GRAFI: VISITA IN PROFONDITÀ

[Deme, seconda edizione] cap. 12

Sezione 12.3.2

[Cormen] cap. 22

Sezione 22.3

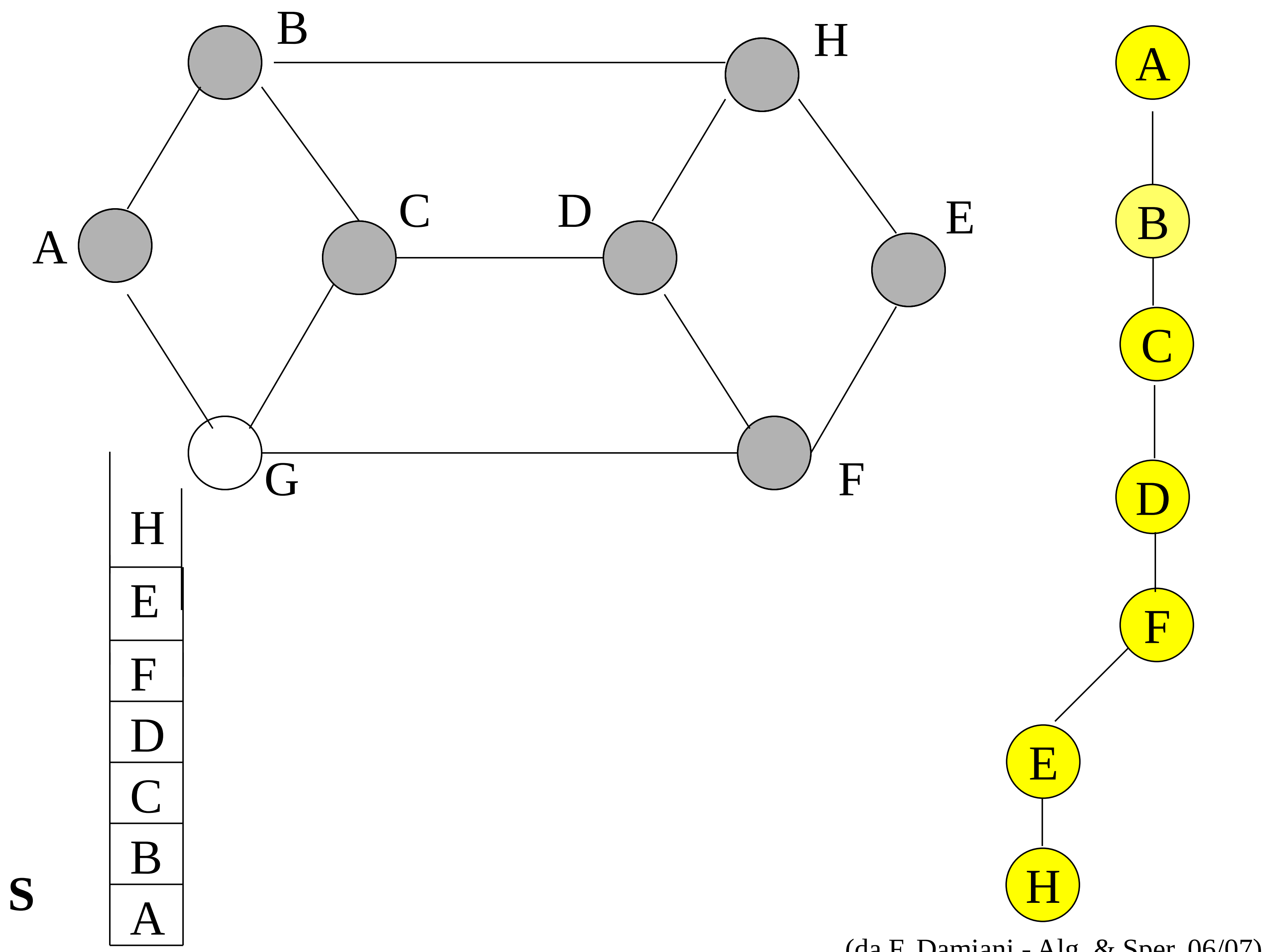
Visita in profondità

La visita in **profondità** (**DFS = deep first search**) esamina i vertici del grafo partendo dall'ultimo vertice incontrato.

VisitaDFS si ottiene dall'algoritmo generico VISITA implementando la struttura dati D con una **pila (o stack)**.

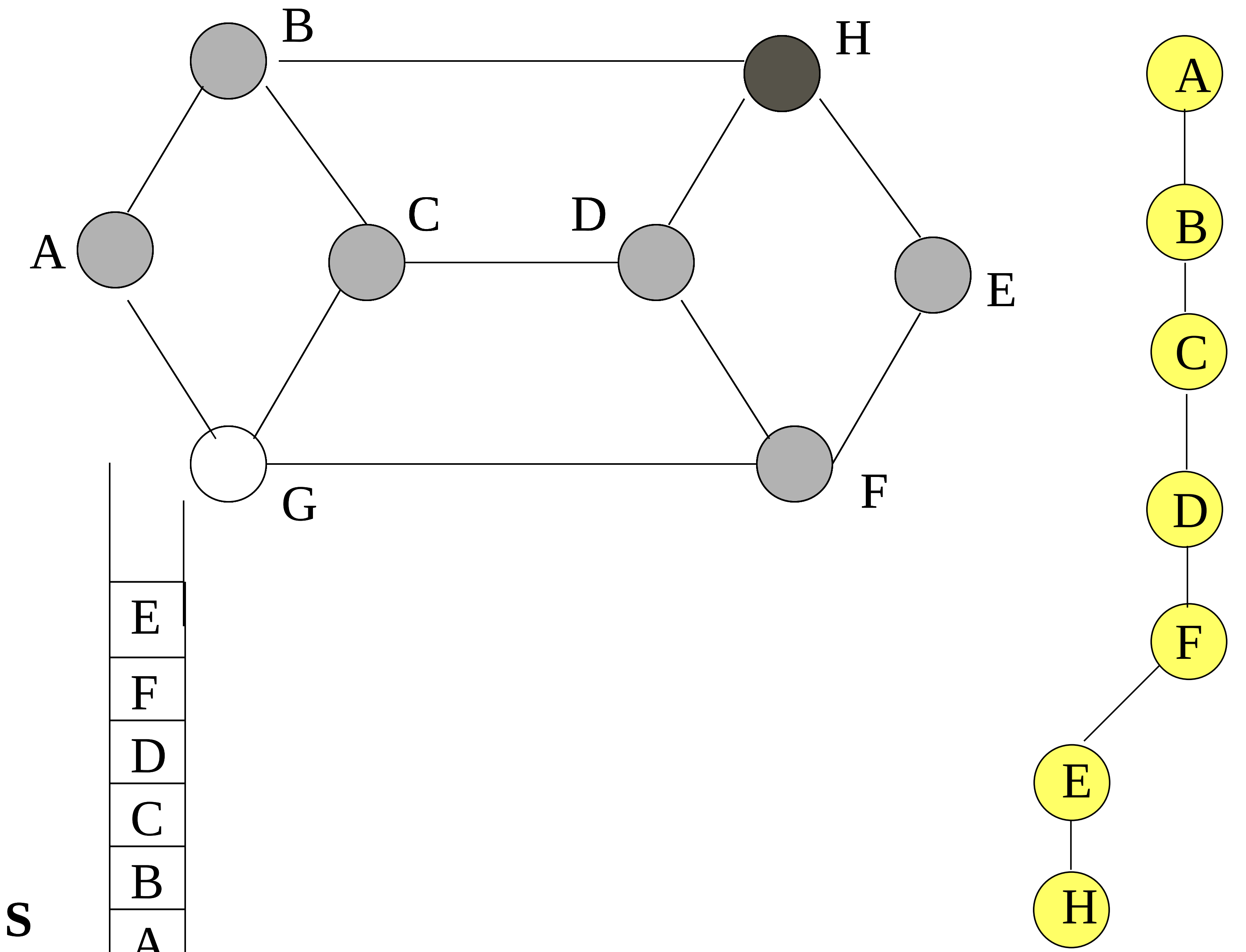
La pila ha una politica di tipo **LIFO** (Last In First Out) -> ciò vuol dire che sarà sempre il nodo **da meno tempo nella pila** ad essere esaminato per primo.

Esempio



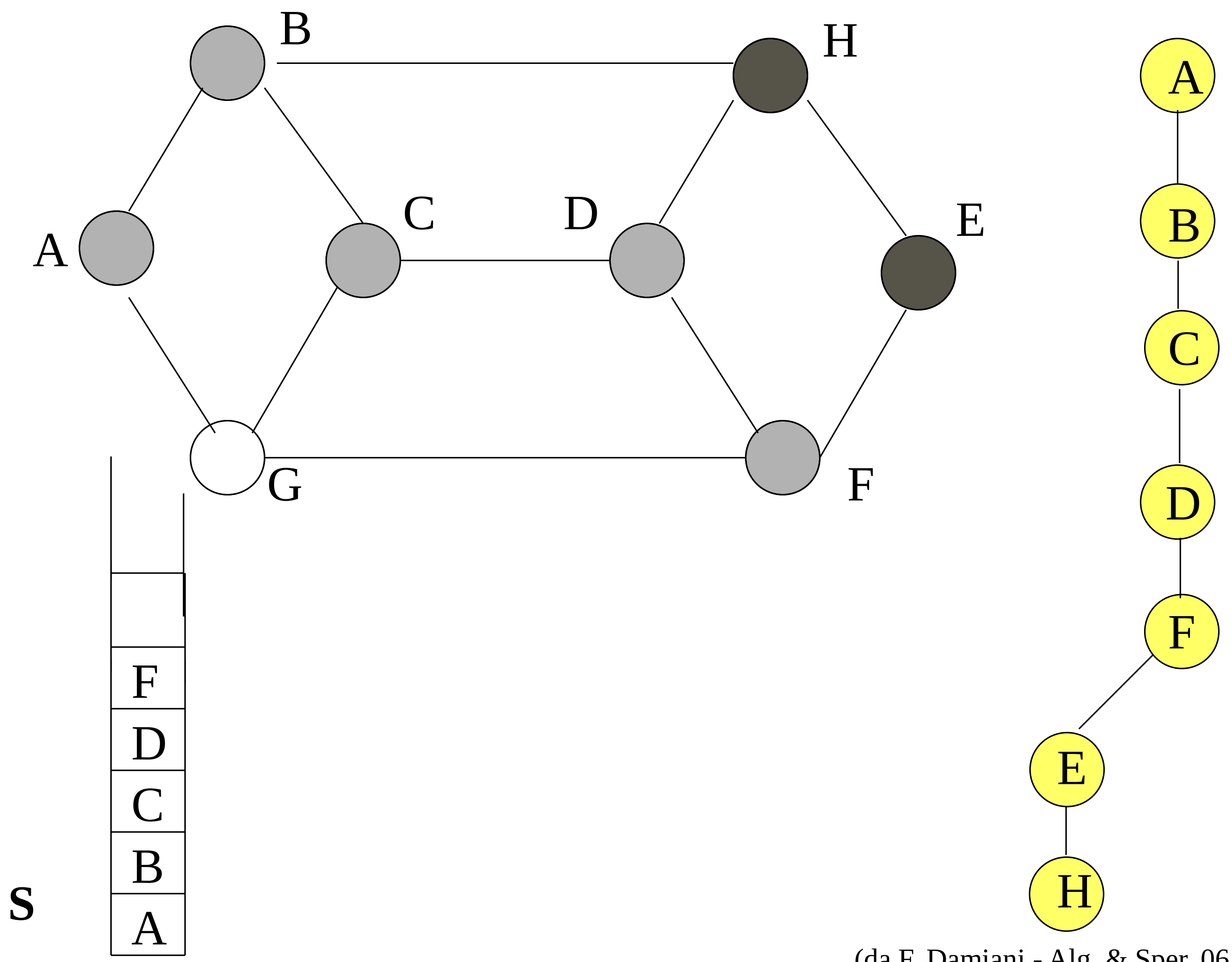
(da F. Damiani - Alg. & Sper. 06/07)

Esempio



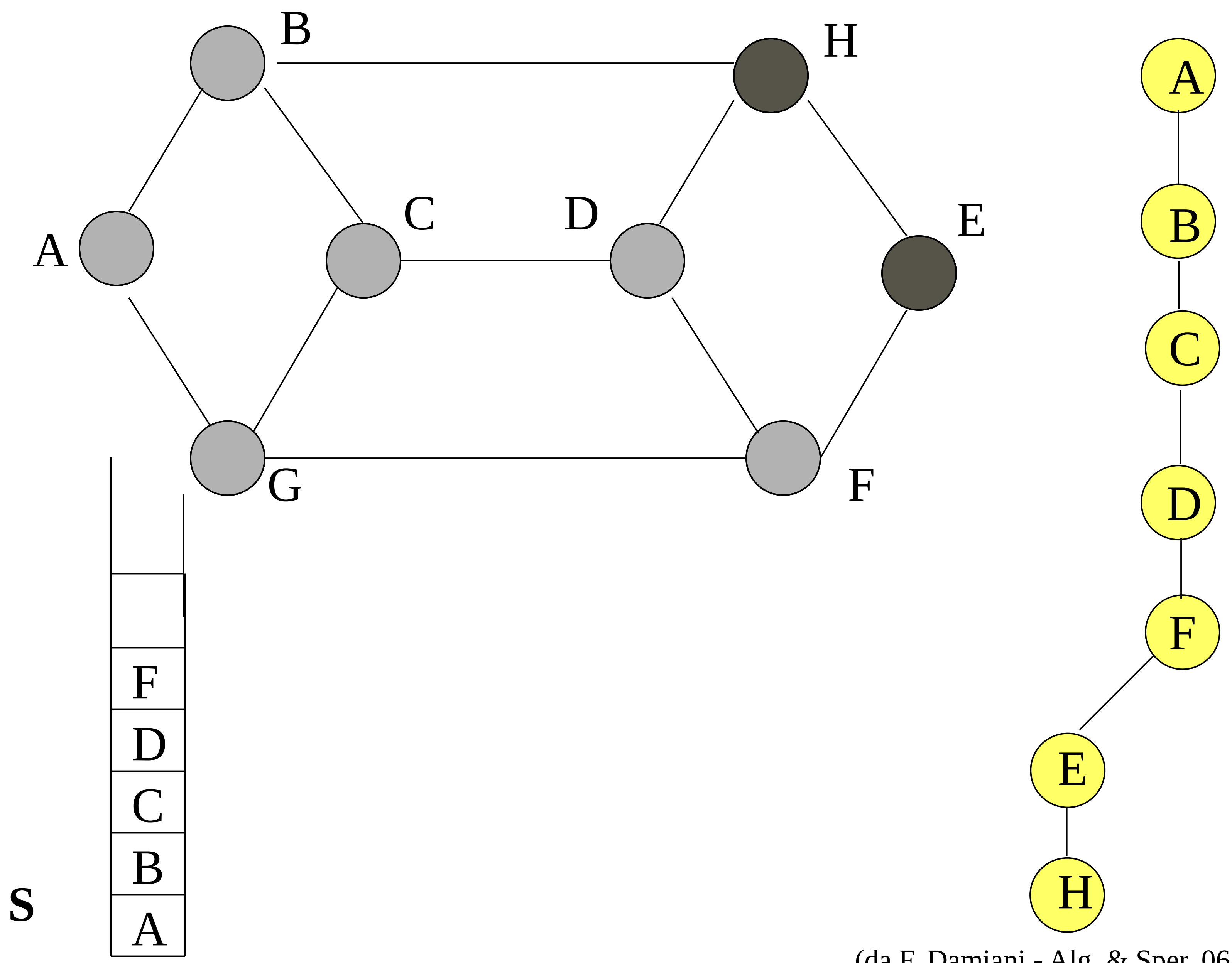
(da F. Damiani - Alg. & Sper. 06/07)

Esempio



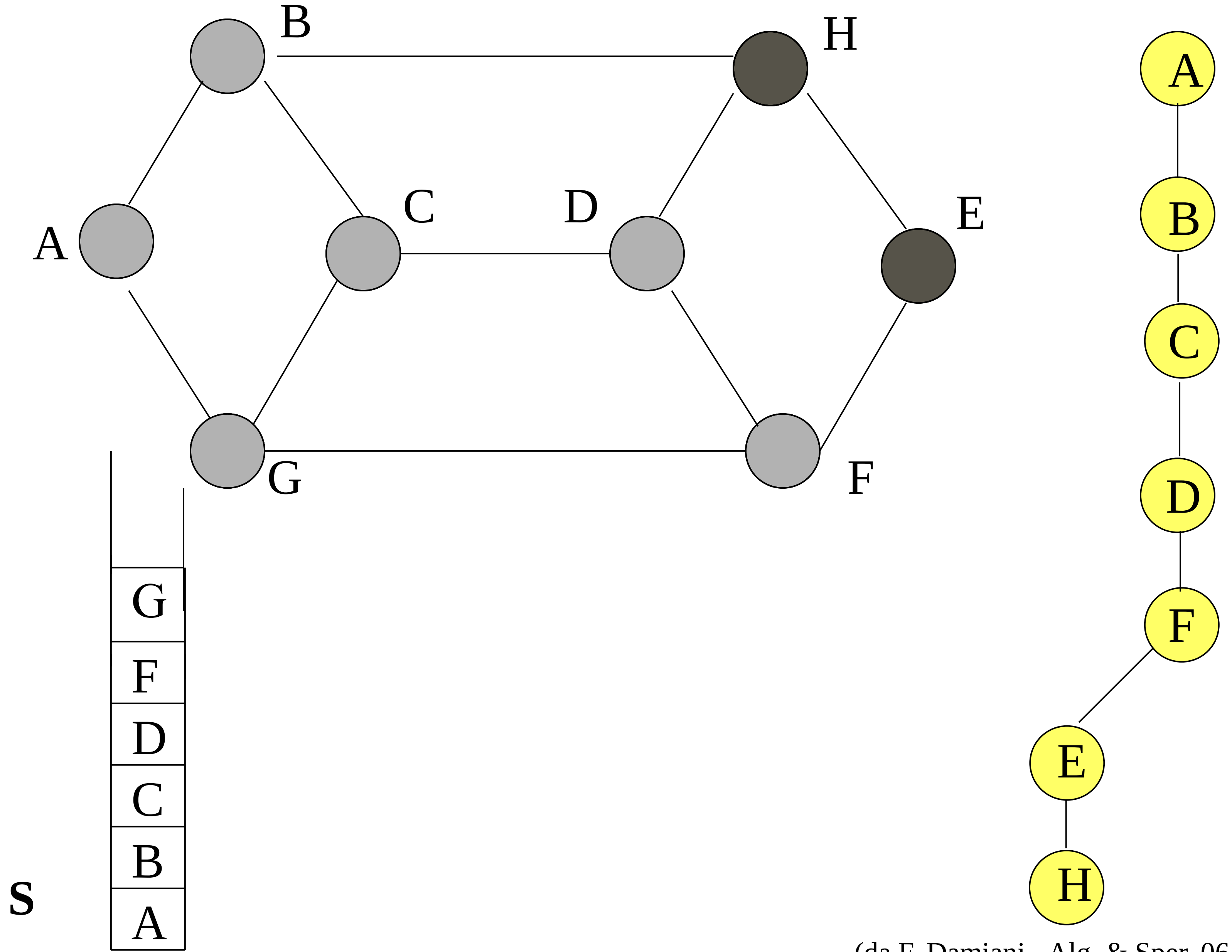
(da F. Damiani - Alg. & Sper. 06/07)

Esempio



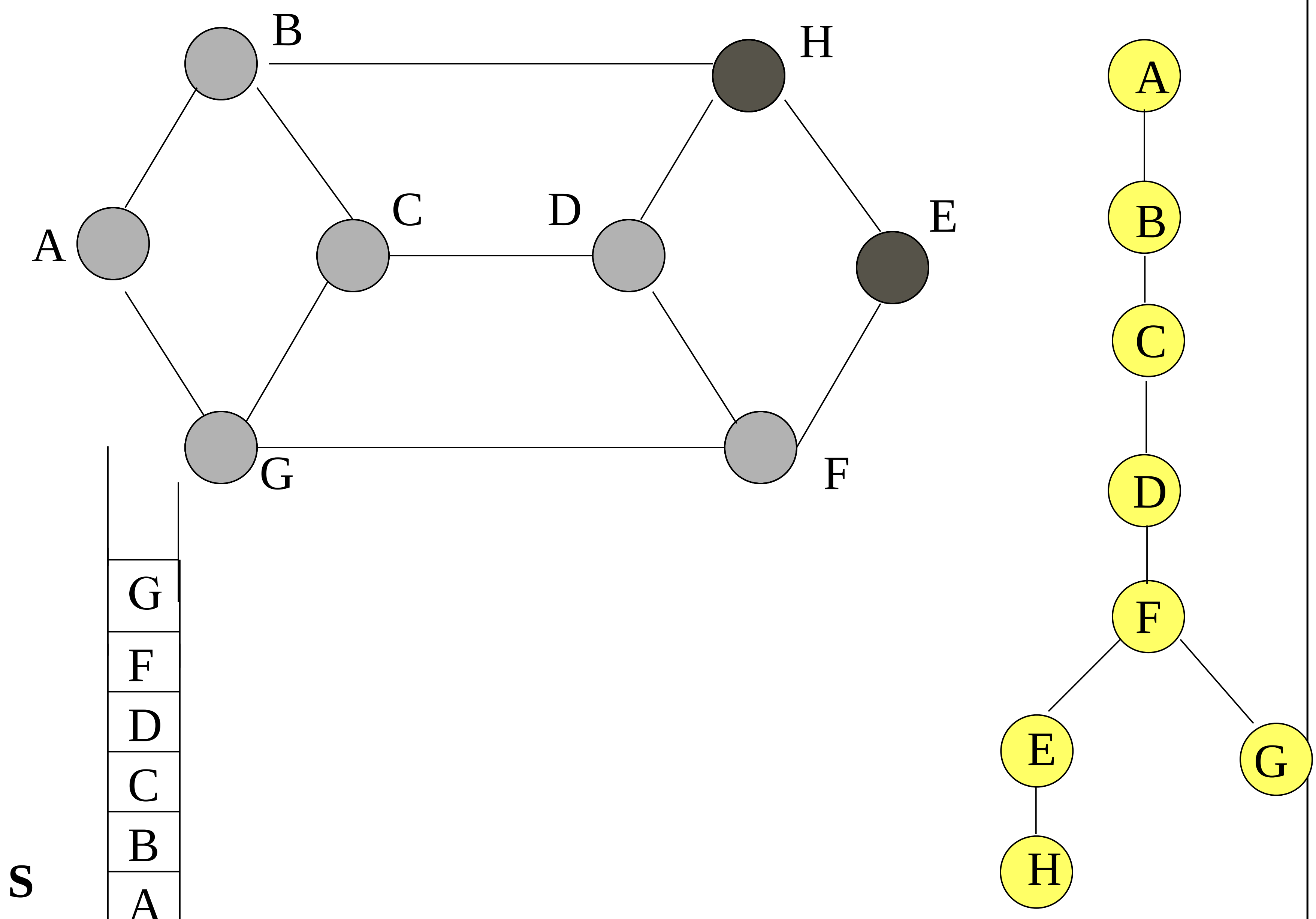
(da F. Damiani - Alg. & Sper. 06/07)

Esempio



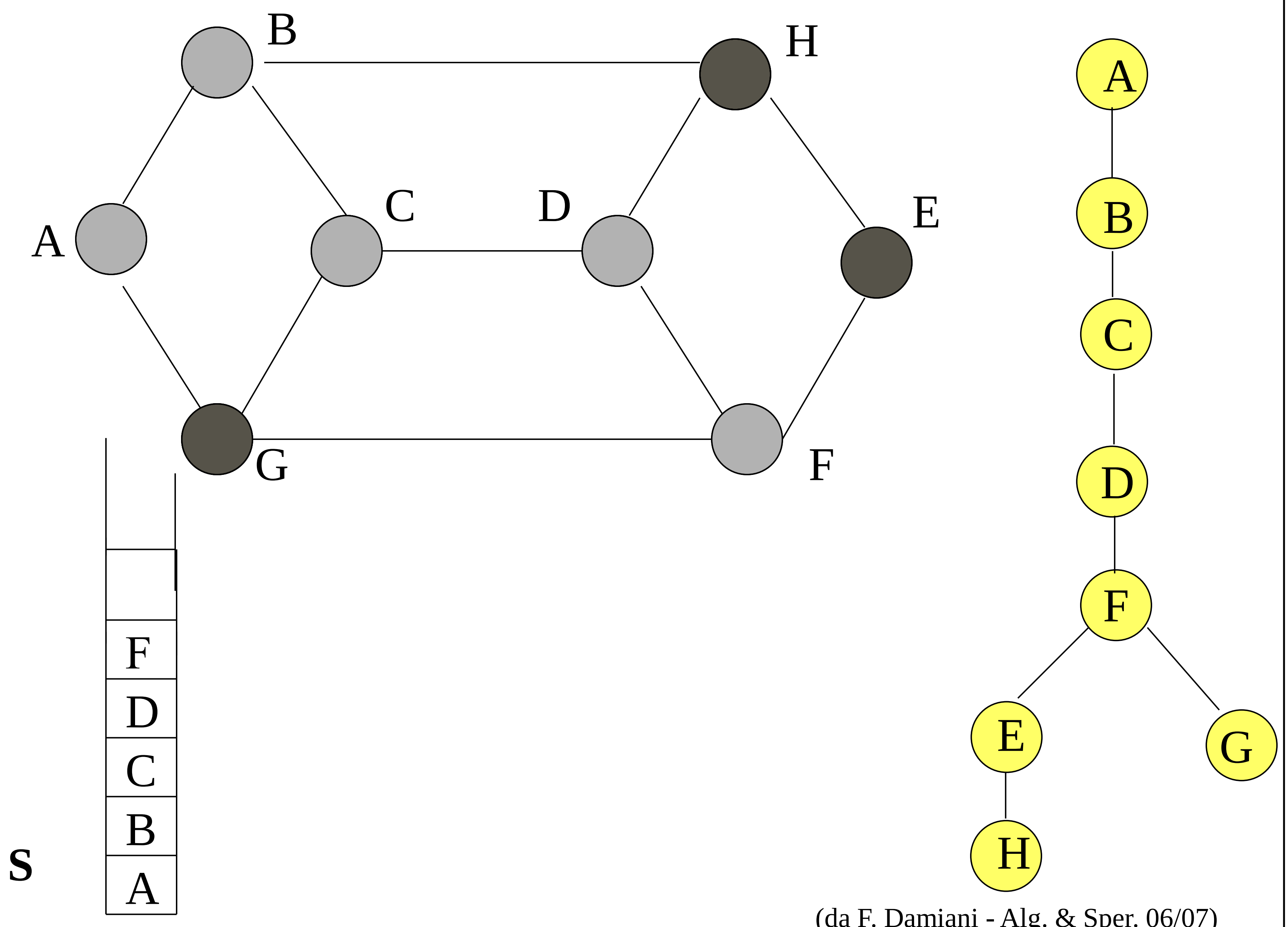
(da F. Damiani - Alg. & Sper. 06/07)

Esempio



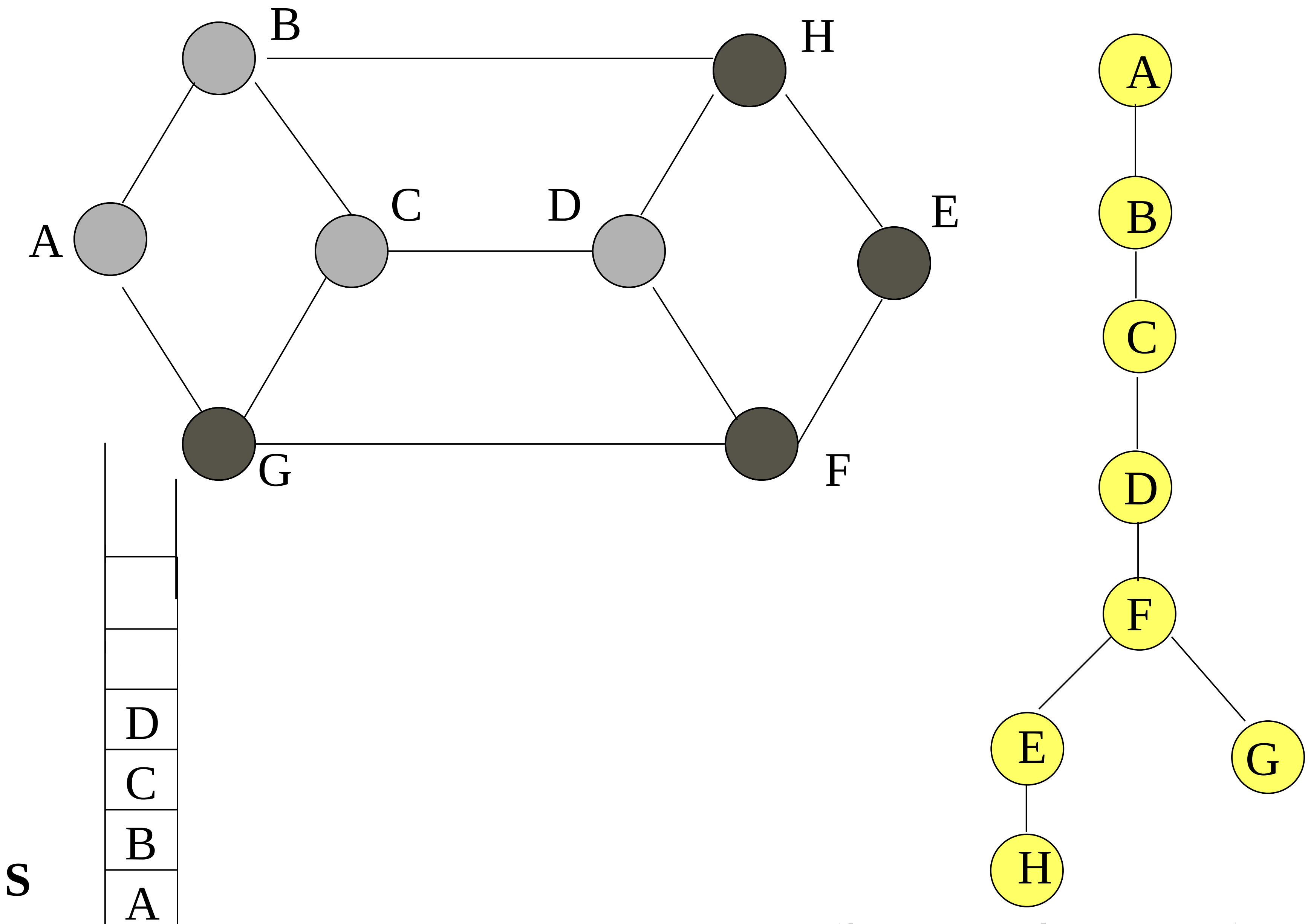
(da F. Damiani - Alg. & Sper. 06/07)

Esempio



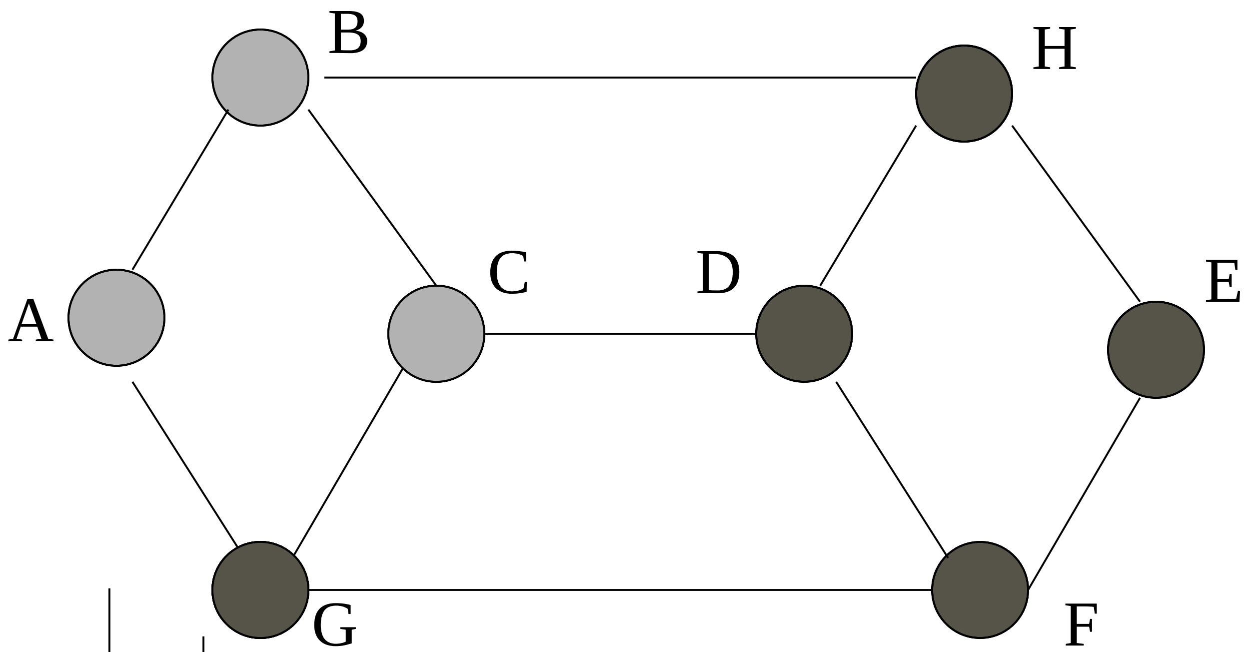
(da F. Damiani - Alg. & Sper. 06/07)

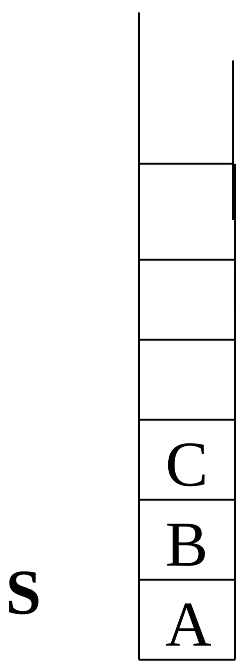
Esempio

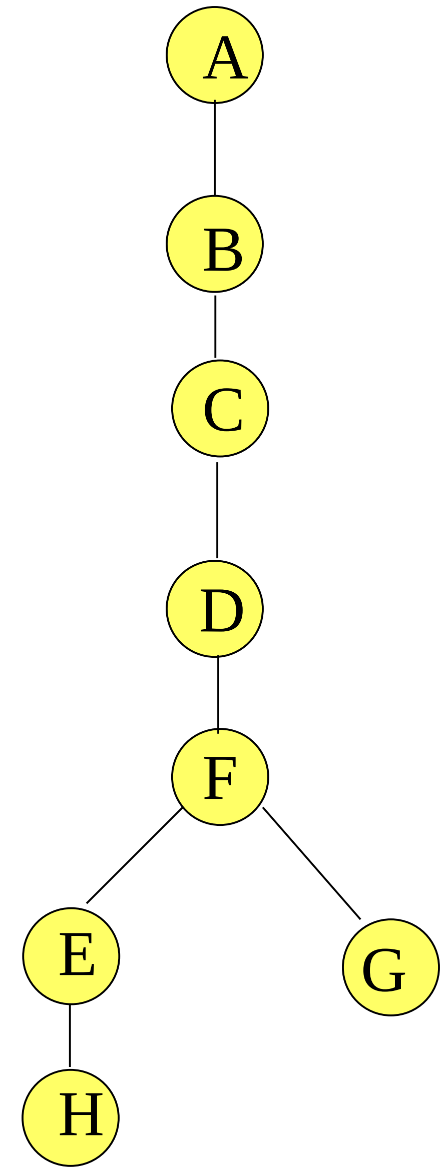


(da F. Damiani - Alg. & Sper. 06/07)

Esempio

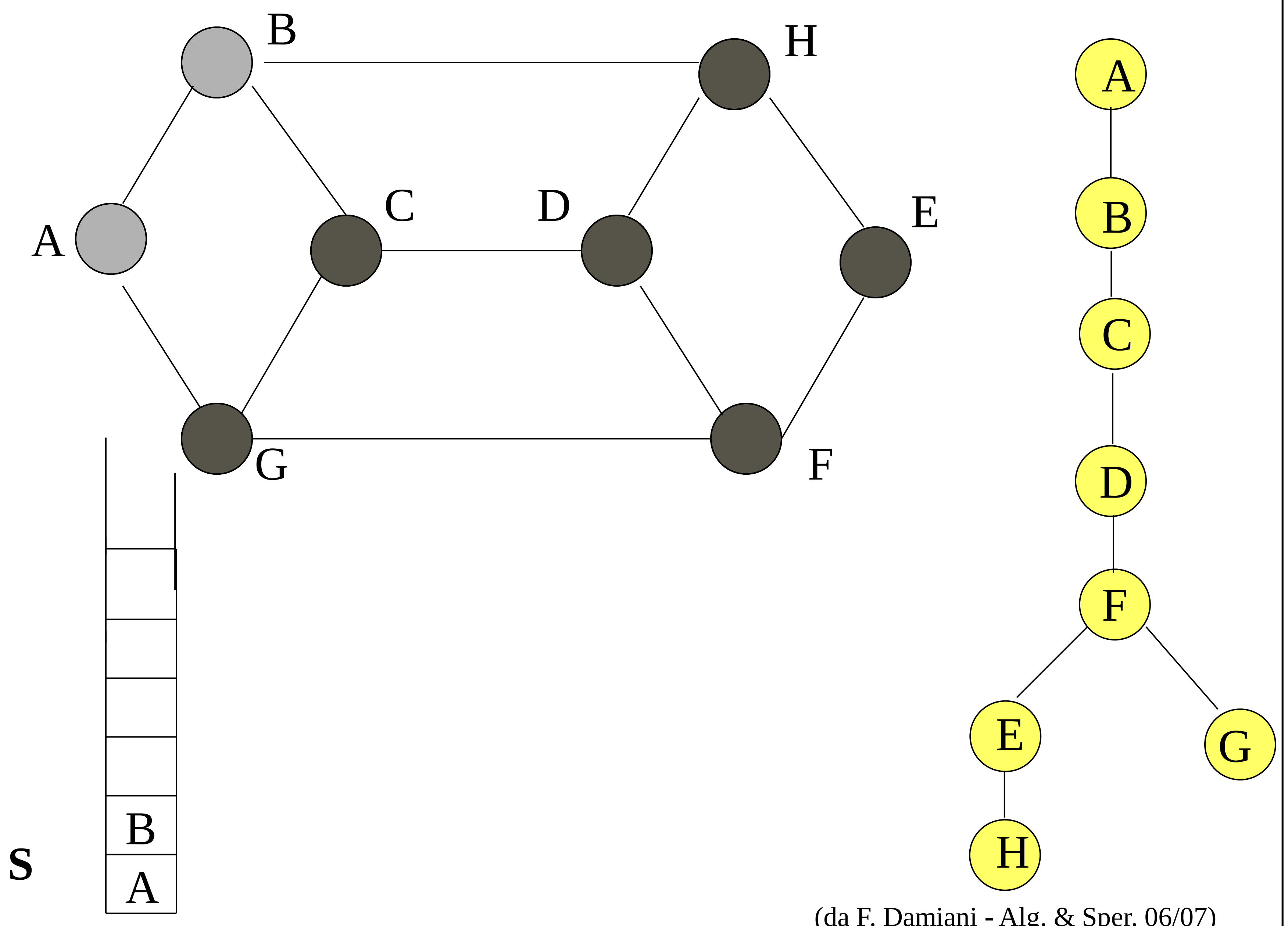






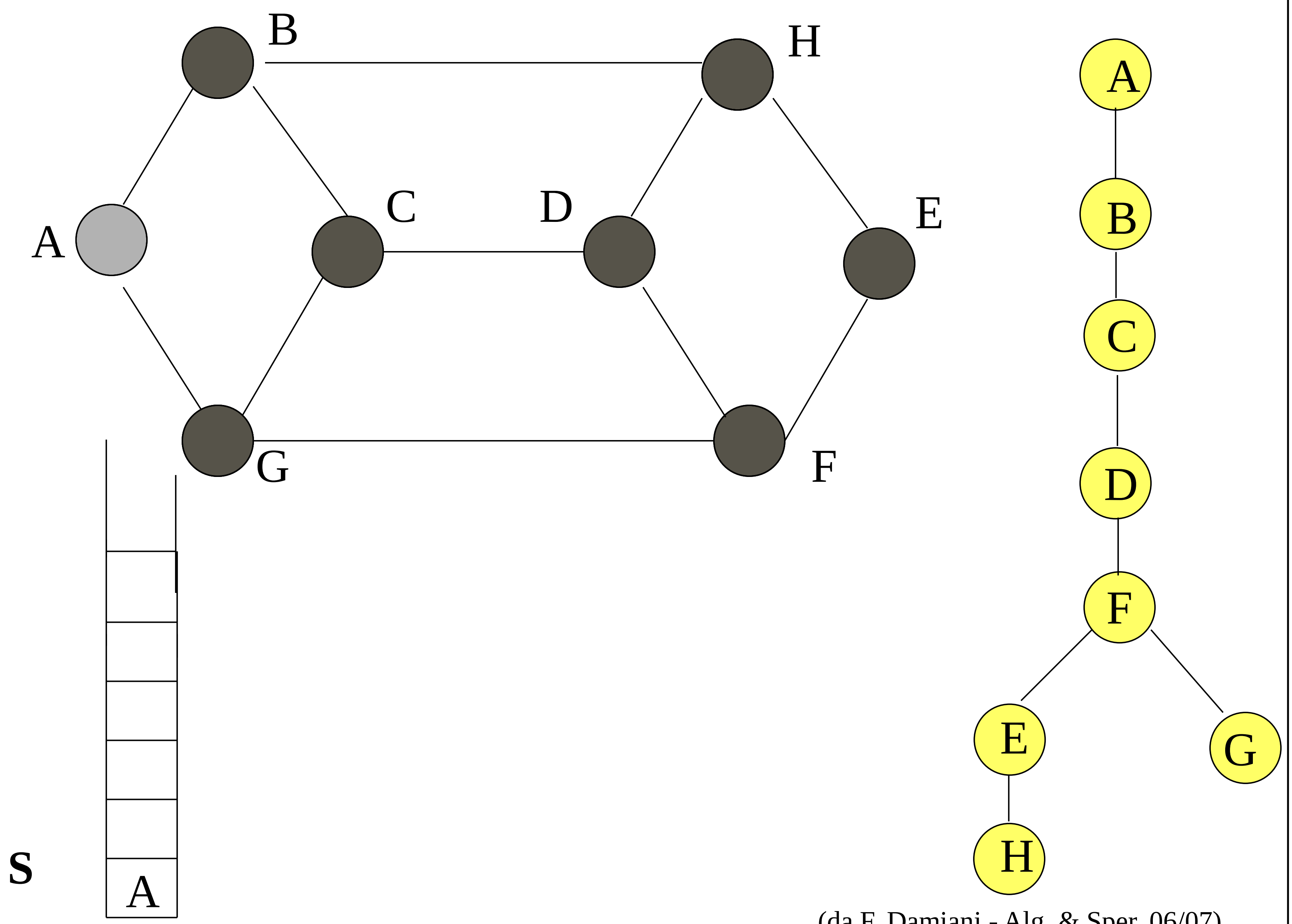
(da F. Damiani - Alg. & Sper. 06/07)

Esempio

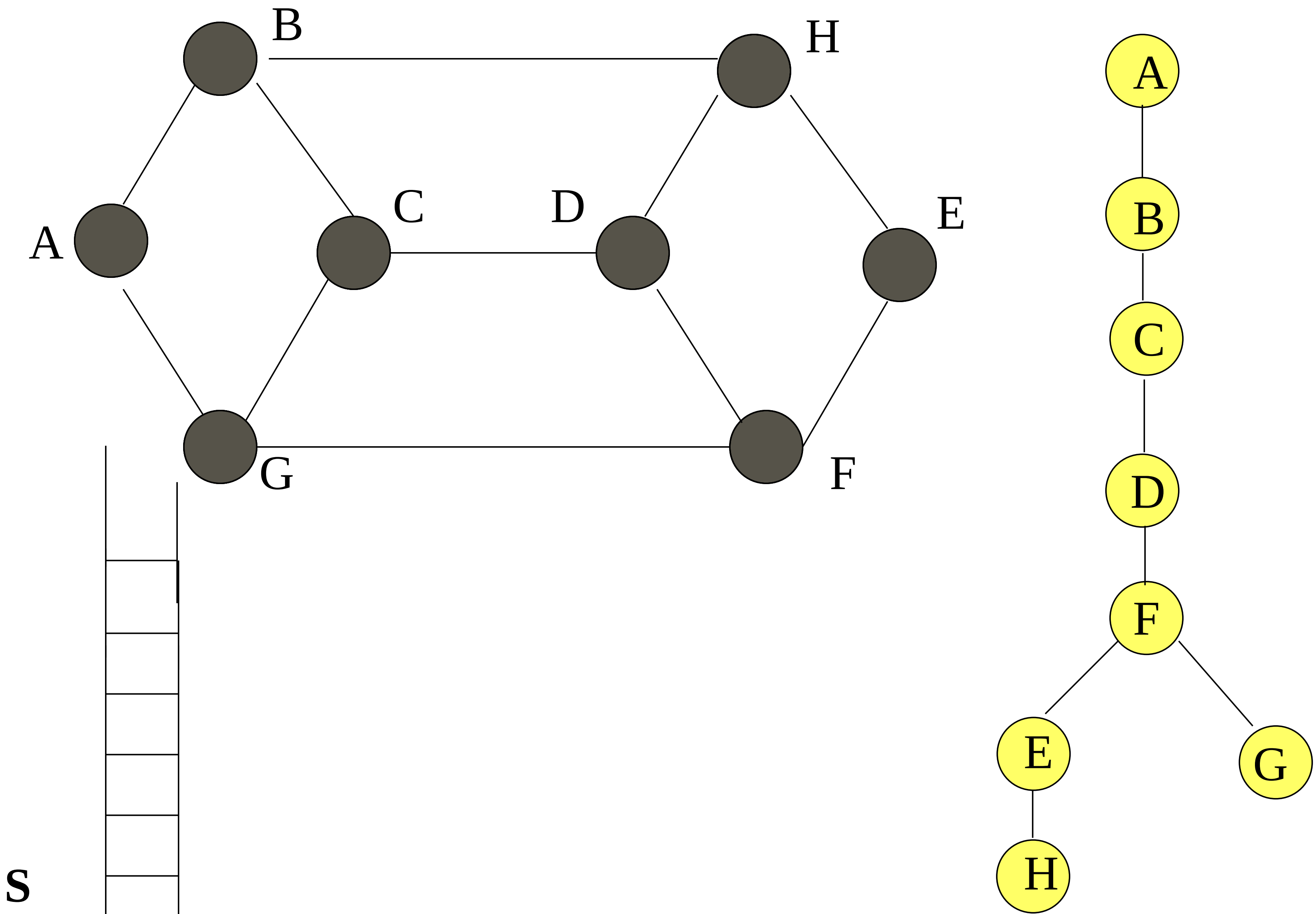


(da F. Damiani - Alg. & Sper. 06/07)

Esempio



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



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

Visita DFS

Con D generico

```
VISITA-DFS (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 stack (pila)

```
VISITA-DFS (G, s)
  D <- empty_stack()
  color [s] <- gray
  {visita s}
  push(D,s)
  while NotEmpty(D) do begin
    u <- top(D)
    if esiste v bianco adj ad u then
      color [v] <- gray
      π[v] <- u
      {visita v}
      push(D,v)
    else
      color [u] <- black
      pop(D)
  end
```

Visita DFS - ottimizzazione

ATTENZIONE: l'ottimizzazione è diversa da quella vista per la visita in ampiezza. Per quella in profondità top(D) può cambiare da ciclo a ciclo!

```
VISITA-DFS (G, s)
  D <- empty_stack()
  color [s] <- gray
  {visita s}
  push(D,s)
  while NotEmpty(D) do begin
     u <- top(D)
     if esiste v bianco adj ad u then
        color [v] <- gray
        π[v] <- u
        {visita v}
        push(D,v)
     else
        color [u] <- black
        pop(D)
  end
```

```
~~u <- top(D)~~
~~if~~ while esiste v non considerato adj a
top(D) then
   if color[v] = white
      color [v] <- gray
      π[v] <- top(D)
      {visita v}
      push(D,v)

color [top(D)] <- black
pop(D)
```

DOMANDA: possiamo usare un iteratore sulle liste di adiacenza come per la visita in ampiezza?

Visita DFS – con puntatori

```
VISITA-DFS (G, s)
  D <- empty_stack()
  color [s] <- gray
  {visita s}
  push(D,s)
  while NotEmpty(D) do begin
     while Ptr[top(D)] != NULL then
        v <- Ptr[top(D)].vert
        Ptr[top (D)] <- Ptr[top(D)].next
        if color[v] = white
           color [v] <- gray
           π[v] <- top(D)
           {visita v}
           push(D,v)
     color [top(D)] <- black
     pop(D)
  end
```

Ricorda che Ptr va adeguatamente inizializzato come vettore contenente i puntatori ai (primi) elementi delle liste di adiacenza dei nodi.

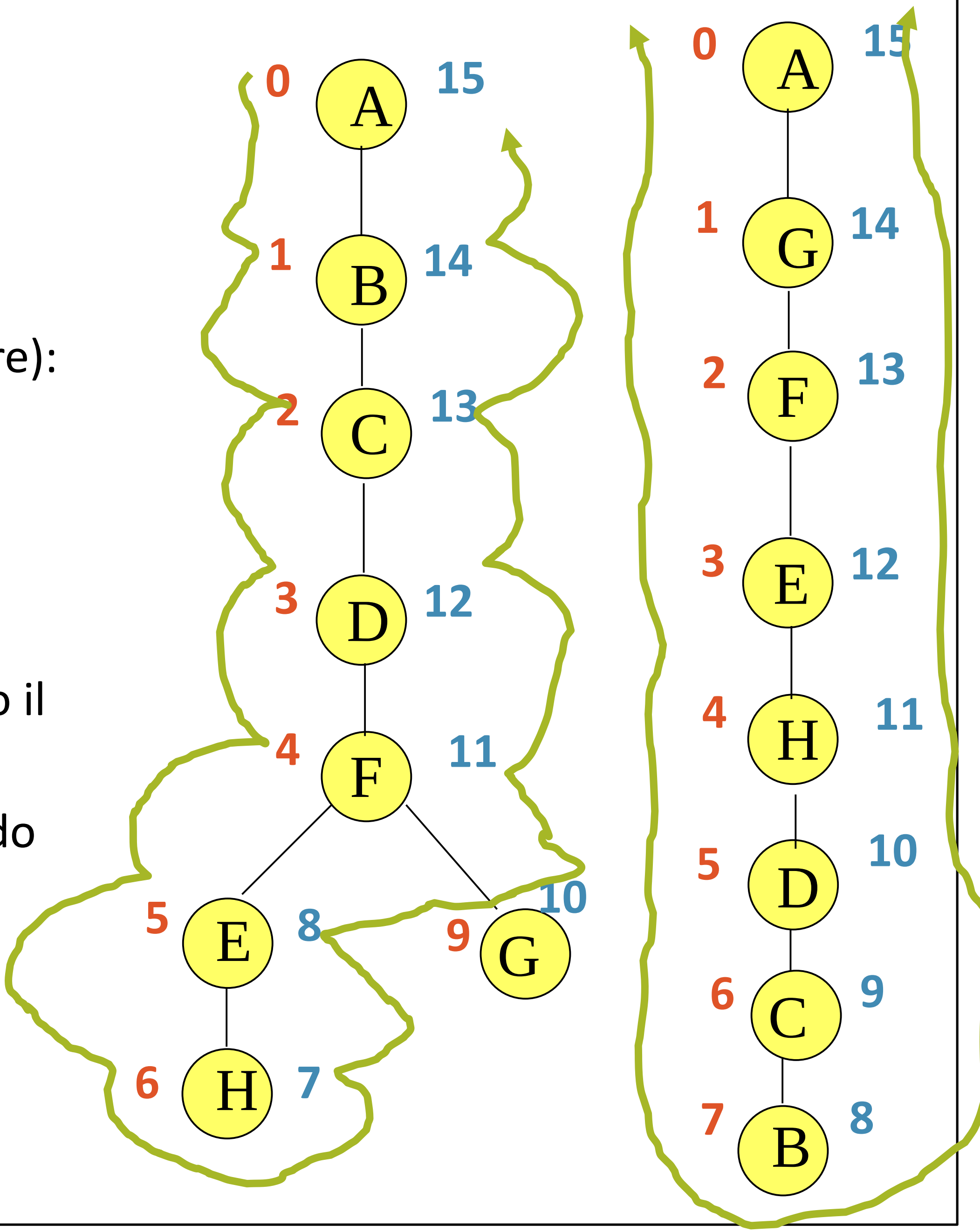
Caratteristiche dell’albero DFS

Consideriamo (usando un contatore):

- L’ordine in cui i vertici vengono **scoperti** (diventano grigi)
- L’ordine in cui i vertici vengono **chiusi** (diventano neri)

L’albero viene creato dall’alto verso il basso, in profondità.

Un vertice viene chiuso solo quando tutti i suoi discendenti sono stati chiusi.



Consideriamo gli intervalli di attivazione

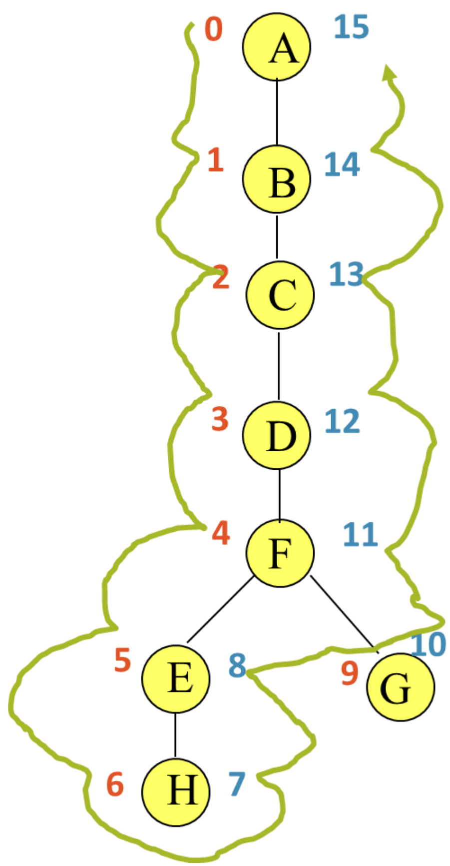
Gli intervalli di “attivazione”

di una qualunque

coppia di vertici sono:

- **disgiunti**, oppure
- **uno interamente contenuto nell’altro**

È l’ordine delle attivazioni delle chiamate di una procedura **ricorsiva**!



Visita in profondità ricorsiva

Costruiamo un algoritmo ricorsivo per effettuare una visita in profondità.

```
VISITA-DFS-ric (G, u)
   color [u] <- gray
   {visita u}
   for ogni v adiacente ad u
      if color[v] = white
         π[v] <- u
         VISITA-DFS-ric (G, v)
   color[u] <- black
```

Ricorda: questo è pseudocodice.

Introduciamo un contatore

Introduciamo un contatore **time** per ricordare l'ordine delle attivazioni e disattivazioni e i due attributi **d** (attivazione) e **f** (disattivazione)

```
VISITA-DFS-ric (G, u)
   color [u] <- gray
   {visita u}
   d[u] <- time
   time++
   for ogni v adiacente ad u
      if color[v] = white
         π[v] <- u
         VISITA-DFS-ric (G, v)
   color[u] <- black
   f[u] <- time
   time++
```

```
INIZIALIZZA (G)
   …
   for ogni vertice u ∈ V[G] do
         color [u] <- white
         π[u] <- NULL
         d[u] <- f[u]<- +∞
   time <- 0
```

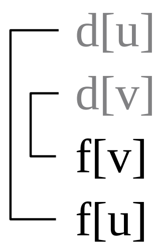
N.B.: Se un vertice non viene "visitato" i suoi tempi di attivazione e disattivazione restano "**infiniti**".

N.B.2: d[u] in una visita DFS **non è** il vettore delle distanze.

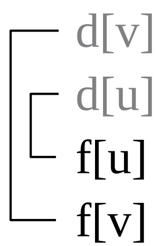
Teorema delle parentesi

In ogni visita DFS di un grafo (orientato o non orientato), per ogni coppia di vertici u, v **una e una sola** delle seguenti condizioni è soddisfatta:

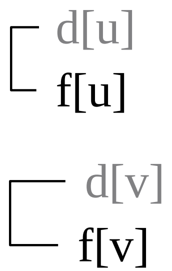
- d[u] < d[v] < f[v] < f[u] e u è un antenato di v in un albero della foresta DFS



- d[v] < d[u] < f[u] < f[v] e u è un discendente di v in un albero della foresta DFS



- d[u] < f[u] < d[v] < f[v] e tra u e v non esiste relazione di antenato – discendente nella foresta DFS



Corollari

Teorema dell'annidamento degli intervalli

Una visita DFS di un grafo (orientato o non orientato), colloca un vertice v come discendente proprio di un vertice u in un albero della foresta DFS **<==>** **d[u] < d[v] < f[v] < f[u]**

Teorema del cammino bianco

In una foresta DFS un vertice v è discendente di un vertice u **<==>** al tempo d[u] v è raggiungibile da u con un cammino contenente **solo vertici bianchi**

Se il grafo non è connesso

***VISITA_TUTTI_I_VERTICI-DFS* (G)**

INIZIALIZZA (G)

for ogni u ∈ V **do**

if color [u] = white

then *VISITA-DFS* (G, u)

oppure

***VISITA_TUTTI_I_VERTICI-DFS-ric* (G)**

INIZIALIZZA (G)

for ogni u ∈ V **do**

if color [u] = white

then *VISITA-DFS-ric* (G, u)

Complessità visita DFS

Dato che la visita in profondità è derivabile dalla visita generica, che come abbiamo visto ha complessità O(m+n), la complessità di una visita DFS è **O(m+n)**.

Cosa devo aver capito fino ad ora

- Come funziona la visita in profondità DFS di grafi
- Come si implementa a partire dalla visita generica
- Come viene costruito l'albero dei predecessori di una visita DFS e quali sono le sue caratteristiche
- Come si implementa una visita DFS ricorsiva e quali sono le caratteristiche del suo stack di attivazione
- Teorema delle parentesi e corollari

…se non ho capito qualcosa

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