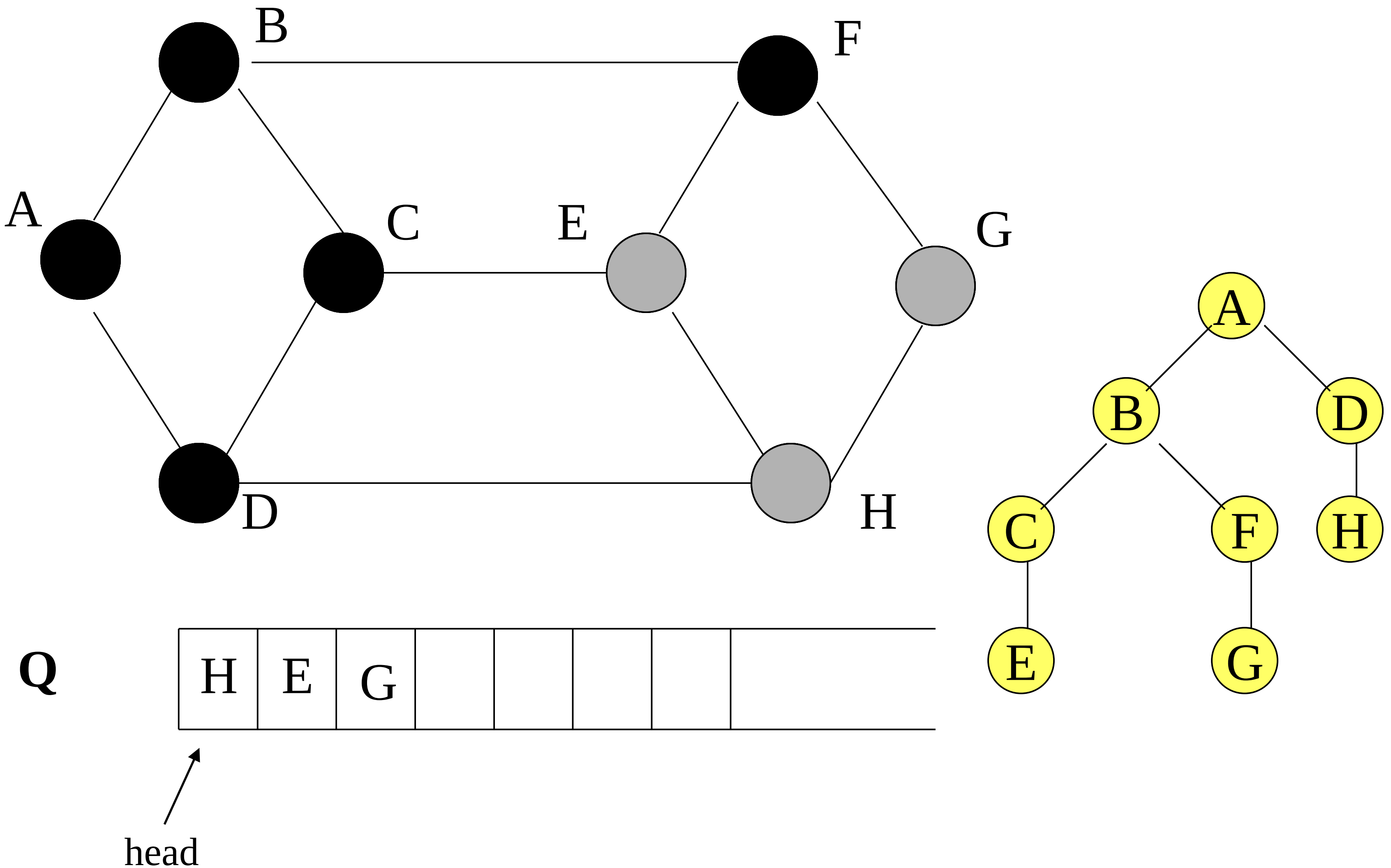
(da F. Damiani - Alg. & Sper. 06/07)

Esempio (5/9)



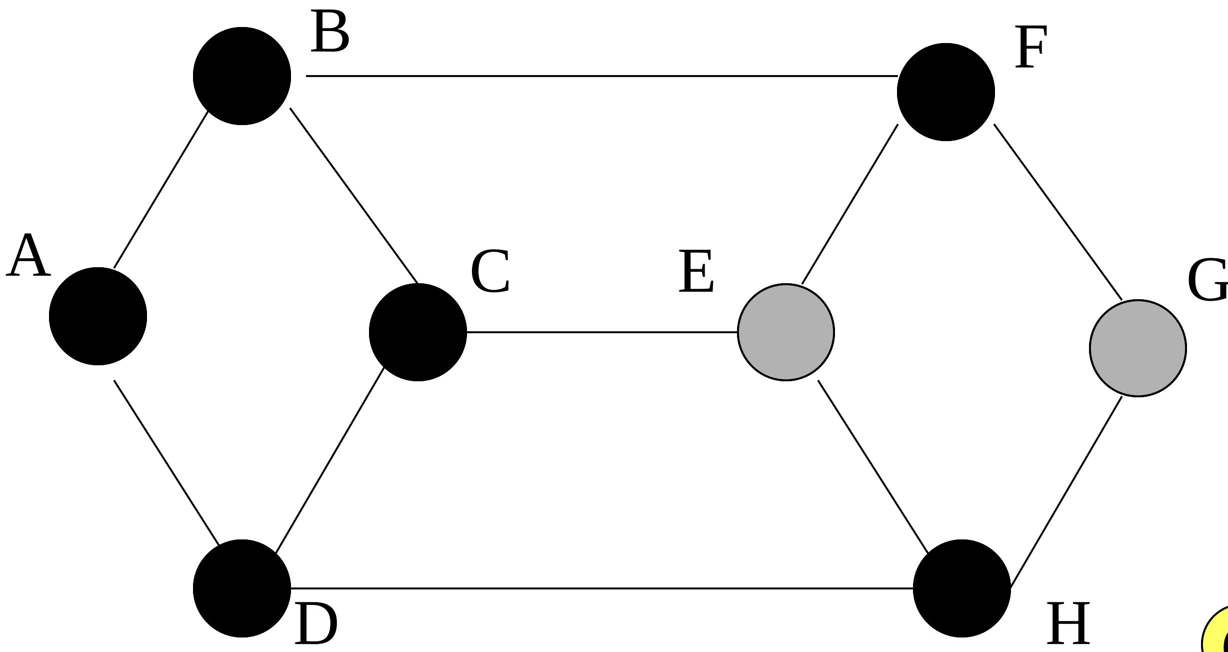
(da F. Damiani - Alg. & Sper. 06/07)

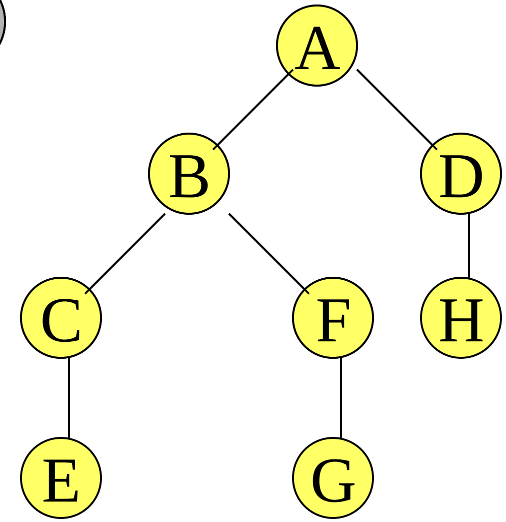
Esempio (6/9)

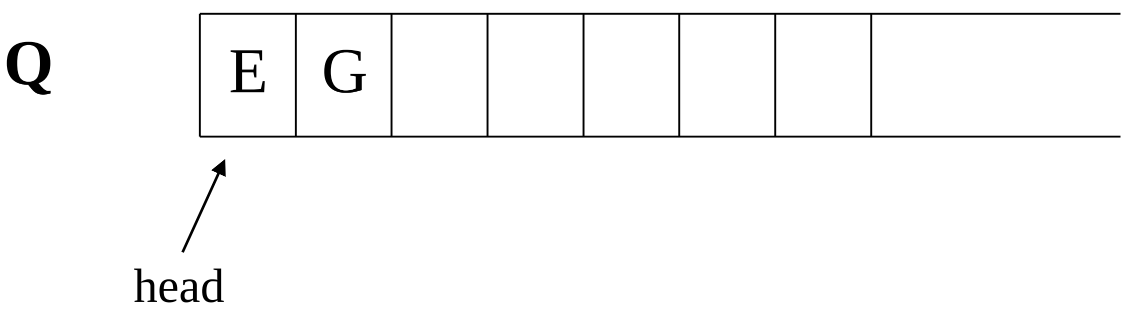


(da F. Damiani - Alg. & Sper. 06/07)

Esempio (7/9)

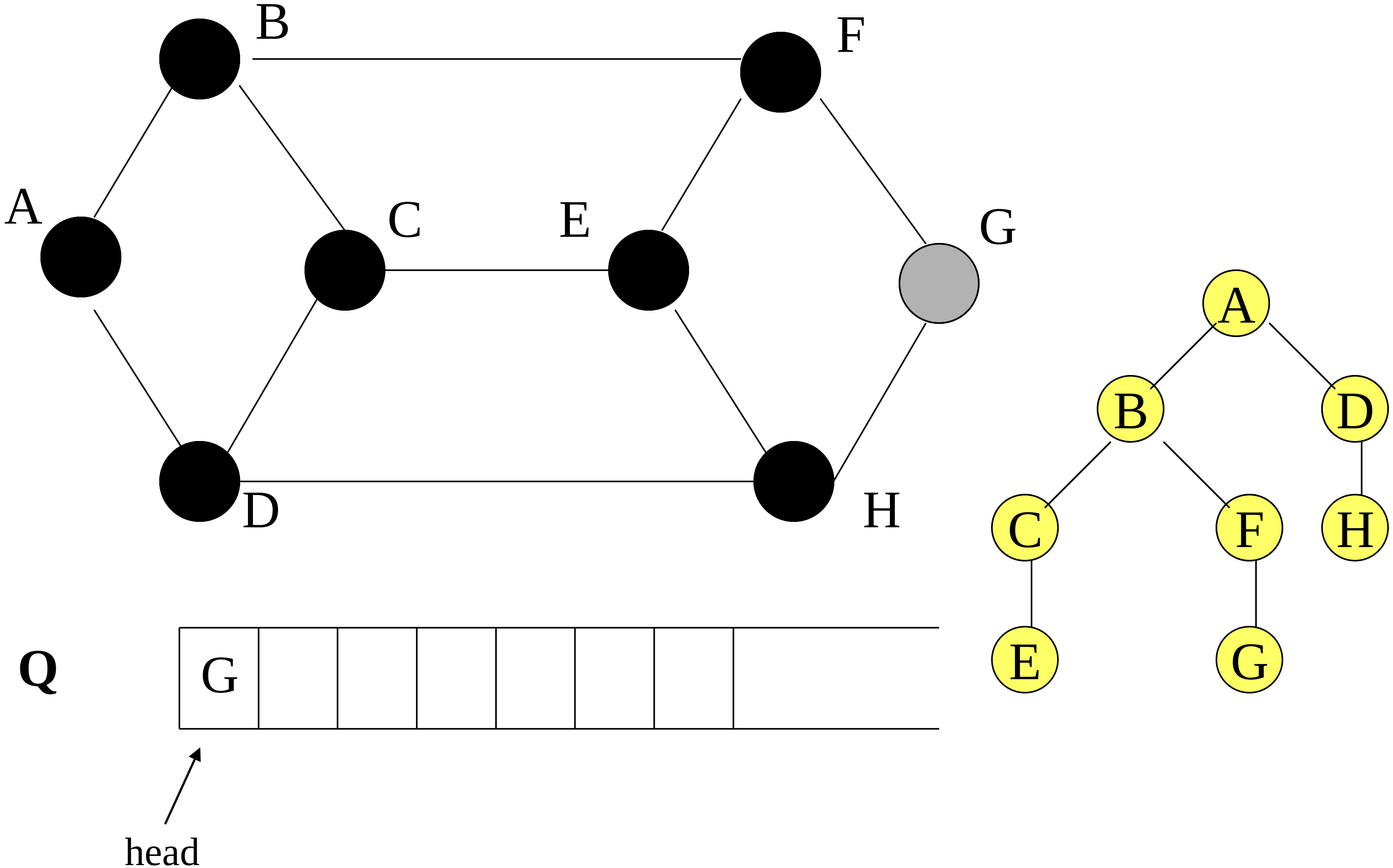






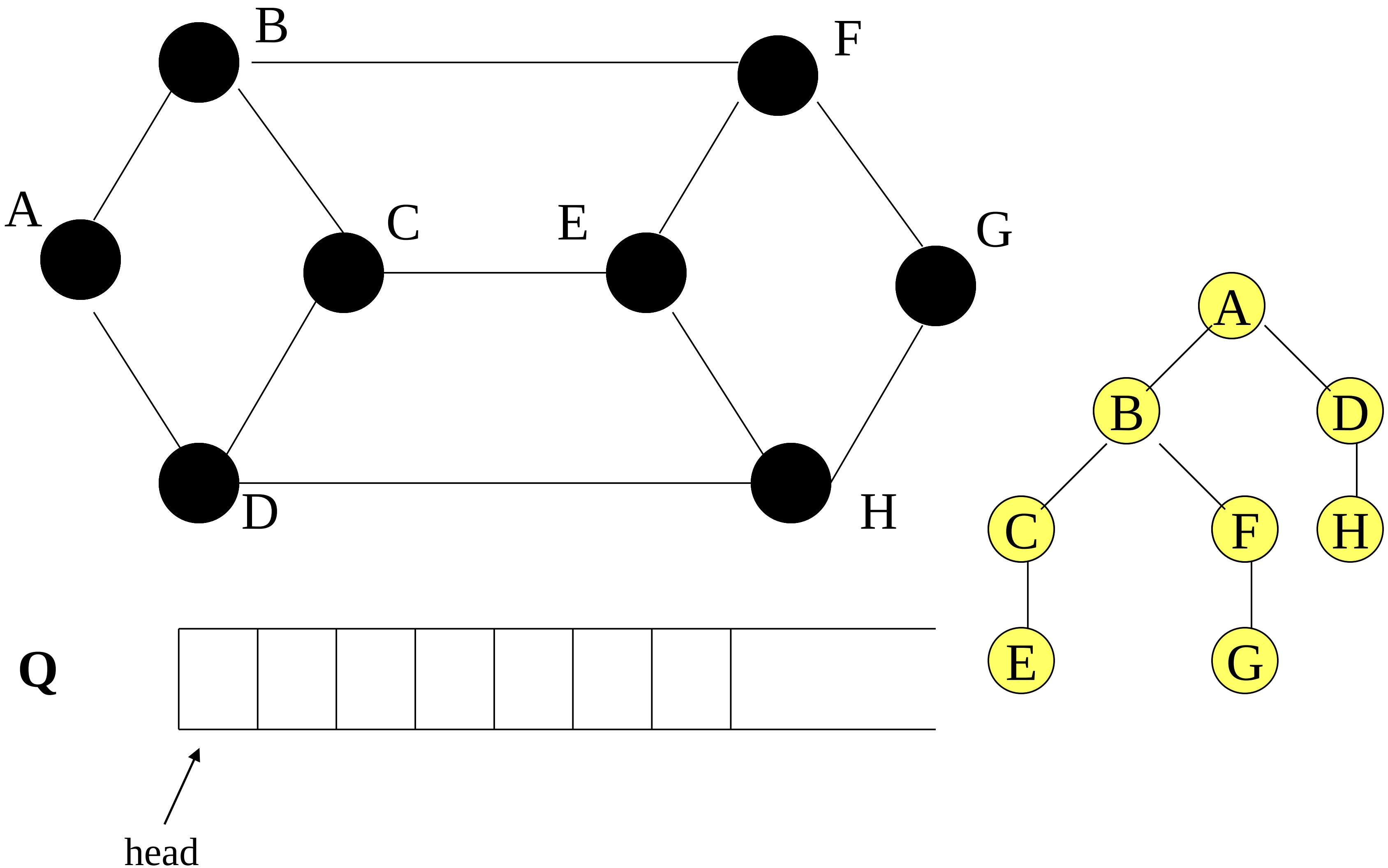
(da F. Damiani - Alg. & Sper. 06/07)

Esempio (8/9)



(da F. Damiani - Alg. & Sper. 06/07)

Esempio (9/9)



(da F. Damiani - Alg. & Sper. 06/07)

Visita BFS

Con D generico

```
VISITA-BFS (G, s)
   D <- Create()
   color [s] <- gray
   {visita s}
   Add(D,s)
   while NotEmpty(D) do begin
      u <- First(D)
      if esiste v bianco adj ad u then
         color [v] <- gray
         π[v] <- u
         {visita v}
         Add(D,v)
      else
         color [u] <- black
         RemoveFirst(D)
   end
```

Con D implementato con coda

```
VISITA-BFS (G, s)
   D <- empty_queue()
   color [s] <- gray
   {visita s}
   enqueue(D,s)
   while NotEmpty(D) do begin
      u <- head(D)
      if esiste v bianco adj ad u then
         color [v] <- gray
         π[v] <- u
         {visita v}
         enqueue(D,v)
      else
         color [u] <- black
         dequeue(D)
   end
```

Visita BFS – possiamo fare di meglio

Sappiamo già che u rimarrà in testa alla coda finché ci saranno vertici bianchi adiacenti ad esso

```
VISITA-BFS (G, s)
  D <- empty_queue()
  color [s] <- gray
 {visita s}
   enqueue(D,s)
   while NotEmpty(D) do begin
      u <- head(D)

      if esiste v bianco adj ad u then
         color [v] <- gray
         π[v] <- u
         {visita v}
         enqueue(D,v)
      else
         color [u] <- black
         dequeue(D)
   end
```

```
   enqueue(D,s)
   while NotEmpty(D) do begin
      u <- head(D)
      {visita u}
      for ogni v adj ad u then
         if color[v] = white
            color [v] <- gray
            π[v] <- u
            enqueue(D,v)
      end for
      color [u] <- black
      dequeue(D)
   end
```

Visita BFS – con gestione liste

```
VISITA-BFS (G, s)
   D <- empty_queue()
   color [s] <- gray
   enqueue(D,s)
   while NotEmpty(D) do begin
      u <- head(D)
      {visita u}

      for ogni v adj ad u then
         if color[v] = white
            color [v] <- gray
            π[v] <- u
            enqueue(D,v)
      end for

      color [u] <- black
      dequeue(D)
   end
```

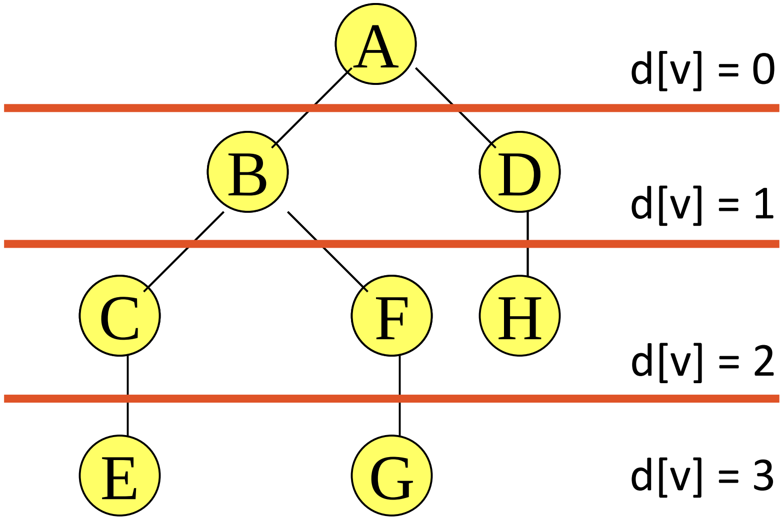
```
Ptr <- lista di adiacenti di u
while Ptr != NULL then
   if color[Ptr.vert] = white
      color [Ptr.vert] <- gray
      π[Ptr.vert] <- u
      enqueue(D, Ptr.vert)
   Ptr <- Ptr.next
end while
```

Ptr è (un puntatore ad un elemento di) una lista di vertici.

Attenzione: non è sintassi C

Caratteristica dell'albero BFS

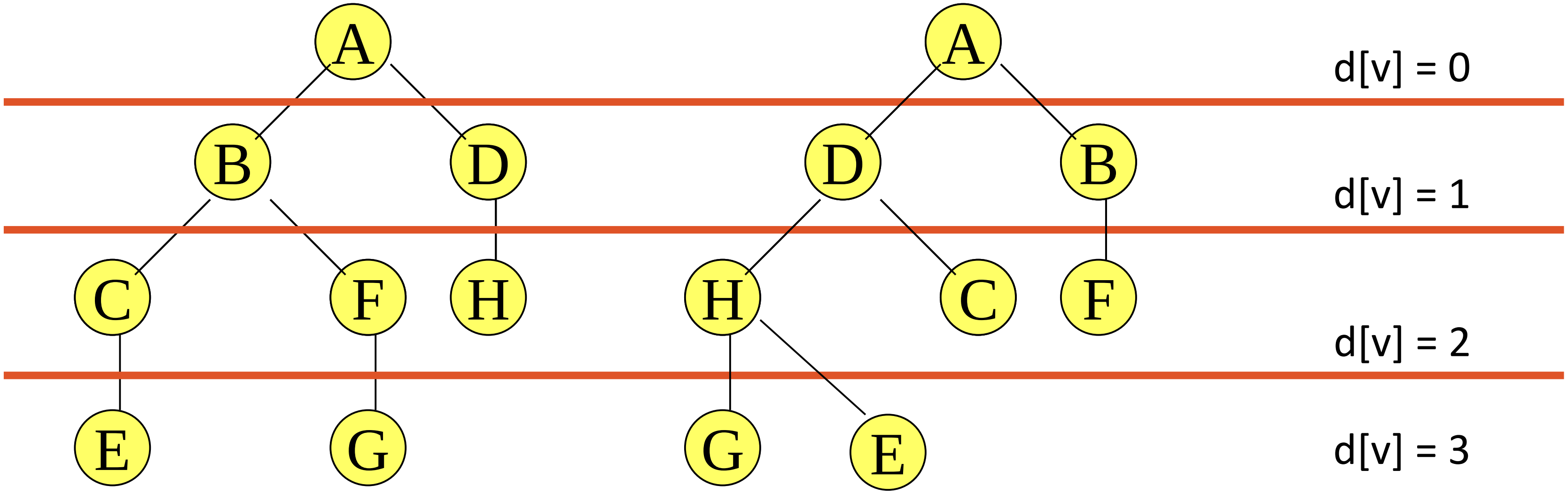
L'albero BFS viene costruito a livelli.



Vogliamo associare ad ogni nodo v un attributo d[v] (**distanza stimata** di v) che ricordi il suo livello nell'albero

Perché?

Nel seguito dimostreremo che al termine della visita BFS **d[v] = $\boldsymbol{\delta(s, v)}$**. Dove $\delta(s, v)$ è la distanza tra s e v. Infatti, anche con liste di adiacenza ad ordine invertito, i d[v] non cambiano.



Ricorda: la distanza tra s e v è la lunghezza del cammino minimo tra s e v

Implementazione con vettore dei livelli

```
VISITA-BFS (G, s)
   D <- empty_queue()
   color [s] <- gray
   d[s] <- 0
   enqueue(D,s)
   while NotEmpty(D) do begin
      u <- head(D)
      {visita u}
      for ogni v adj ad u then
         if color[v] = white
            color [v] <- gray
            π[v] <- u
            d[v] <- d[u]+1
            enqueue(D,v)
      end for
      color [u] <- black
      dequeue(D)
   end
```

```
INIZIALIZZA (G)
      …
      for ogni vertice u ∈ V[G] do
            color [u] <- white
            π[u] <- NULL
            d[u] <- +∞
```

Proprietà

PROPRIETA' 1: In D ci sono tutti e soli i vertici grigi

PROPRIETA' 2: Se $\langle v_1, v_2, \dots, v_n \rangle$ è il contenuto di D

allora:

(i) $\mathbf{d[v_i] \leq d[v_{i+1}]}$ **per 1 ≤ i ≤ n-1**

(ii) $\mathbf{d[v_n] \leq d[v_1] + 1}$

Cioè i vertici sono ordinati per livelli nella coda

Cioè la coda contiene sempre al massimo 2 livelli

Dimostrazione per induzione:

Caso base: in D c'è solo s. P2 è banalmente vera.

Passo: nuova operazione su D può essere dequeue(D) o enqueue(D,v):

dequeue(D): o D rimane vuota (e P2 è banalmente vera) o rimangono $\langle v_2, \dots, v_n \rangle$ ma
(i) le disuguaglianze sono ancora vere e
quindi anche (ii) $\mathbf{d[v_n] \leq}$ $d[v_1] + 1 \leq$ $\mathbf{d[v_2] + 1}$

enqueue(D,v): v viene reso figlio di v_1 e messo in coda a D. Quindi $d[v] = d[v_1] + 1$. Quindi

(i) $\mathbf{d[v_n]} \leq d[v_n] + 1 = \mathbf{d[v]}$ e
(ii) $\mathbf{d[v]} = d[v_n] + 1 \leq \mathbf{d[v_1] + 1}$

Proprietà

Lemma (INV4)

La seguente proprietà è un invariante dei due cicli dell'algoritmo BFS-VISITA:

(INV4) $d[v] = \delta(s,v)$ per tutti i vertici v grigi o neri.

Dimostrazione

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

Dimostrazione **d[v] = δ(s,v)**

Sia v un vertice di G. Procediamo dimostrando le due implicazioni.

--- Dimostriamo d[v] ≥ δ(s,v)

Dato che l'albero dei predecessori π contiene solo archi appartenenti a G, il cammino da s a v nell'albero è un cammino che appartiene anche a G.

Quindi **d[v] ≥ δ(s,v)**

(la lunghezza del cammino da s a v nell'albero è maggiore o uguale alla distanza tra s e v)

--- Dimostriamo d[v] ≤ δ(s,v)

Definiamo l'insieme dei vertici a distanza k da s nel grafo.

$$\mathrm{V_k} = \{\mathrm{v} \in \mathrm{V} | \boldsymbol{\delta}(\mathbf{s}, \mathbf{v}) = \mathbf{k}\}$$

Dimostriamo che, per qualsiasi $\mathrm{v} \in \mathrm{V_k}$, quando v è inserito nella coda (cioè quando diventa grigio) **d[v] ≤ k**
(questo basta per dimostrare d[v] ≤ δ(s,v)).

Dimostrazione **d[v] = δ(s,v)** - II

Procediamo per **induzione su k**.

BASE: $\mathbf{k = 0}$

L'unico vertice a distanza 0 da s è s stesso e **d[s]** = 0 **≤ k**

(uno dei primi passi dell'algoritmo è d[s] <- 0).

PASSO INDUTTIVO: $\forall \mathbf{w} \in \mathbf{V_{k-1}}\ \mathbf{d[w] \leq k-1} \Rightarrow \forall \mathbf{v} \in \mathbf{V_k}\ \mathbf{d[v] \leq k}$

Sia $\mathrm{v} \in \mathrm{V_k}$. Allora $\delta(\mathrm{s}, \mathrm{v}) = \mathrm{k}$ (per definizione di $\mathrm{V_k}$).

Con $\mathrm{k} > 0$, esisterà almeno un vertice w tale che $\delta(\mathrm{s}, \mathrm{w}) = \mathrm{k} - 1$ e $(\mathrm{w}, \mathrm{v}) \in \mathrm{E}$ (cioè con un arco che va da w a v).

Definiamo quindi $\mathrm{U_{k-1}} = \{\mathrm{w} \in \mathrm{V_{k-1}} | (\mathrm{w}, \mathrm{v}) \in \mathrm{E}\}$ (l'insieme dei vertici appartenenti a $\mathrm{V_{k-1}}$ con un arco entrante in v)

Tra questi, sia u il primo vertice di $\mathrm{U_{k-1}}$ad essere scoperto e inserito nella coda

Dimostrazione **d[v] = δ(s,v)** - III

Per la politica **FIFO**, u sarà anche il primo ad essere estratto dalla coda

Poiché u appartiene all'insieme ($\mathrm{U}_{\mathrm{k}-1}$) dei vertici che hanno un nodo entrante in v (e che hanno distanza da s uguale a k - 1), ed è il primo ad essere estratto dalla coda, quando u viene estratto v non è ancora stato scoperto (è bianco), e **v verrà inserito nell'albero come figlio di u** (cioè π[v] = u) con $\mathbf{d[v]} = \mathbf{d[u]} + \mathbf{1}$.

Inoltre, per l'ipotesi induttiva, $\mathbf{d[u]} \leq \mathbf{k} - \mathbf{1}$

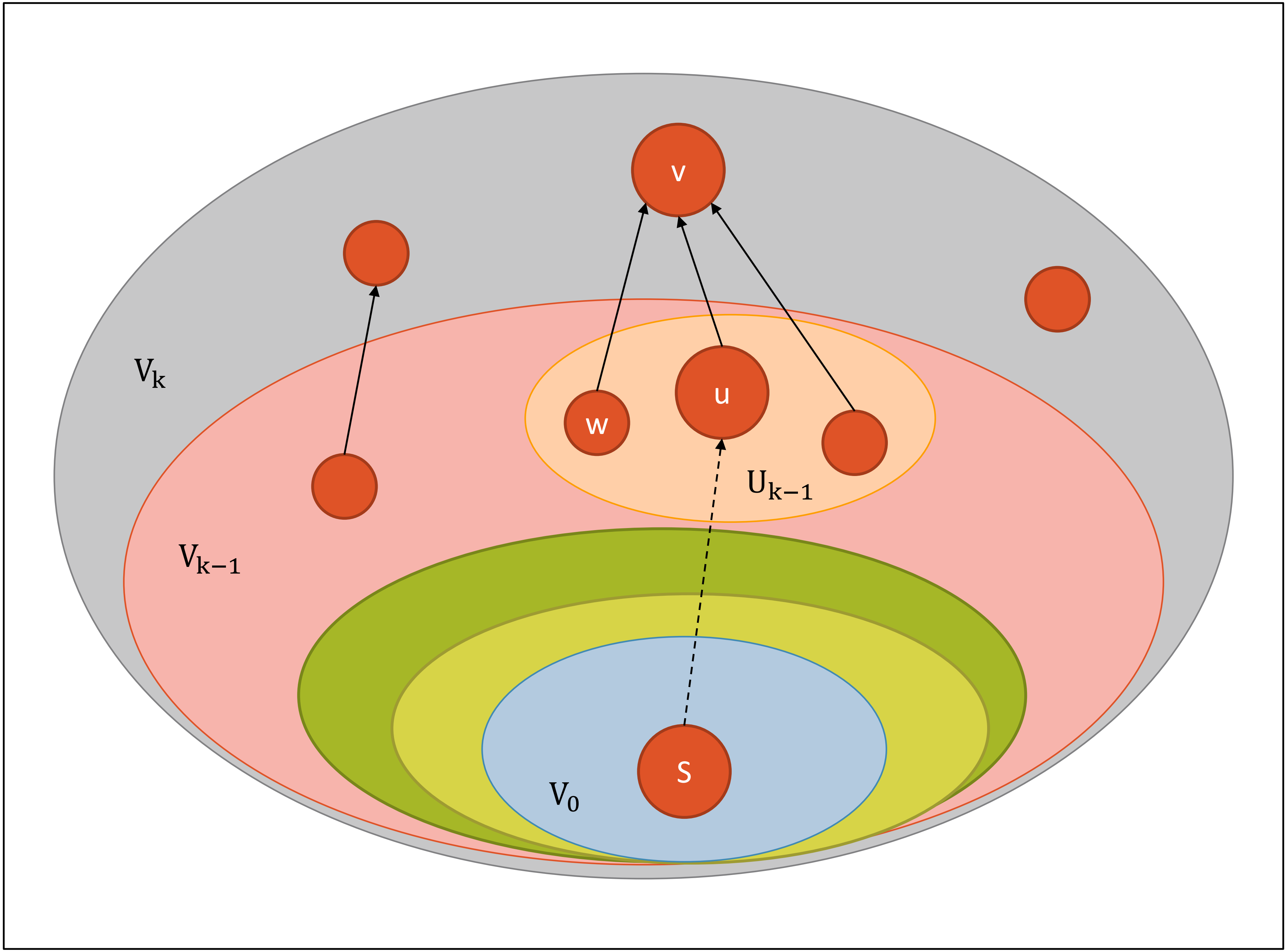
Quindi, quando inseriremo v nell'albero,

$\mathbf{d[v]} = \mathrm{d[u]} + 1$

$\mathbf{d[v]} \leq (\mathrm{k} - 1) + 1$ (abbiamo sostituito = d[u] con ≤ k-1)

$\mathbf{d[v]} \leq \mathbf{k}$ (banale semplificazione)

CVD.


v
w
u
S
V_k
V_{k-1}
U_{k-1}
V_0

Teorema 3 (distanza nell'albero BFS)

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

Dimostrazione.

Se v non è raggiungibile 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.

Cosa devo aver capito fino ad ora

- Come funziona una visita in ampiezza BFS
- Come essa si istanzia a partire dalla visita generica
- L’albero BFS ottenuto da una visita in ampiezza e le sue proprietà
- Dimostrazioni delle proprietà dell’albero BFS

…se non ho capito qualcosa

- Alzo la mano e chiedo
- Ripasso sul libro
- Chiedo aiuto sul forum
- Chiedo o mando una mail al docente