Algoritmi e Strutture Dati (Mod. B)

Algoritmi su grafi

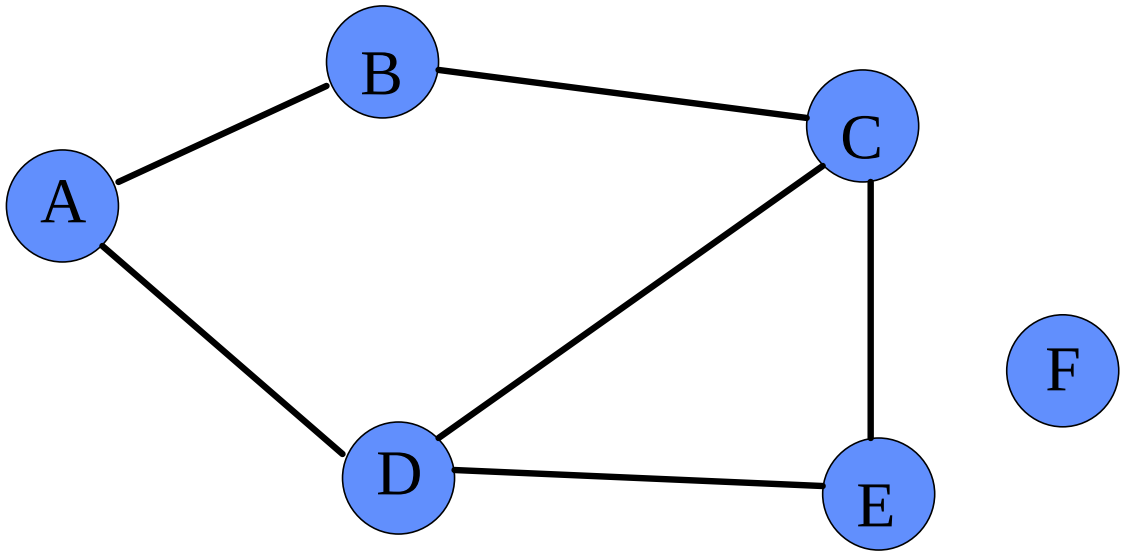
Ricerca in ampiezza

(Breadth-First Search)

Definizione del problema

Attraversamento di un grafo

Dato un grafo *G*=<*V*,*E*> ed un vertice *s* di *V* (detto *sorgente*), esplorare *ogni vertice raggiungibile* nel grafo dal vertice *s*

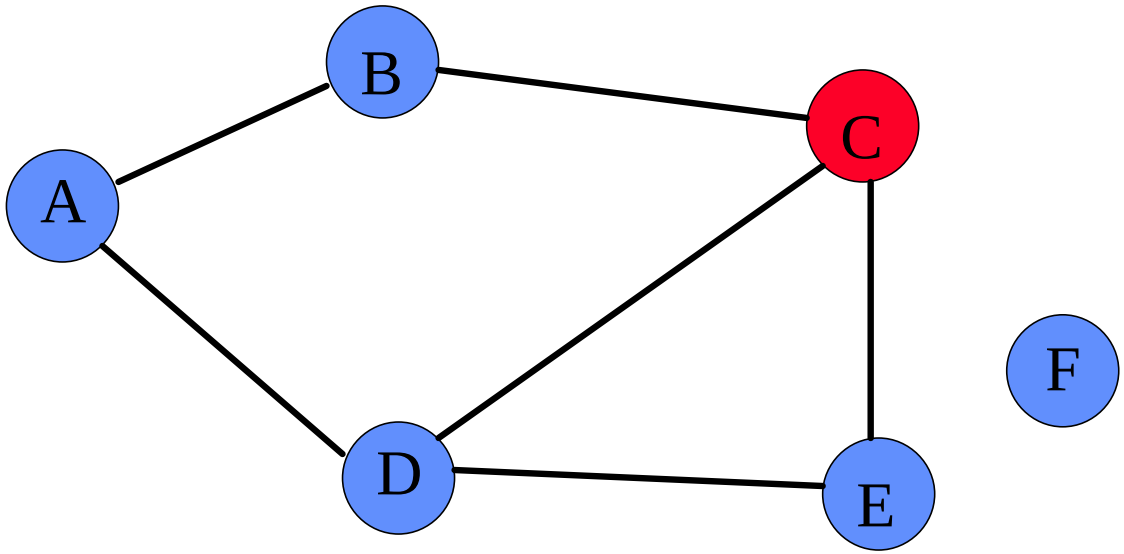


***s* = C**

Definizione del problema

Attraversamentodi un grafo

Dato un grafo *G*=<*V*,*E*> ed un vertice *s* di *V* (detto *sorgente*), esplorare *ogni vertice raggiungibile* nel grafo dal vertice *s*

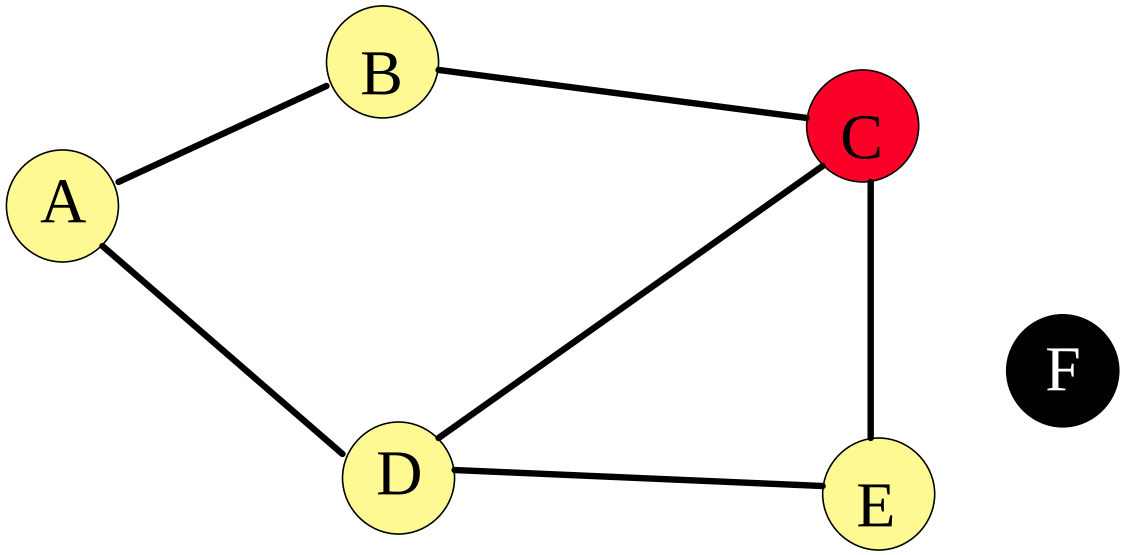


***s* = C**

Definizione del problema

Attraversamentodi un grafo

Dato un grafo *G*=<*V*,*E*> ed un vertice *s* di *V* (detto *sorgente*), esplorare *ogni vertice raggiungibile* nel grafo dal vertice *s*



***s* = C**

F è l'unico vertice non raggiungibile

Algoritmo BFS per alberi

```
Visita-Ampiezza(T:albero)
      Coda ={T}
      while Coda ≠ ∅
         do u = Testa[Coda]
            "visita u"
            for each "figlio F di u da sinistra"
               do Accoda(Coda,F)
            Decoda(Coda)
```

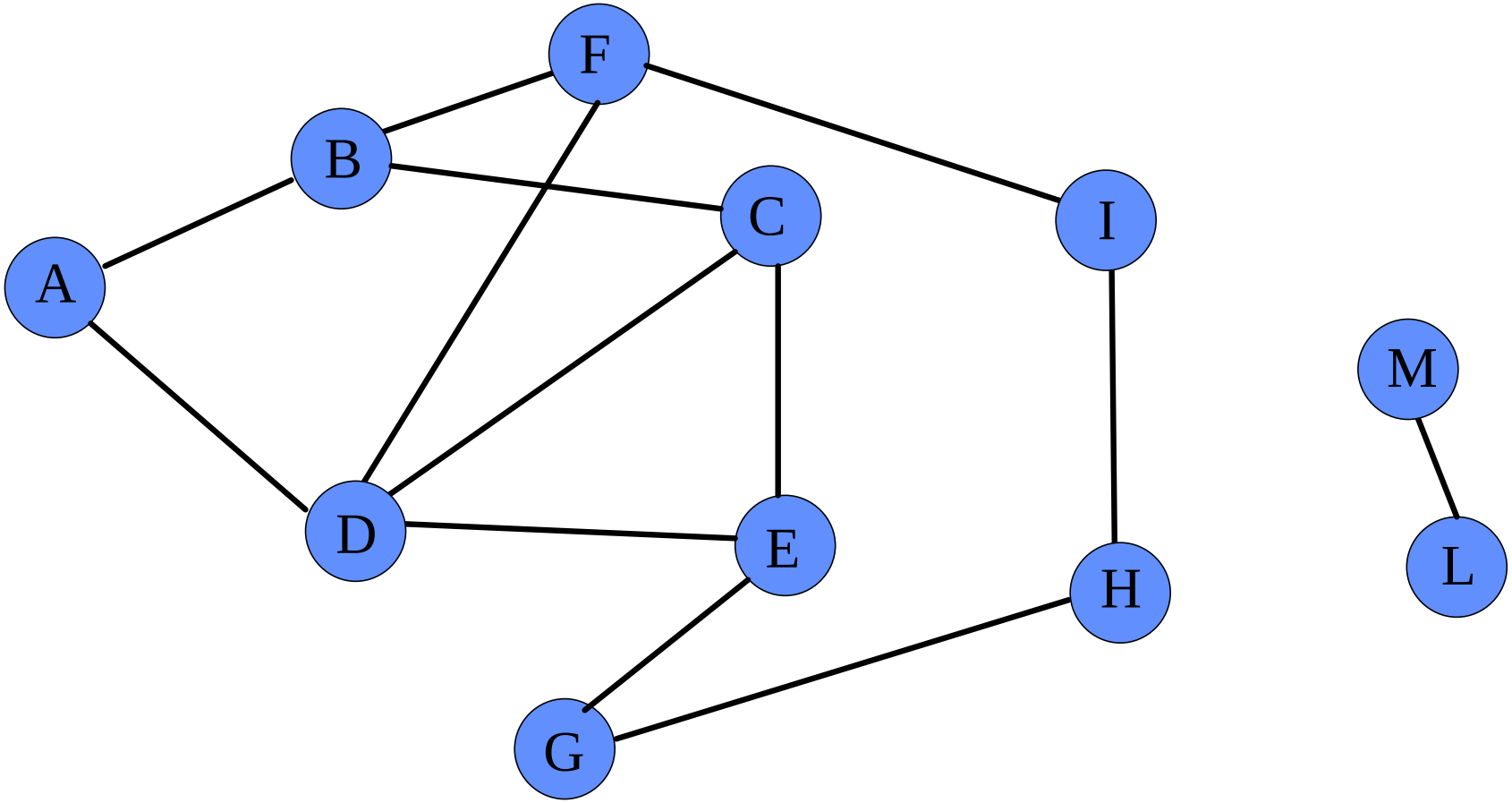
Nel nostro algoritmo generico per i grafi, come operazione di *"visita"* di un vertice useremo la memorizzazione di quale sia il *"padre"* (o *avo immediato*) del vertice durante la *BFS*.

Algoritmo BFS: I

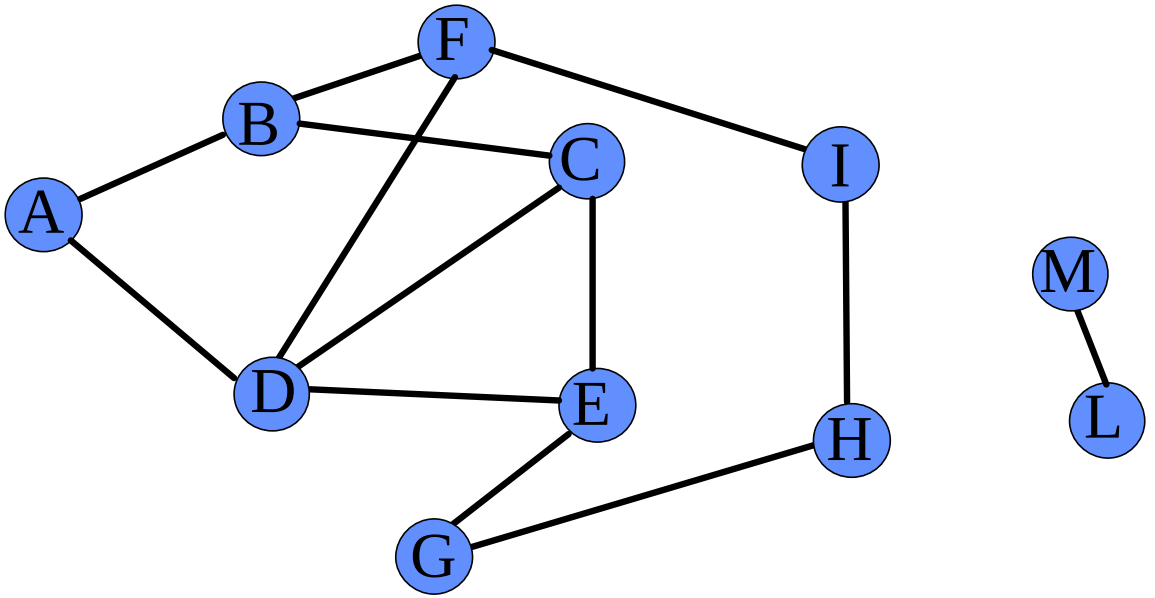
```
BSF(G:grafo, s:vertice)
  pred[s] = Nil
  Coda = {s}
  while Coda ¹ ∅
     do u = Testa[Coda]
        for each v Î Adiac(u)
            do pred[v] = u
               Accoda(Coda,v)
        Decoda(Coda)
```

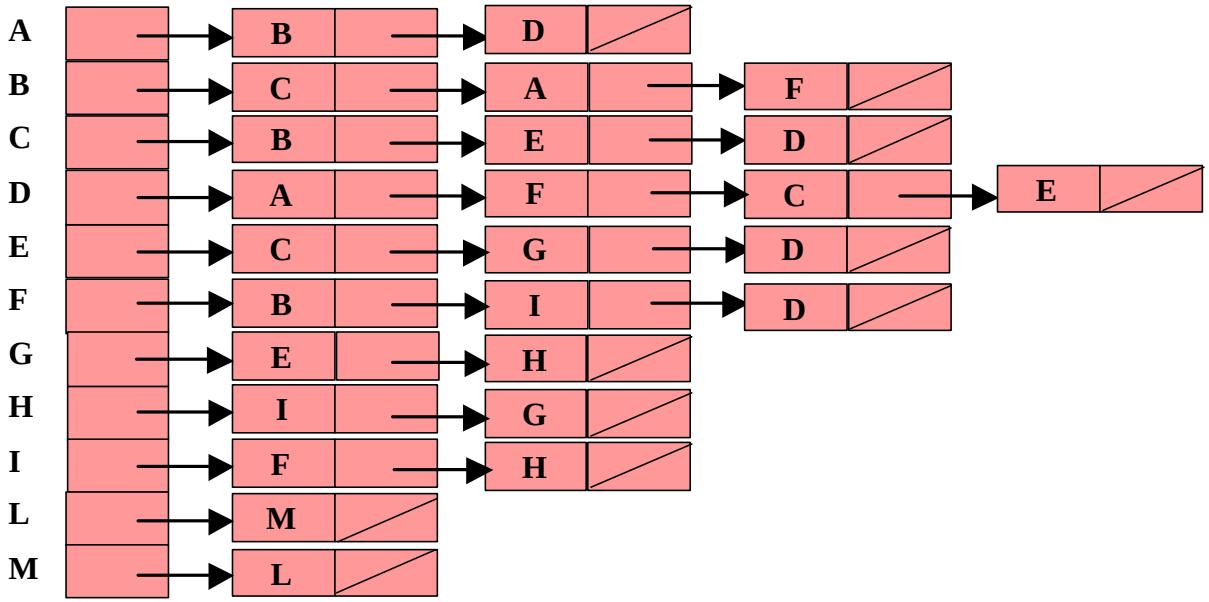
```
Visita-Ampiezza(T:albero)
      Coda = {T}
      while Coda ≠ ∅
        do u = Testa[Coda ]
           "visita u"
           for each "figlio F di u da sinistra"
               do Accoda(Coda,F)
           Decoda(Coda)
```

Algoritmo BFS I

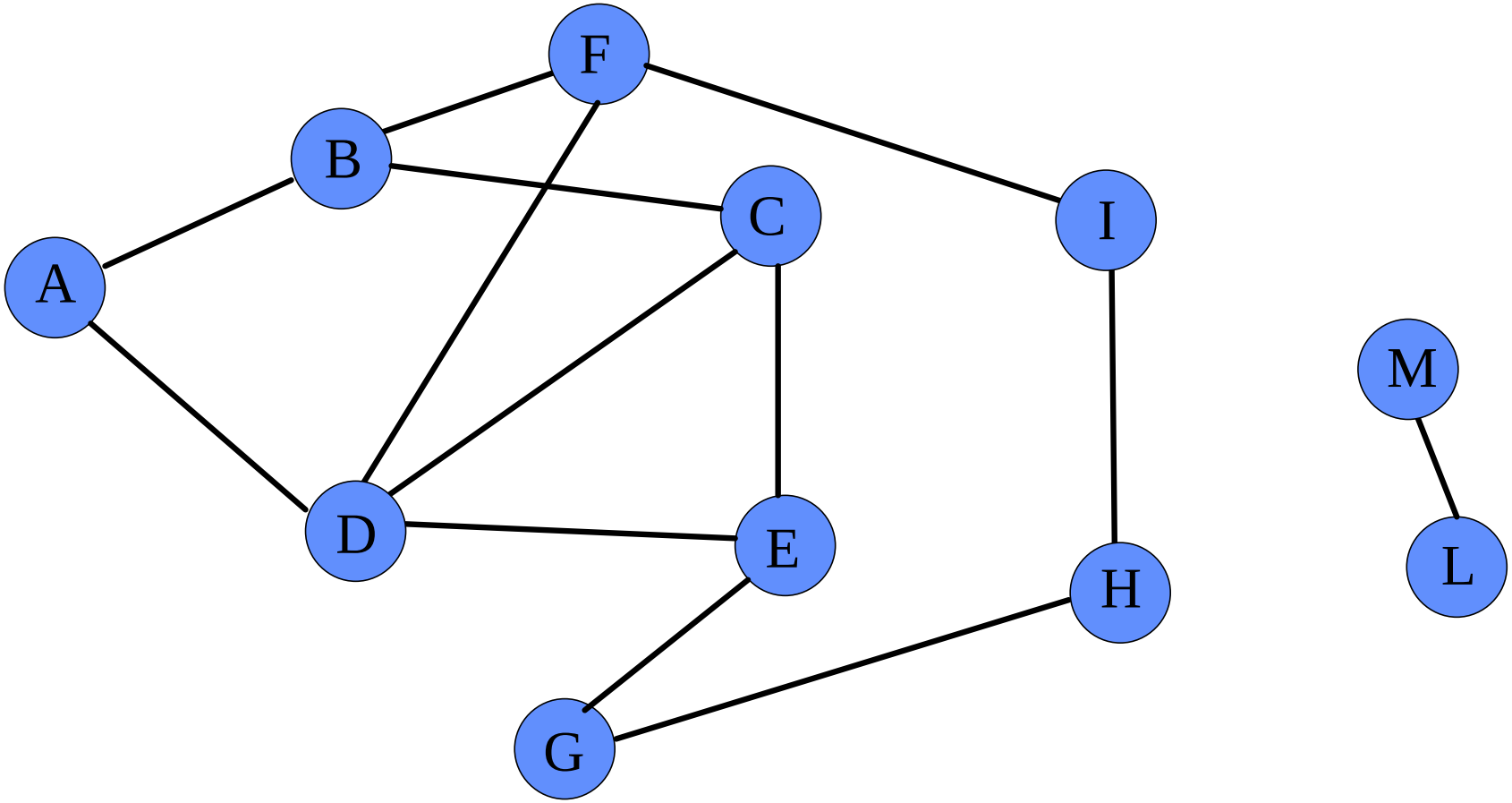


Algoritmo BFS I





Algoritmo BFS I

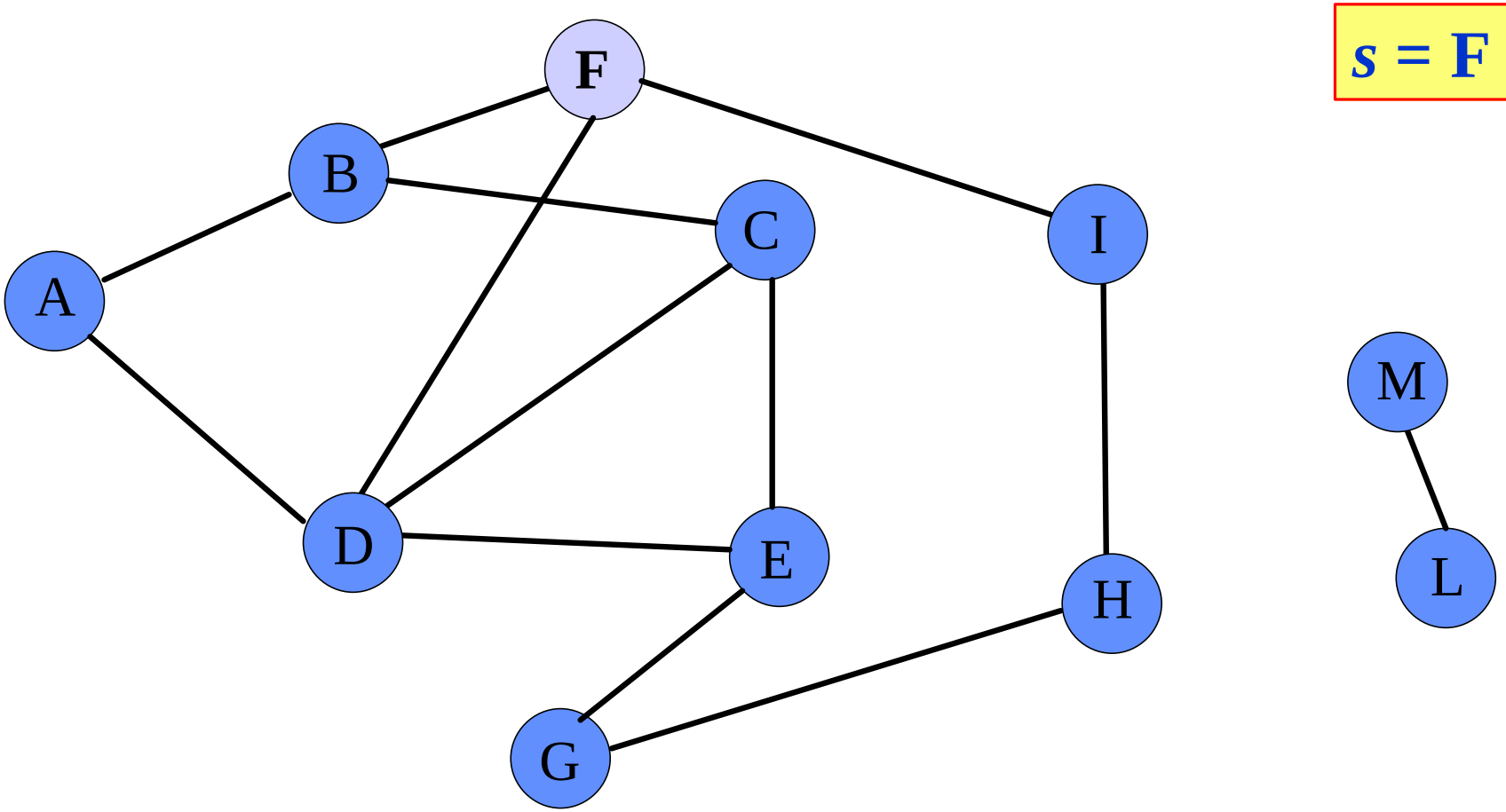


Coda:{}

Algoritmo BFS I

```
for each v Î Adiac(u)
  do pred[v] = u
     Accoda(Coda,v)
Decoda(u)
```

$s = F$

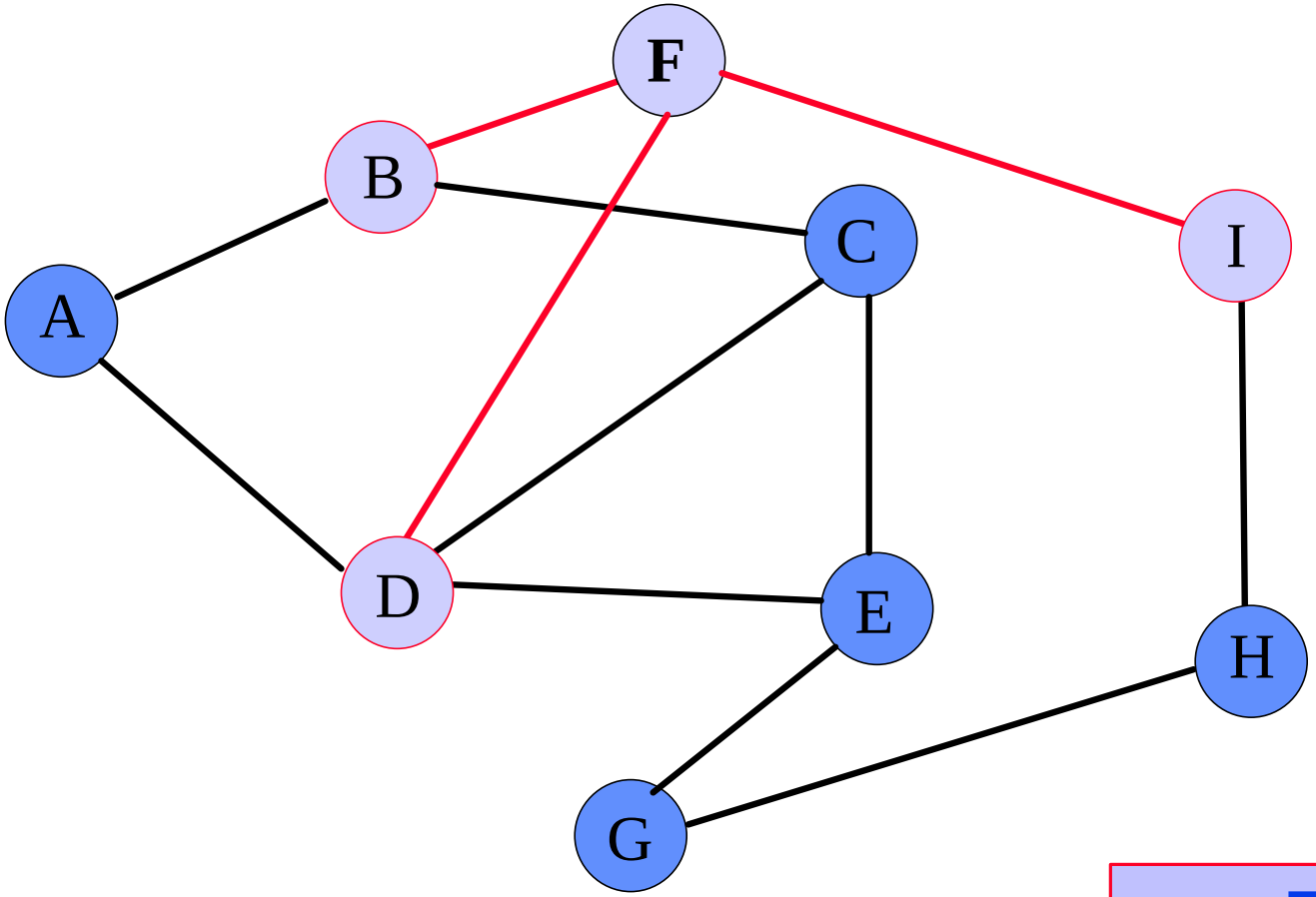


Coda:{F}

Algoritmo BFS I

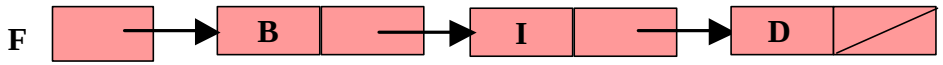
```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```

$s = F$



$u = F$

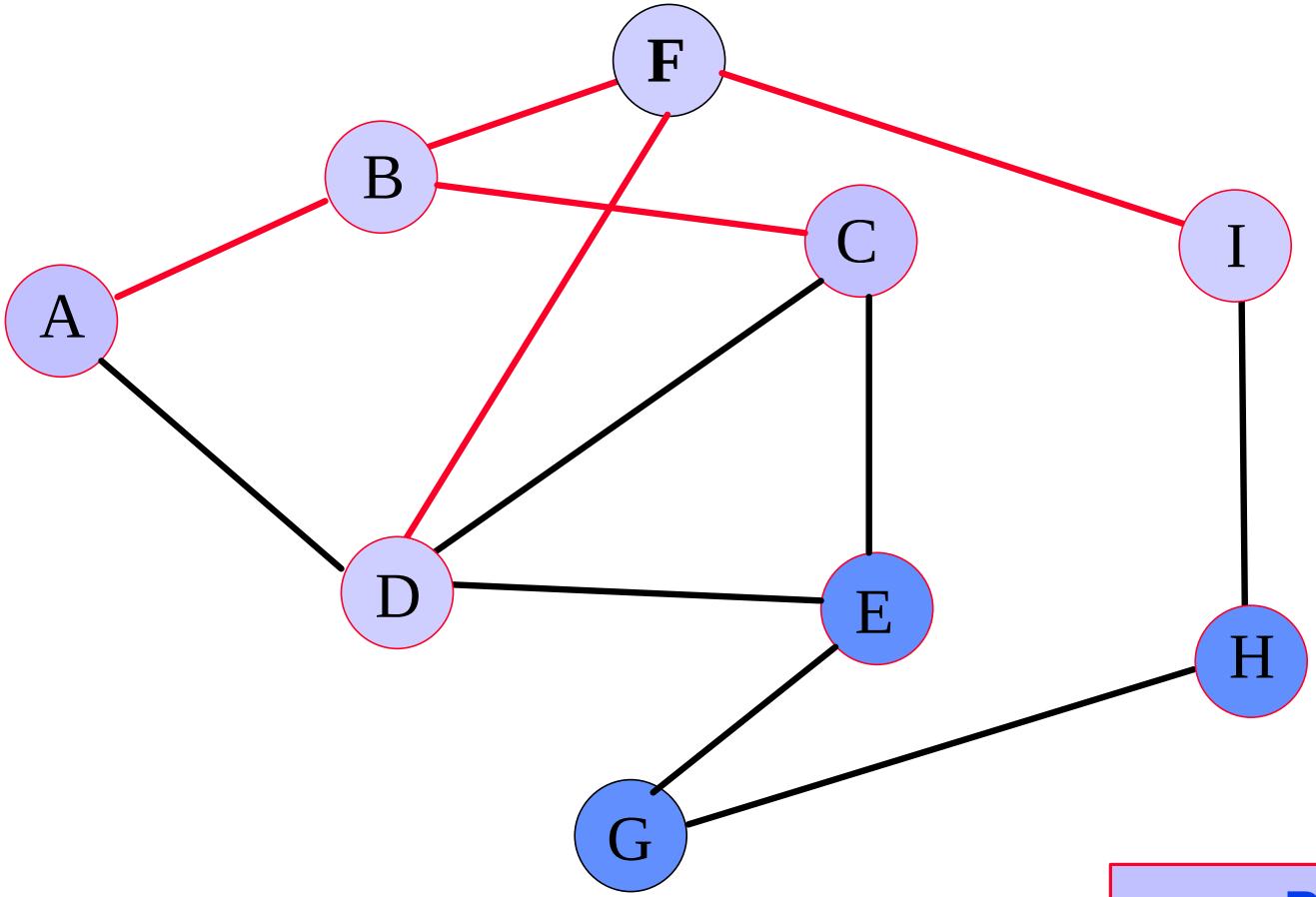
Coda: {B,I,D}

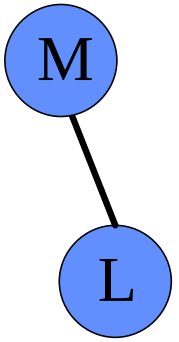


Algoritmo BFS I

```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```

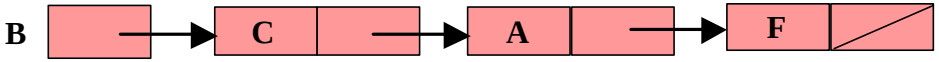
$s = F$





$u = B$

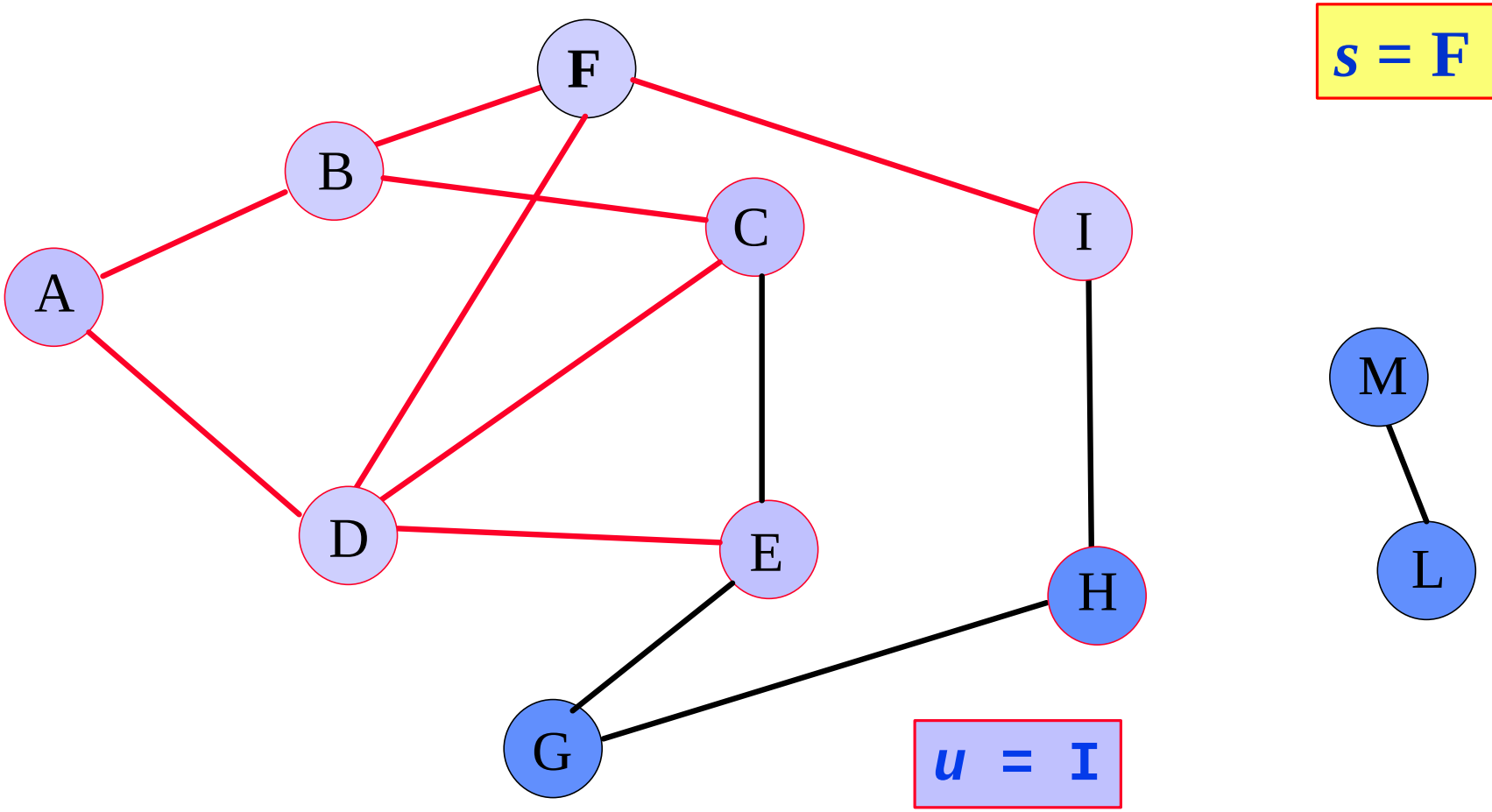
Coda: {I, D, C, A, F}



Algoritmo BFS I

```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```

$s = F$



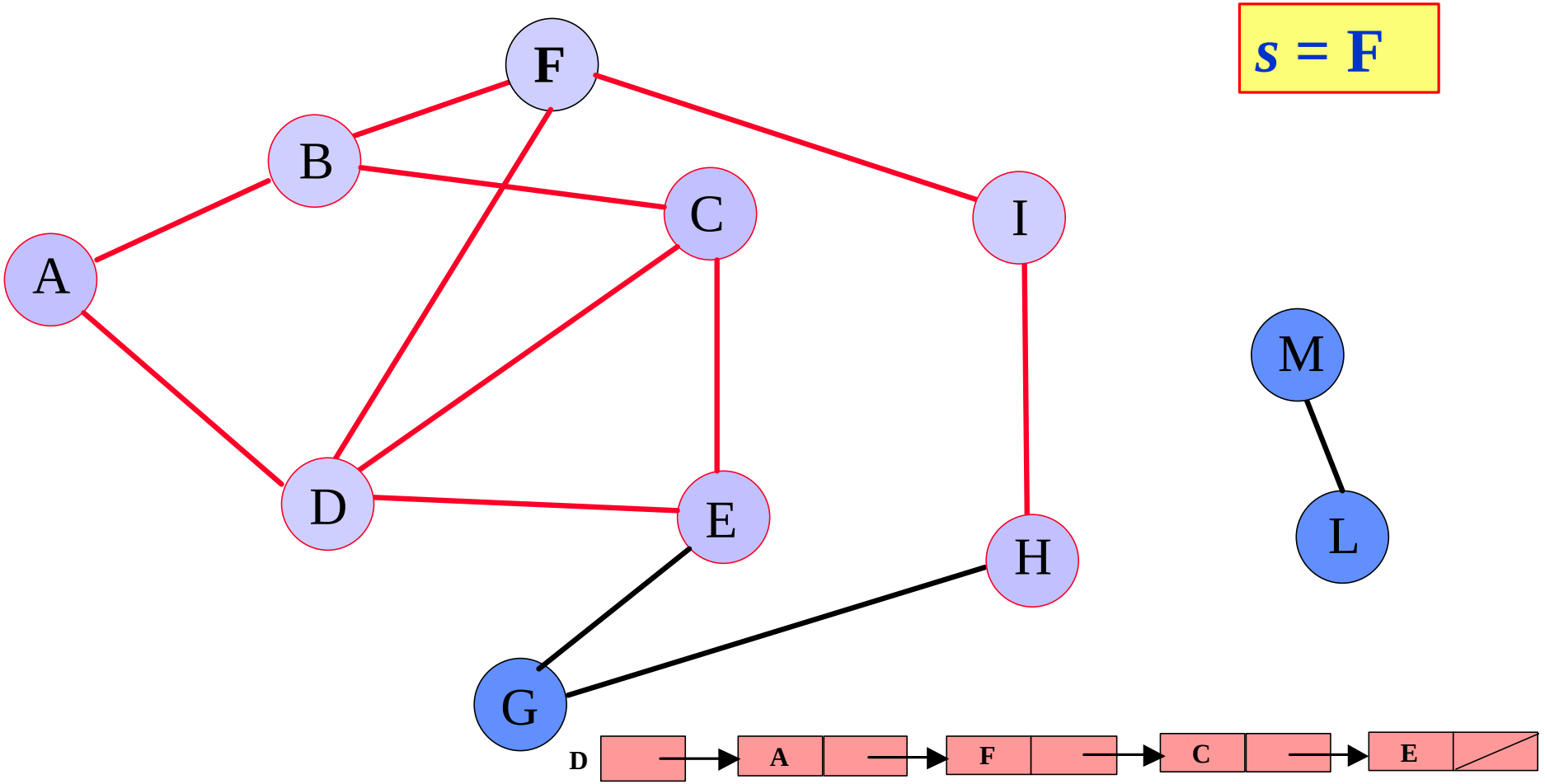
$u = I$

Coda: {D, A, C, F, F, H}

Algoritmo BFS I

```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```

$s = F$

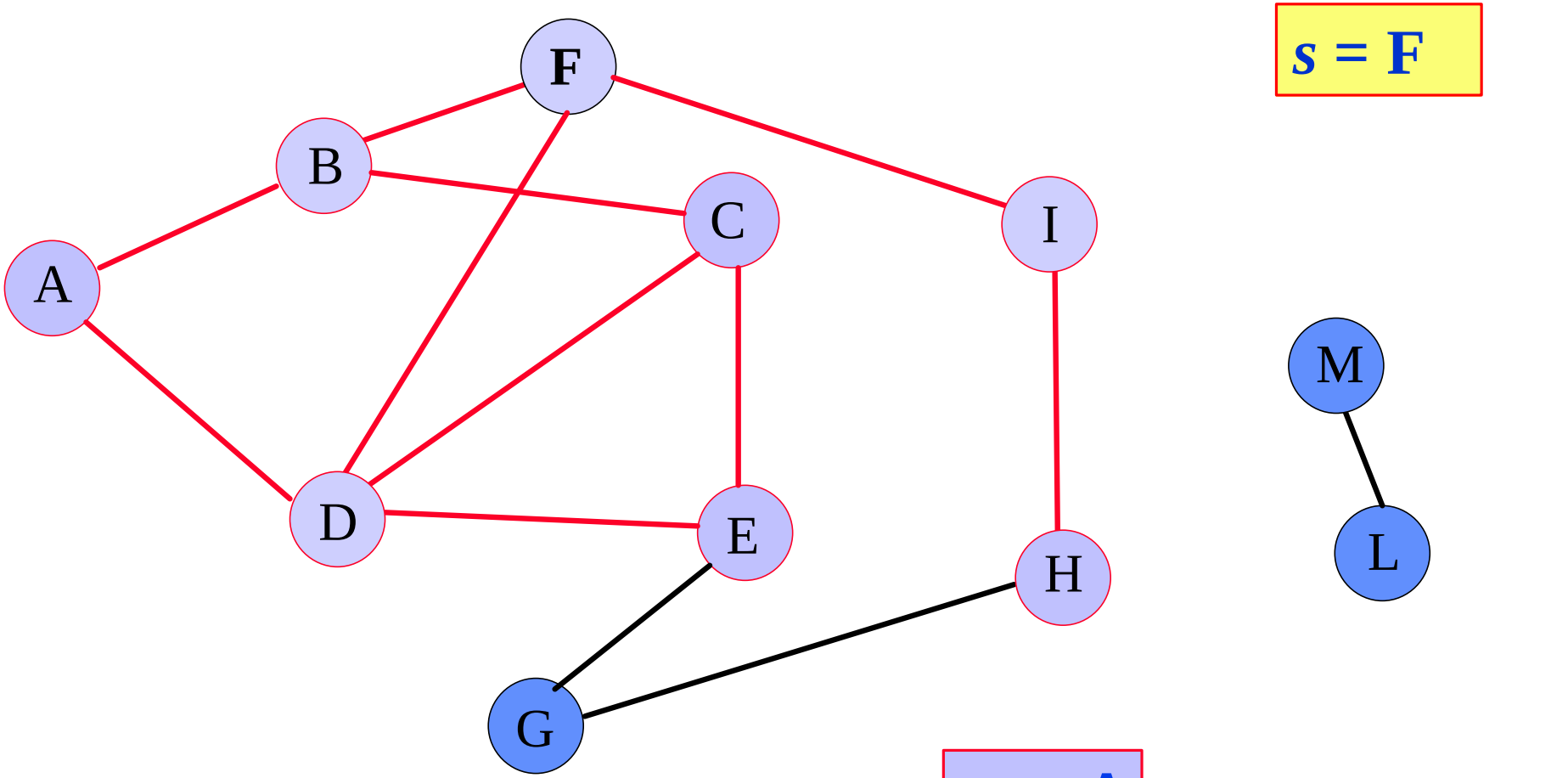


Coda: {A, C, F, F, H, A, F, C, E}

$u = D$

Algoritmo BFS I

```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```



s = F

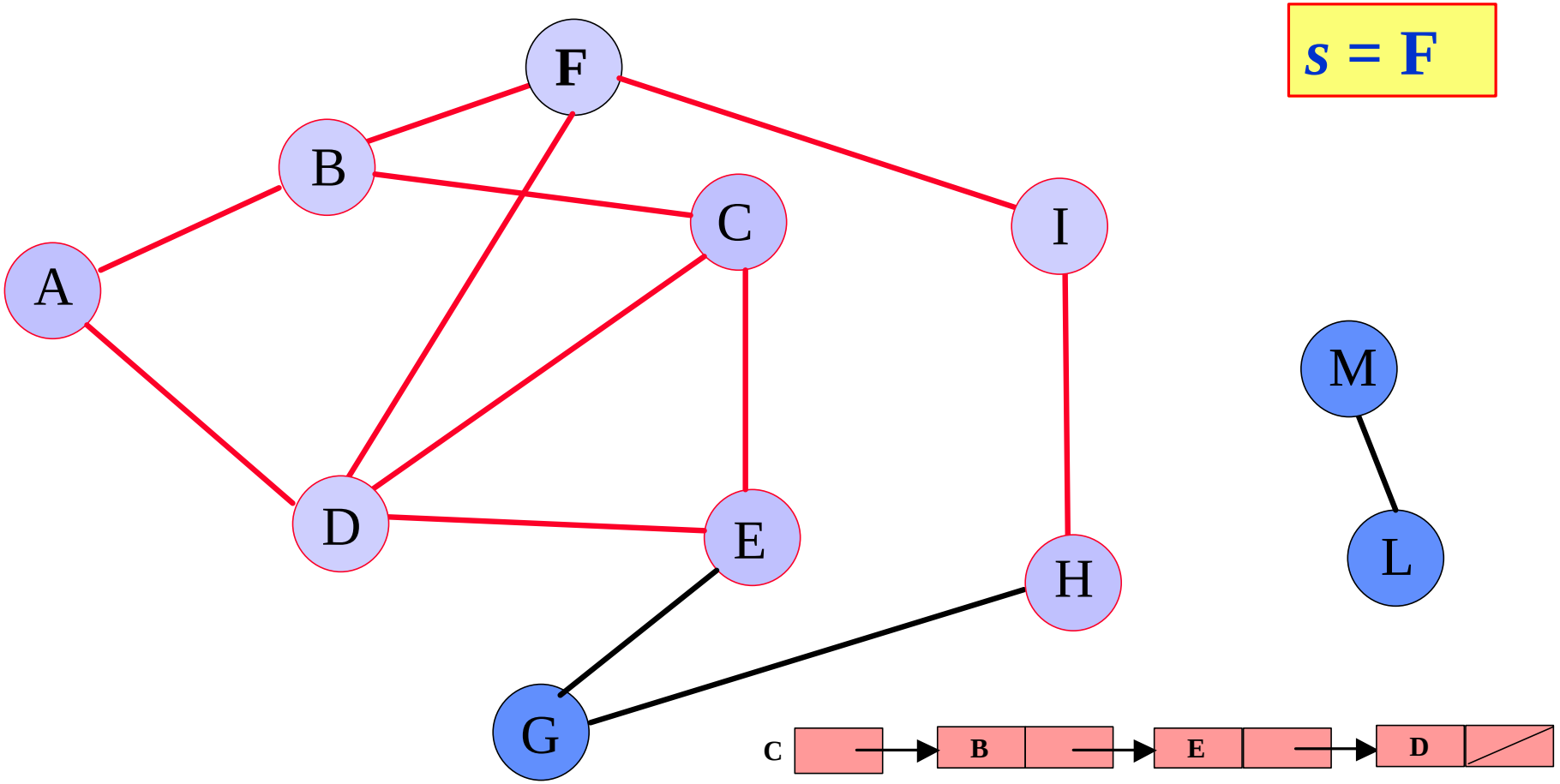
u = A

Coda: {C, F, F, H, A, F, C, E, B, D}

Algoritmo BFS I

```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```

$s = F$



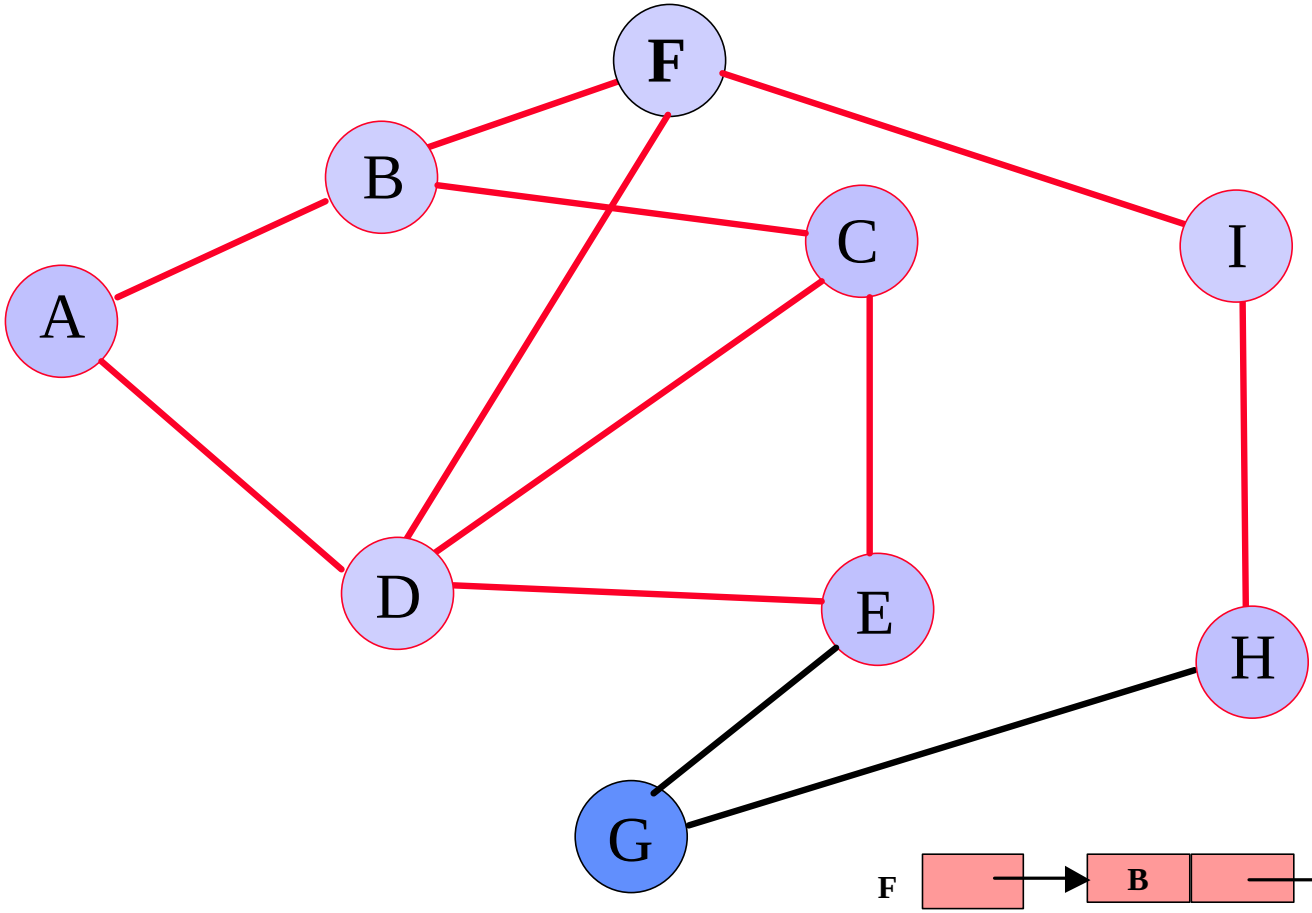
$u = C$

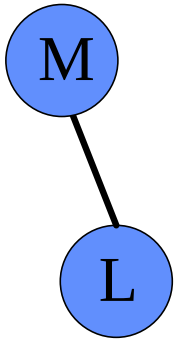
Coda:{F,F,H,A,F,C,E,B,D,B,E,D}

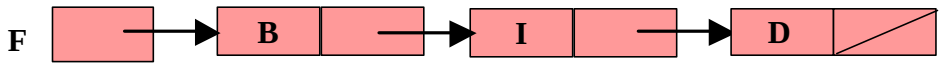
Algoritmo BFS I

```
for each v Î Adiac(u)
   do pred[v] = u
      Accoda(Coda,v)
Decoda(u)
```

$s = F$







Coda:{F,F,H,A,F,C,E,B,D,B,E,D,B,I,D}

$u = F$

Algoritmo BFS I: problema

- **È necessario ricordarsi dei nodi che abbiamo già visitato per non rivisitarli nuovamente.**

- **Dobbiamo distinguere tra i vertici non visitati, quelli visitati e quelli processati.**

Algoritmo BFS I: problema

- **È necessario ricordarsi dei nodi che abbiamo già visitato per non rivisitarli nuovamente.**

- **Dobbiamo distinguere tra i vertici non visitati, quelli visitati e quelli processati**
  - **un vertice è stato *visitato* se è comparso nella coda**
  - **un vertice è stato *non visitato* se non è stato ancora visitato**
  - **un vertice è stato *processato* se tutti i vertici ad esso adiacenti sono stati visitati**

Algoritmo BFS II: soluzione

- **Per distinguere tra i vertici non visitati, quelli visitati, e quelli processati colo-reremo**
  - **ogni vertice *visitato* di grigio**
  - **ogni vertice *non visitato* di bianco**
  - **ogni vertice *processato* di nero**

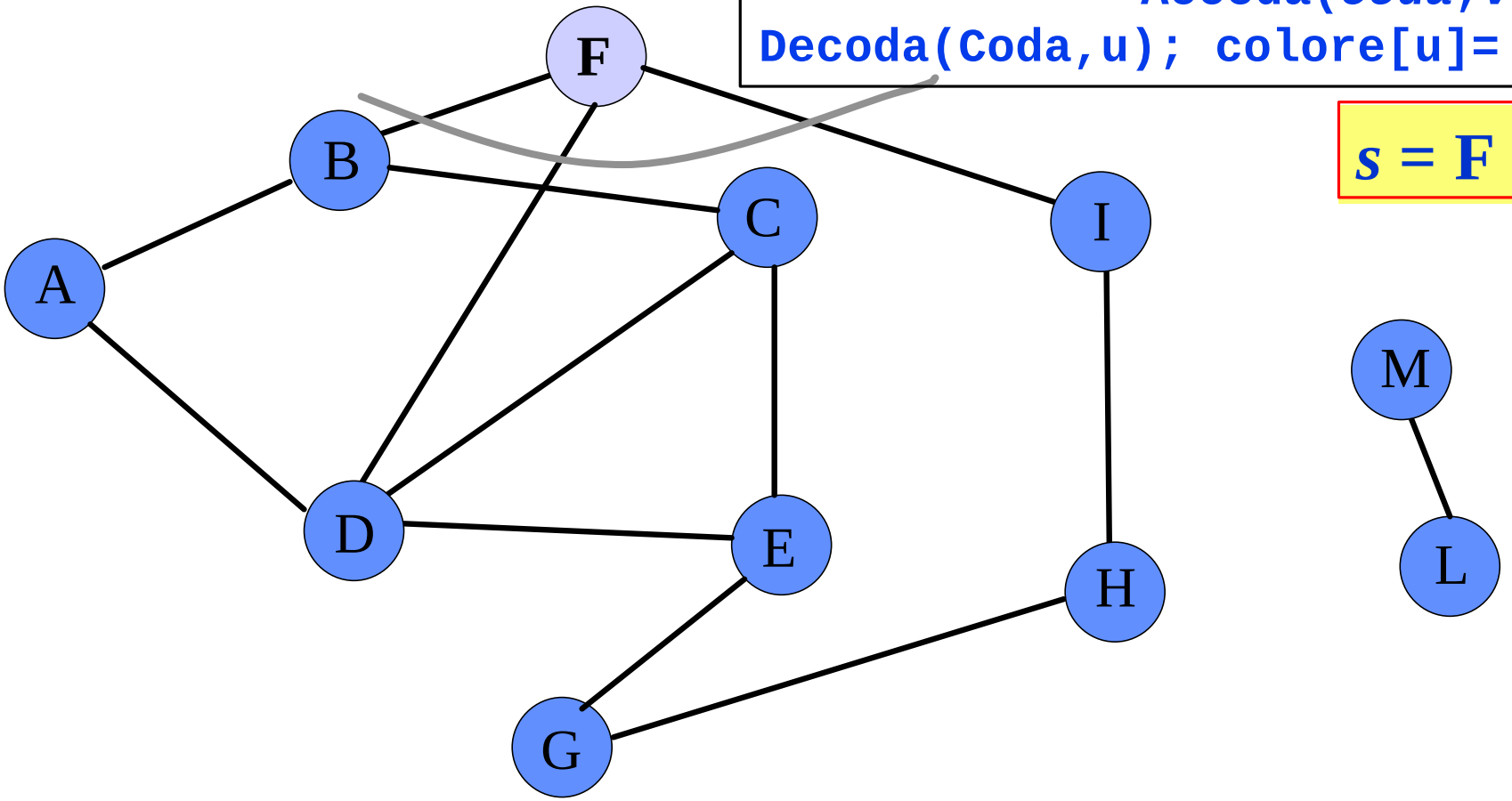
Algoritmo BFS II: soluzione

- **Per distinguere tra i vertici non visitati, quelli visitati, e quelli processati colo-reremo**
  - **ogni vertice *visitato* di grigio**
  - **ogni vertice *non visitato* di bianco**
  - **ogni vertice *processato* di nero**
- **Vengono *accodati* solo i vertici che *non* sono ancora stati *visitati* (cioè *bianchi*)**
- **I vertici in *coda* saranno i vertici *visitati* e *non* ancora *processati* (cioè *grigi*)**
- **I vertici *processati* non vengono più visitati**

Algoritmo BFS II

```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```

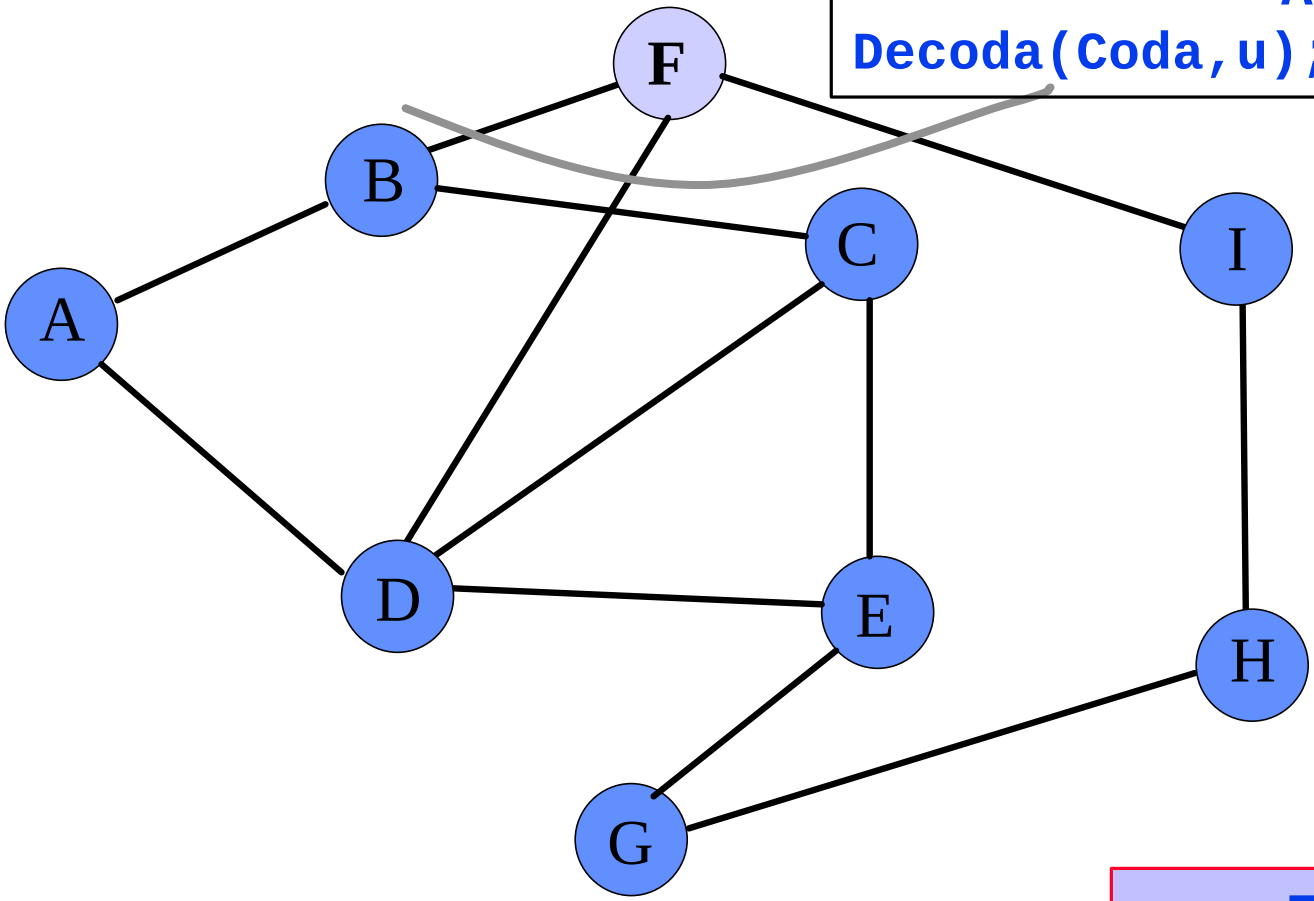
$s = F$



Coda:{F}

Algoritmo BFS II

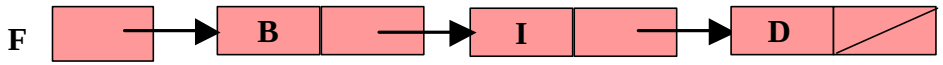
```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



$s = F$

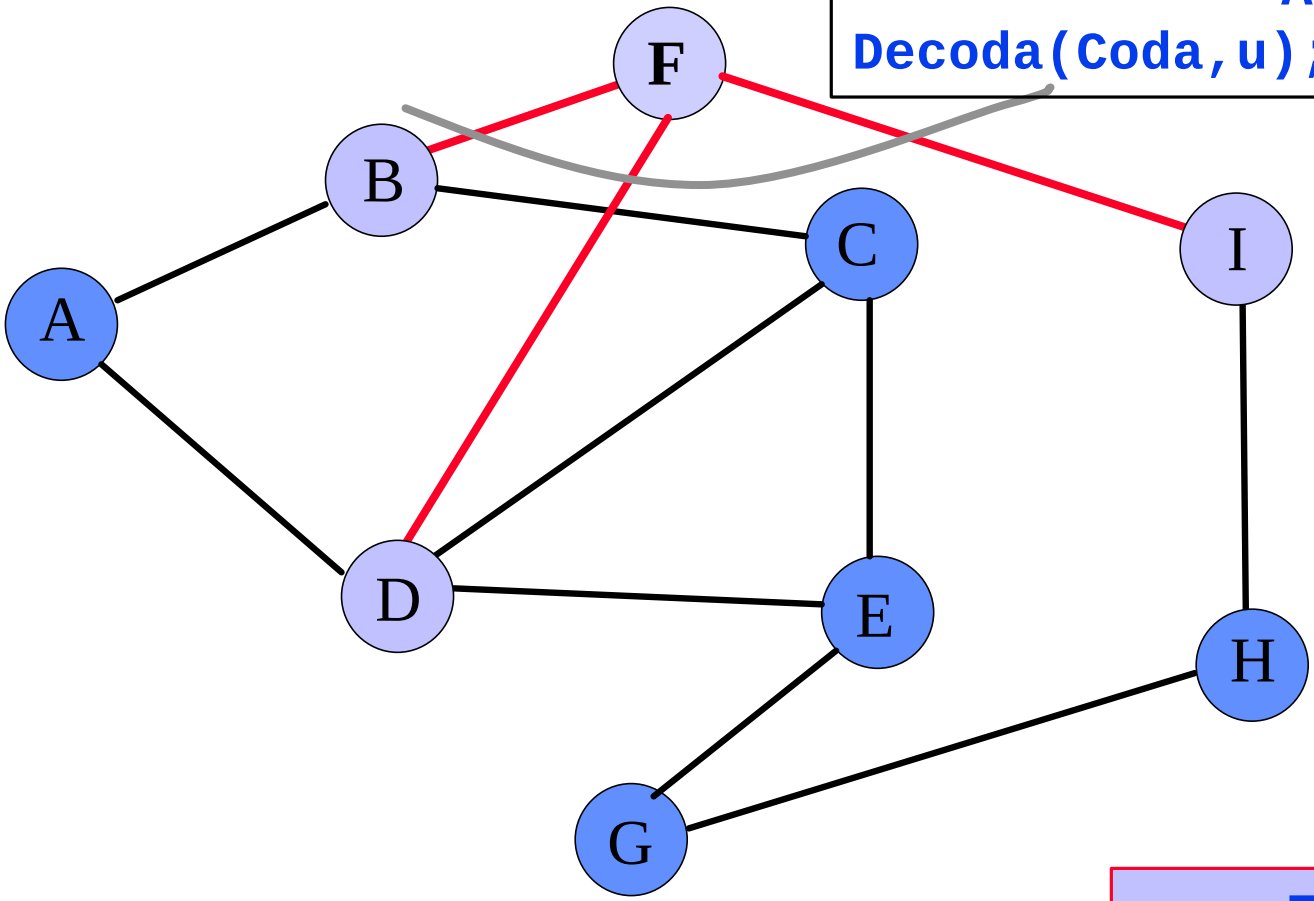
$u = F$

Coda:{F}



Algoritmo BFS II

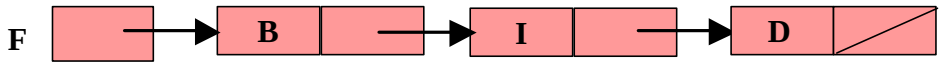
```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



$s = F$

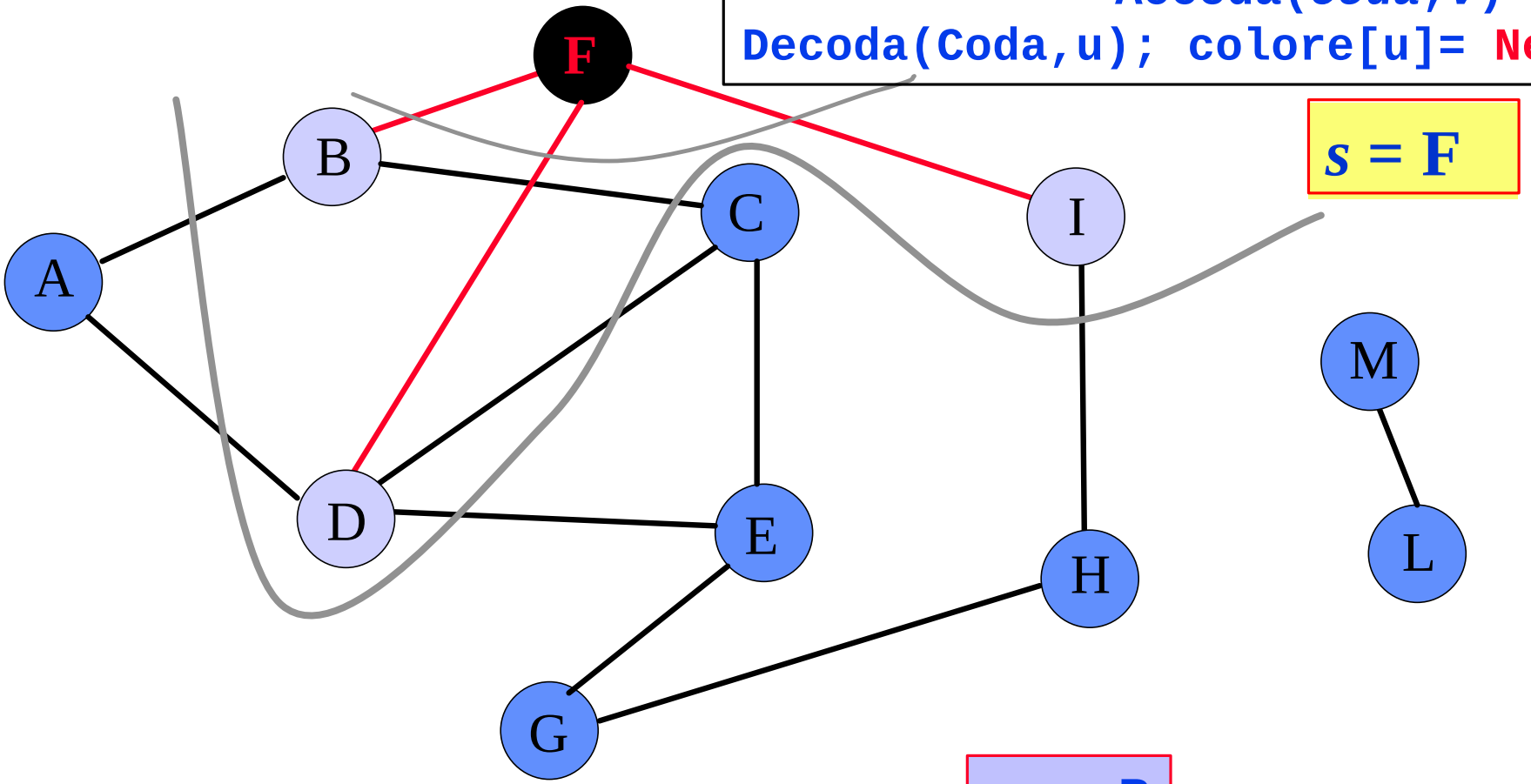
$u = F$

Coda:{F,B,I,D}



Algoritmo BFS II

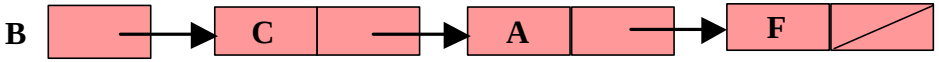
```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



s = F

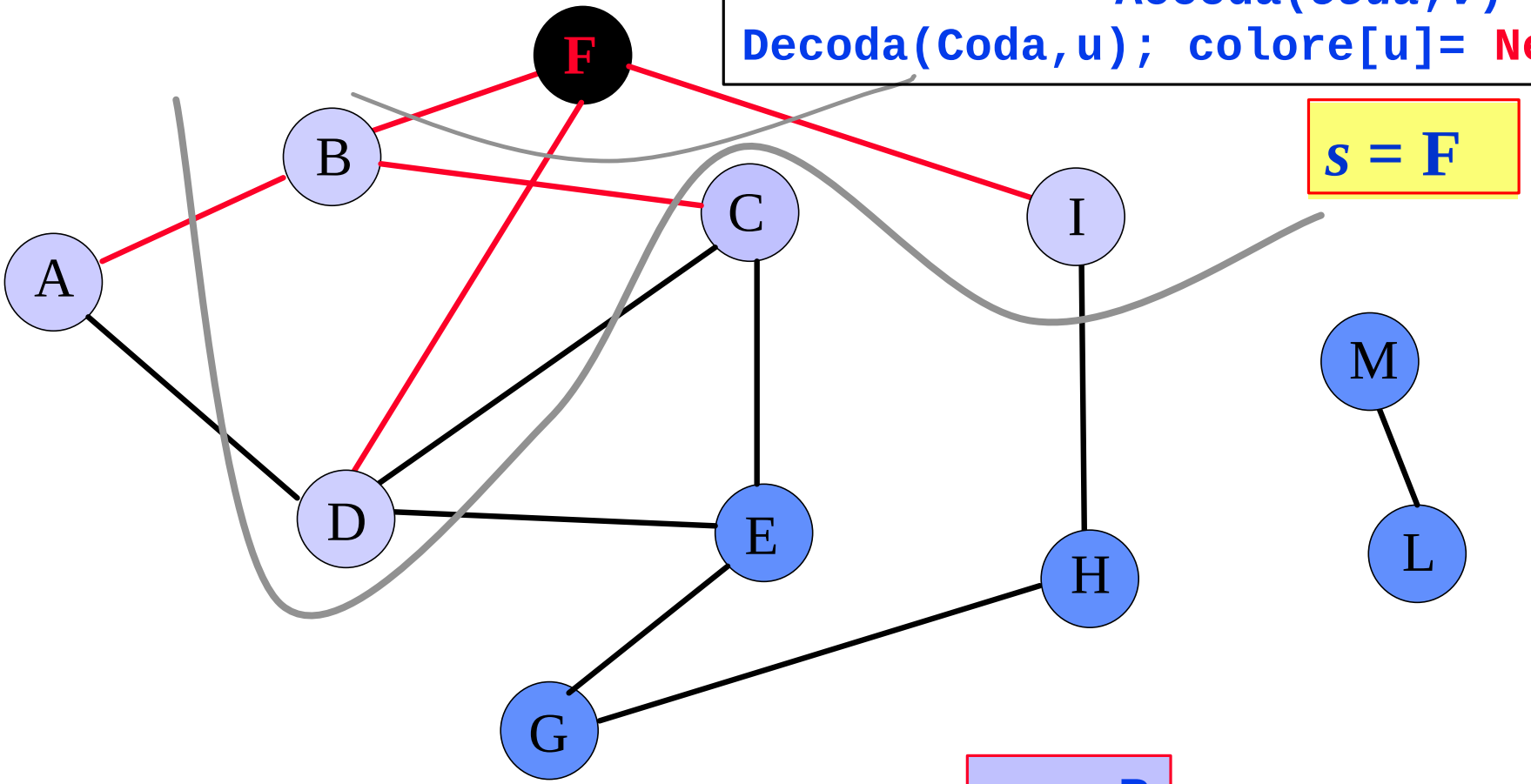
u = B

Coda:{B,I,D}



Algoritmo BFS II

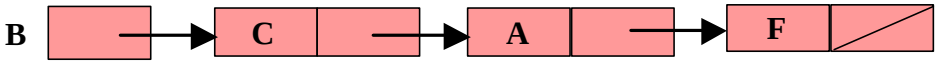
```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



s = F

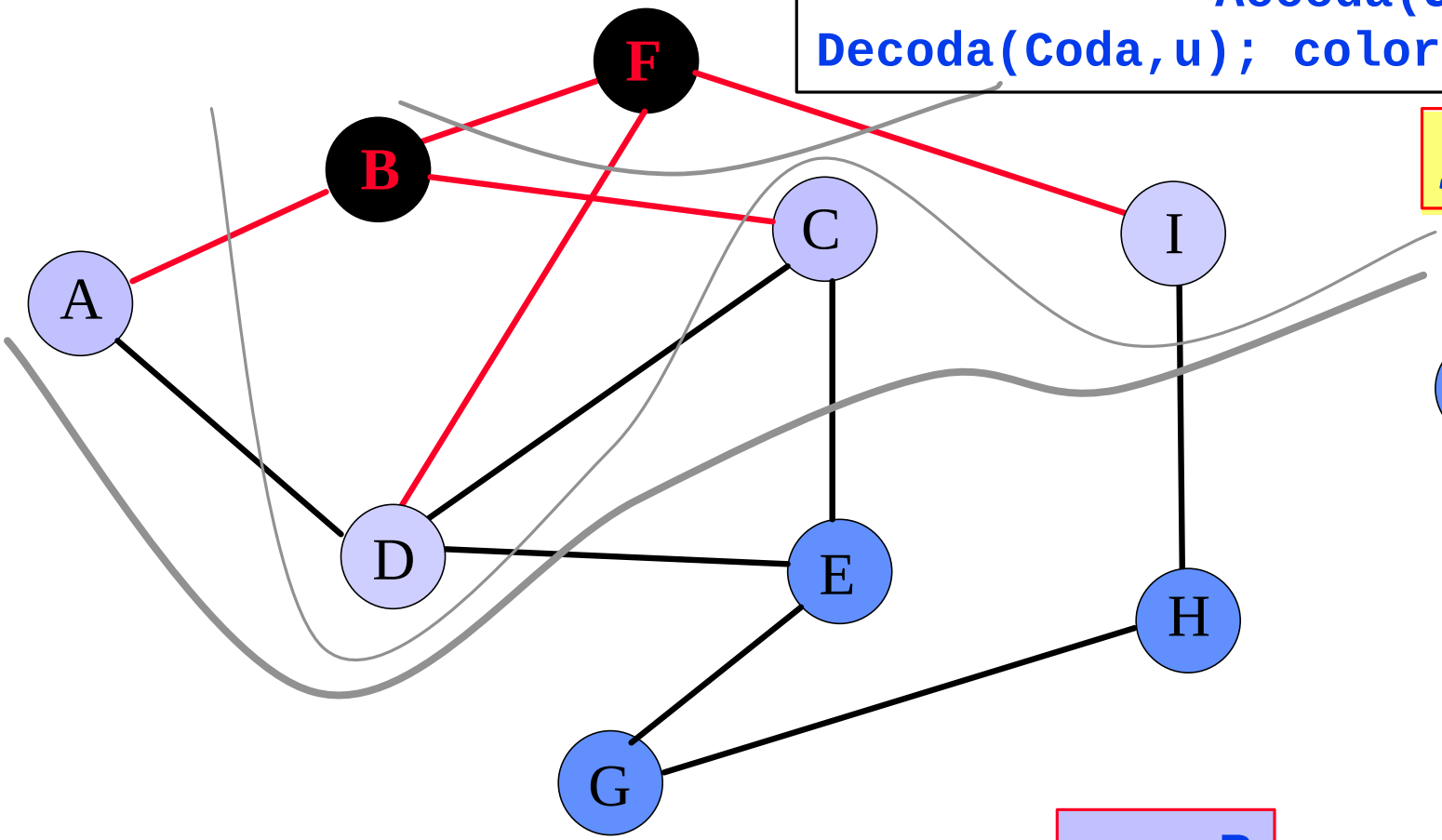
u = B

Coda:{B,I,D,C,A}



Algoritmo BFS II

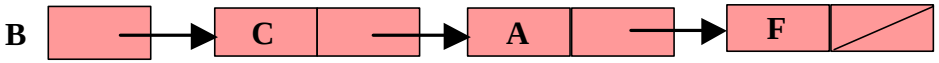
```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



s = F

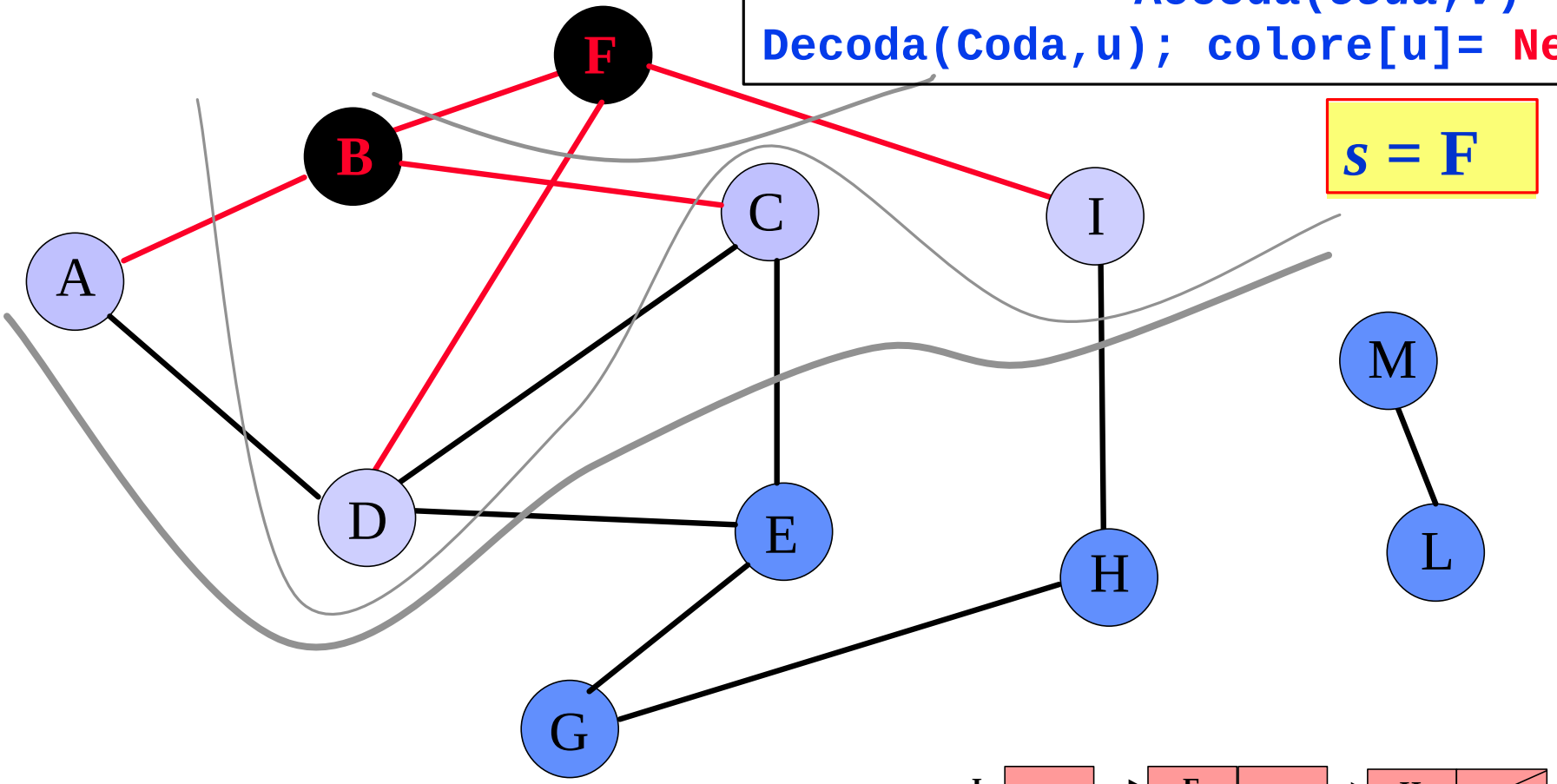
u = B

Coda:{B,I,D,C,A}



Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



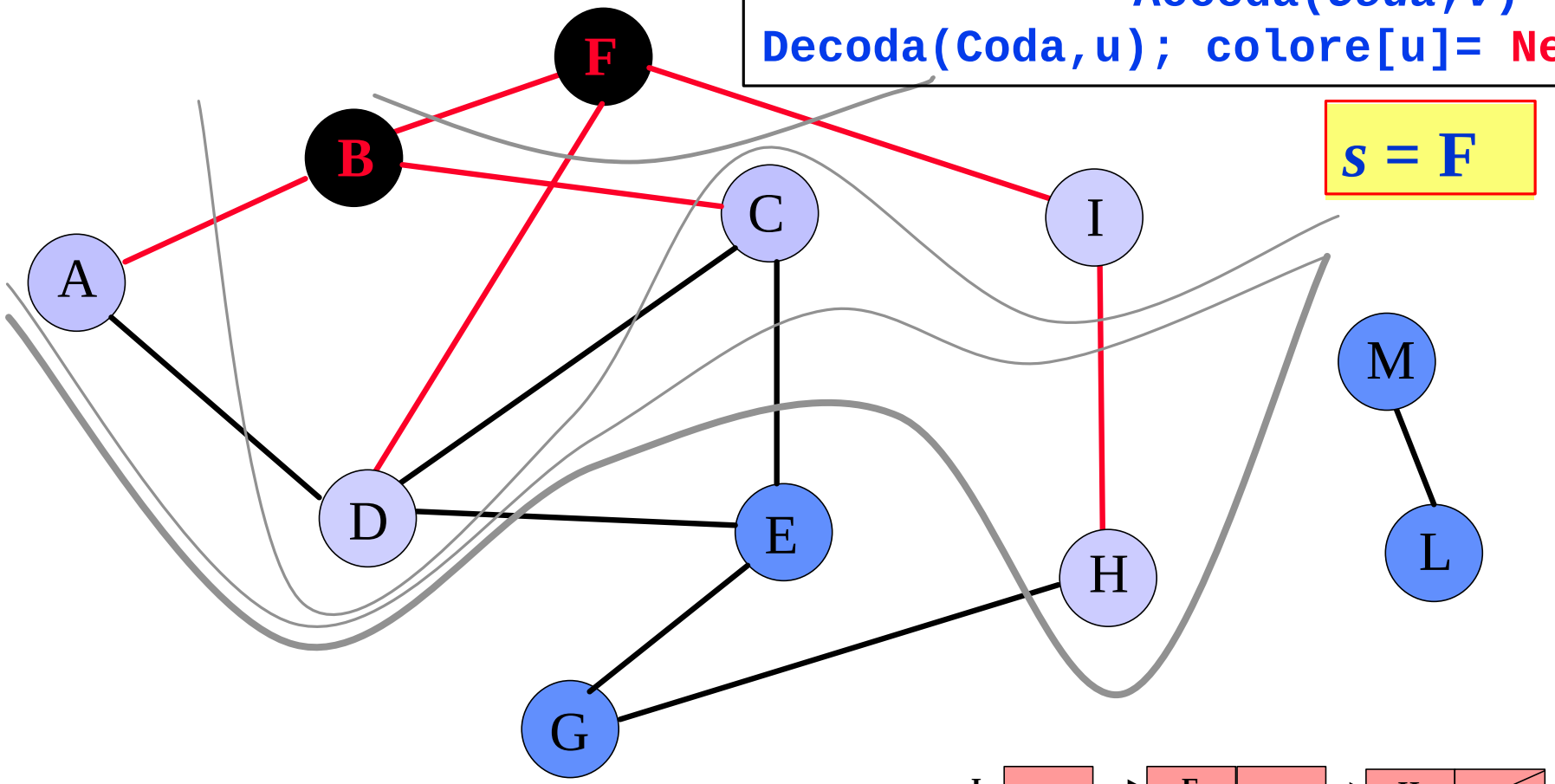
$s = F$

Coda:{I,D,C,A}

$u = I$

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



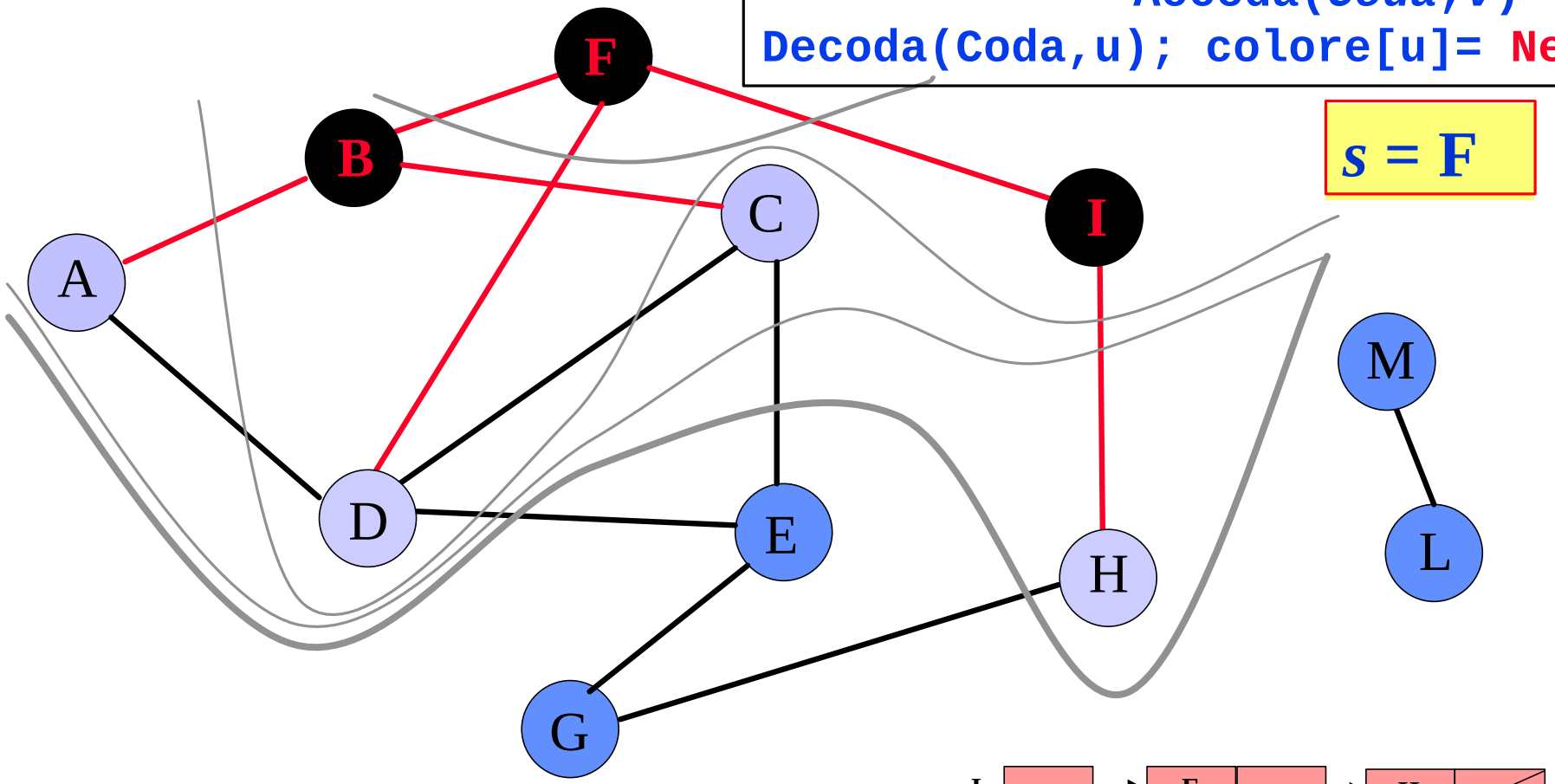
$s = F$

Coda:{I,D,C,A,H}

$u = I$

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



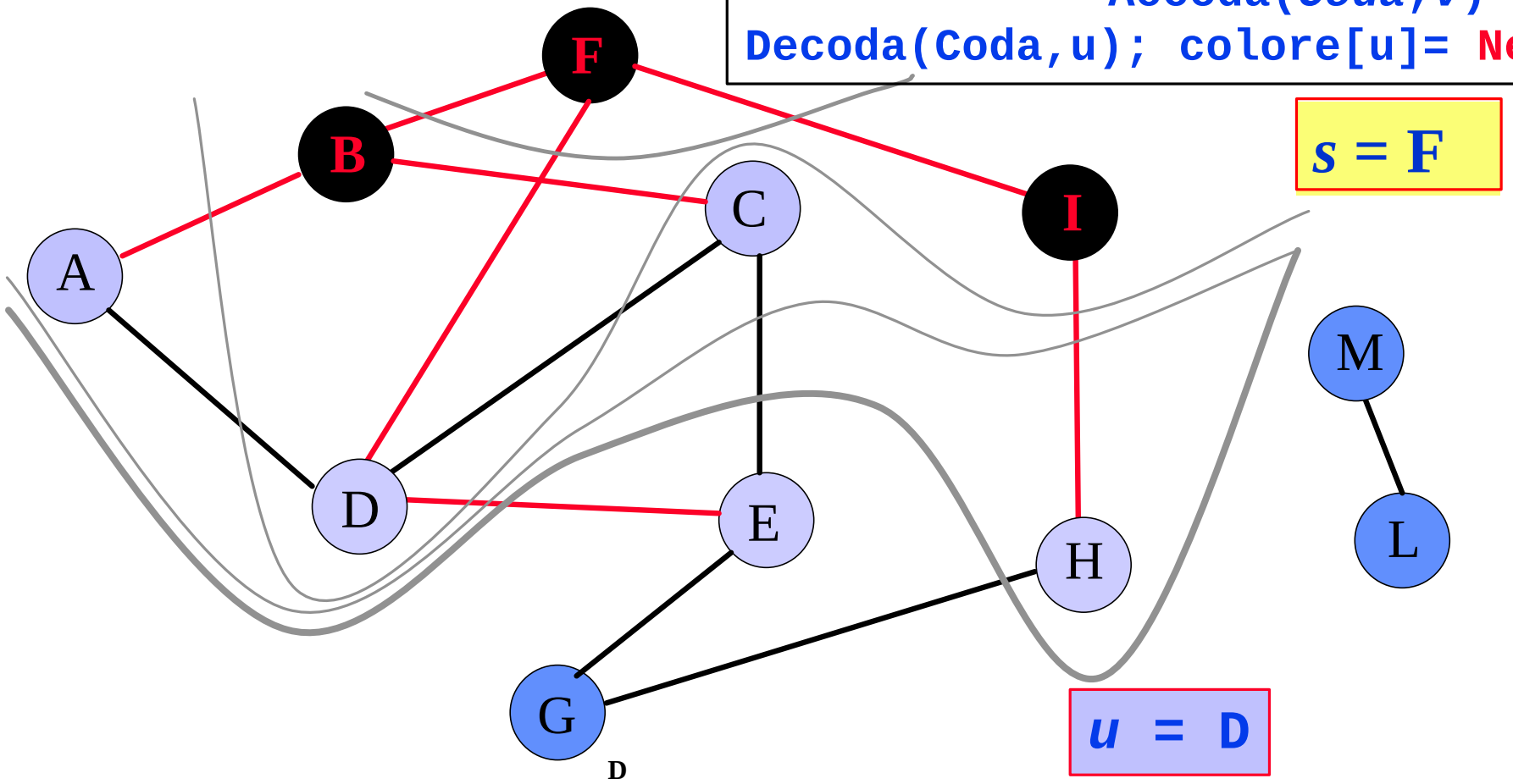
$s = F$

Coda:{D,I,C,A,H}

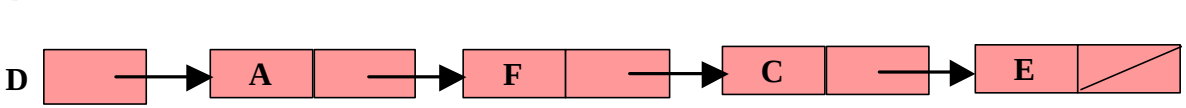
$u = I$

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```

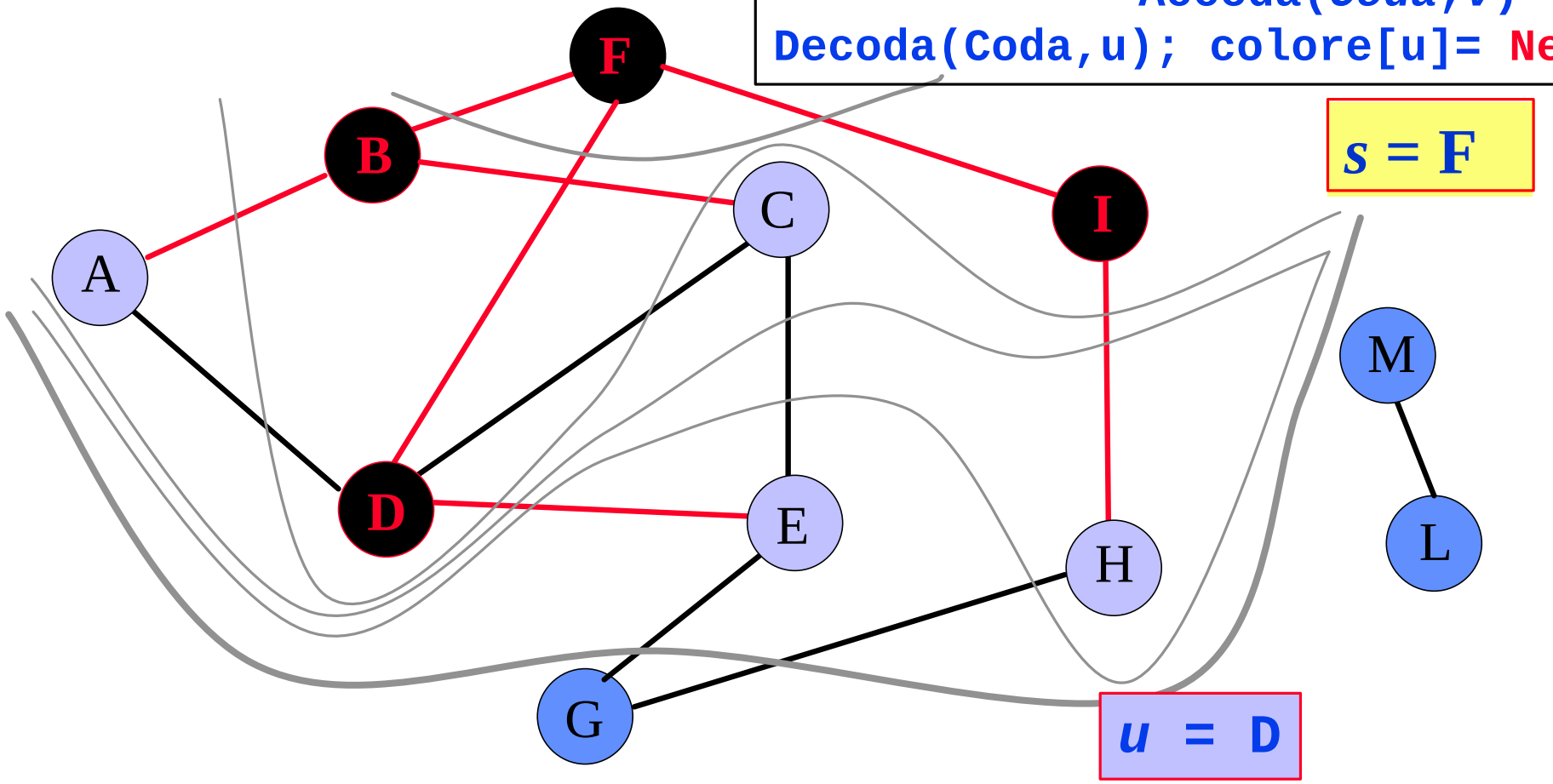


Coda:{D,C,A,H,E}

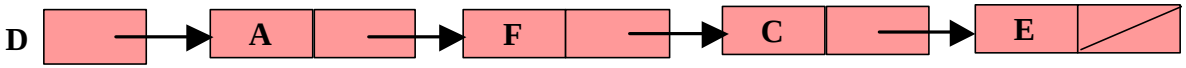


Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```

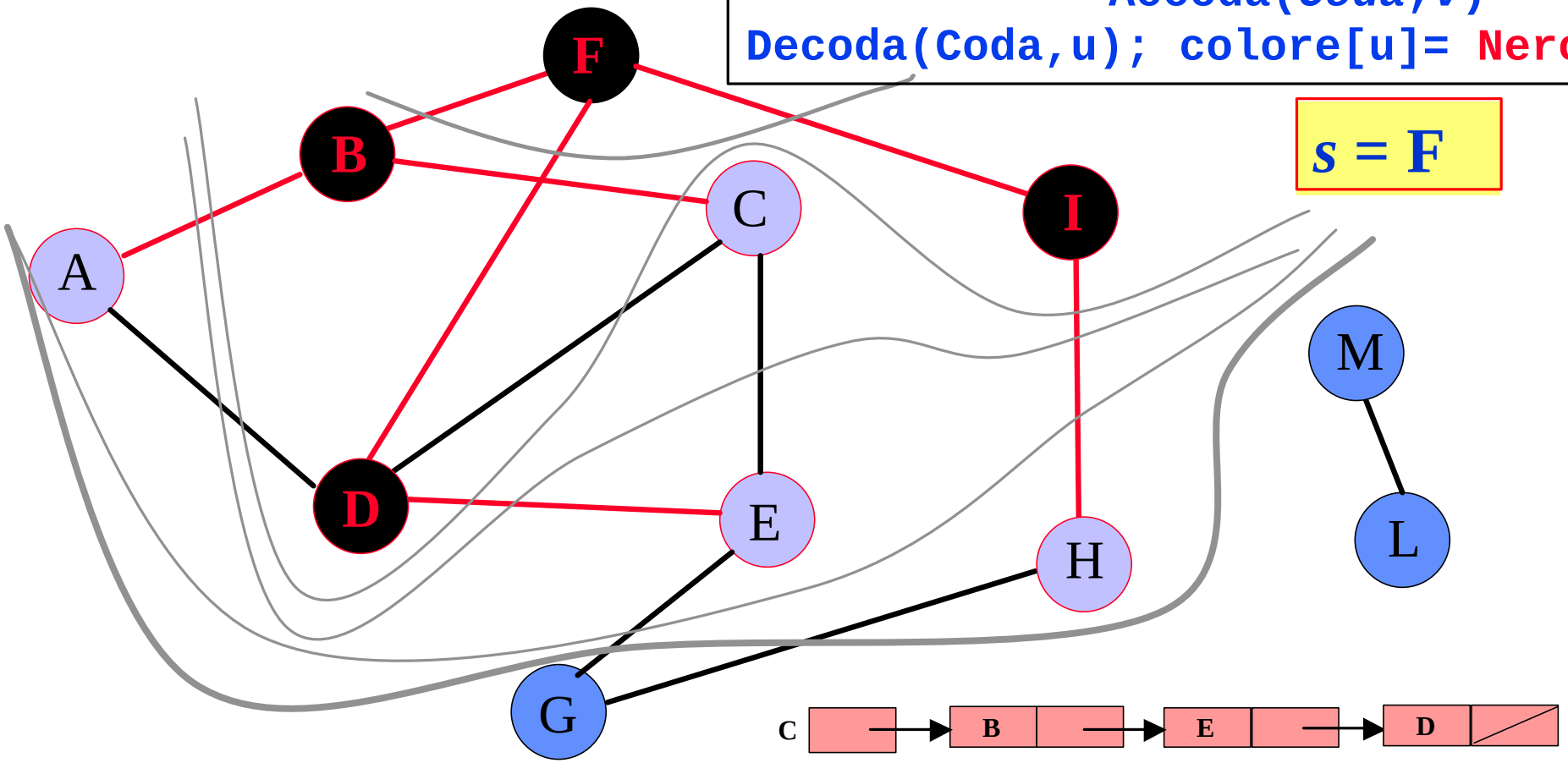


Coda:{D,C,A,H,E}



Algoritmo BFS II

```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



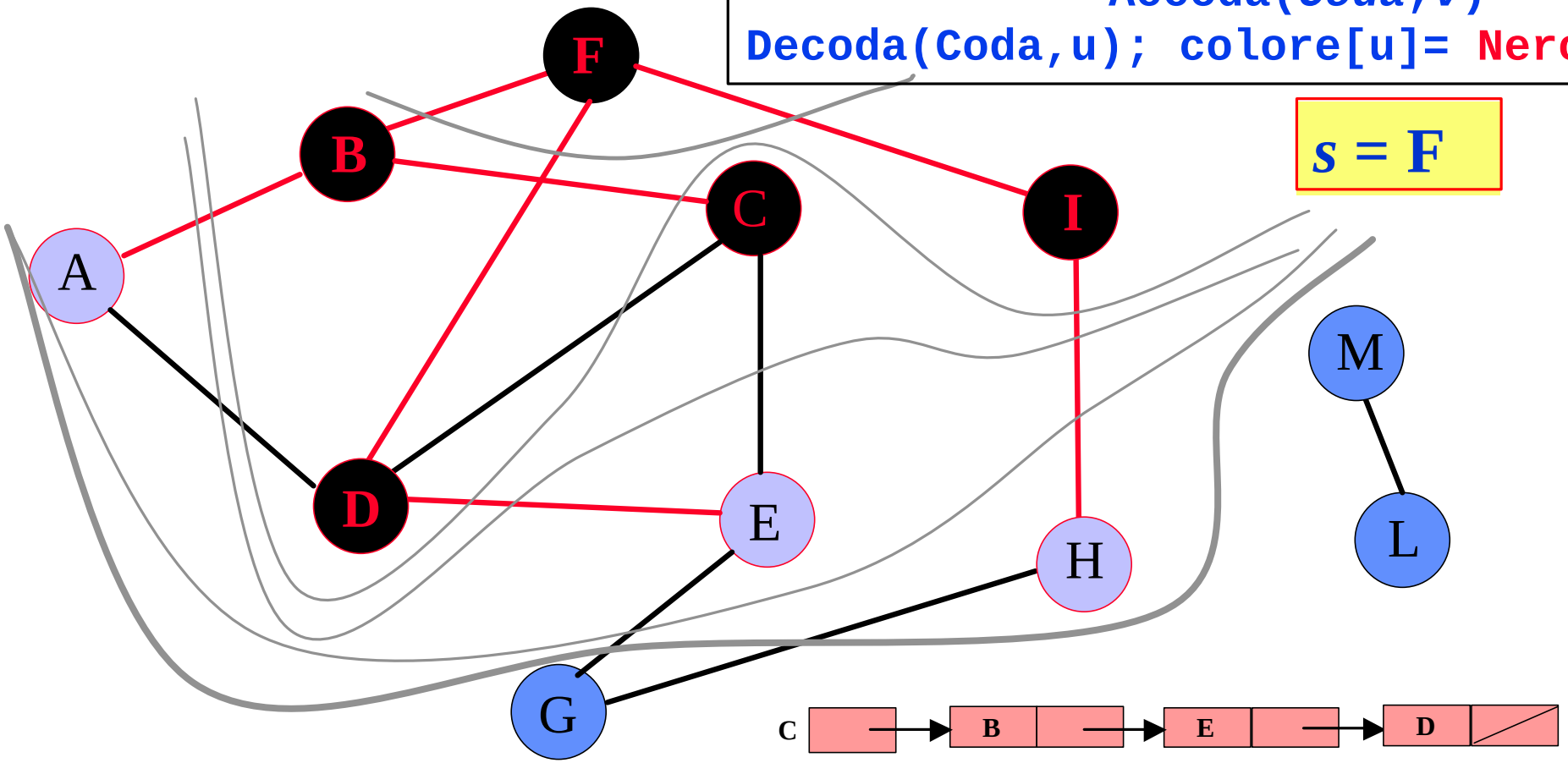
s = F

Coda:{C,A,H,E}

u = C

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



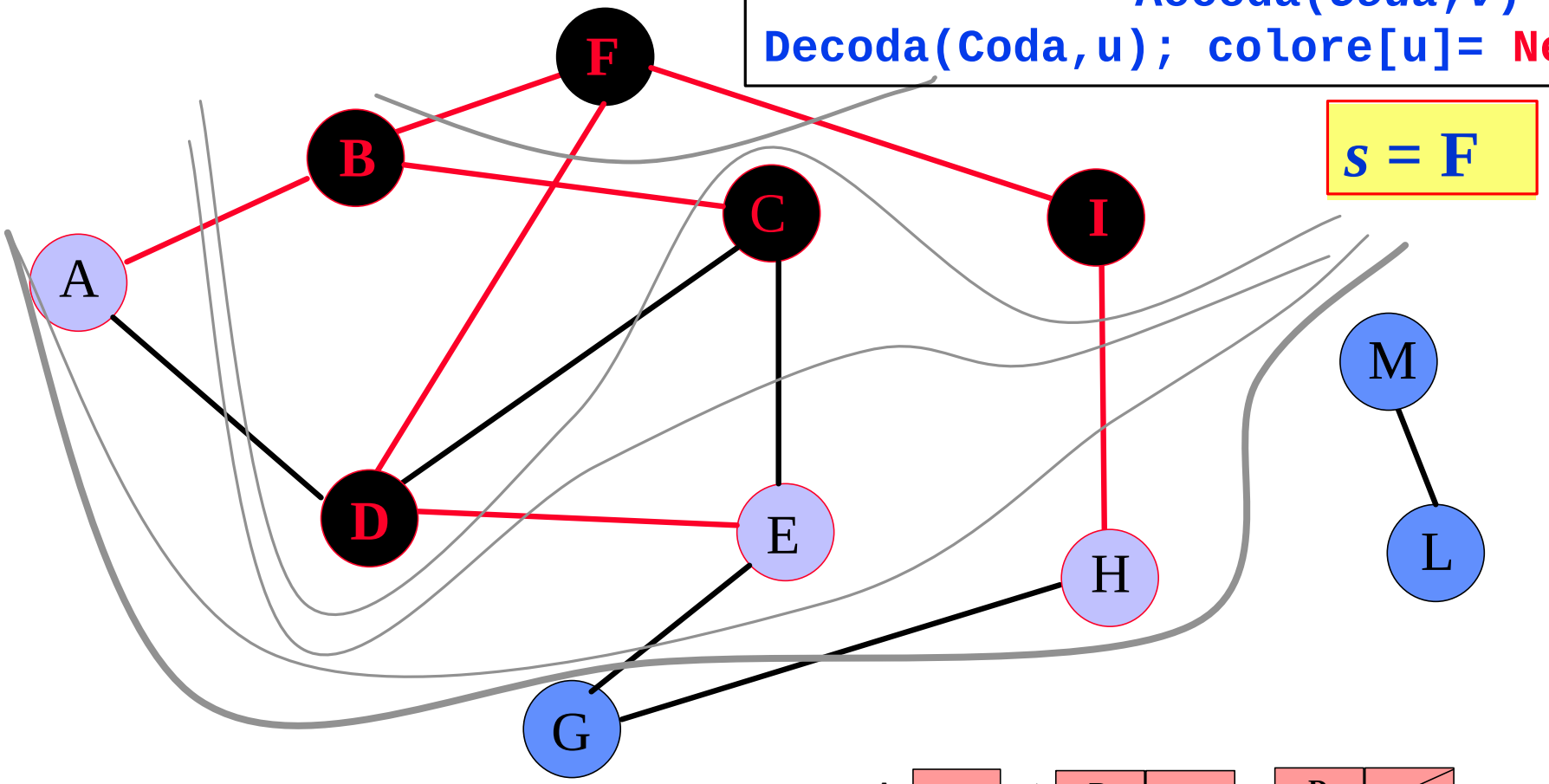
$s = F$

Coda:{C,A,H,E}

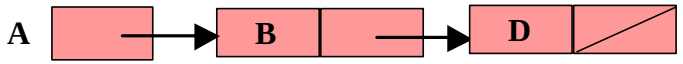
$u = C$

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



$s = F$

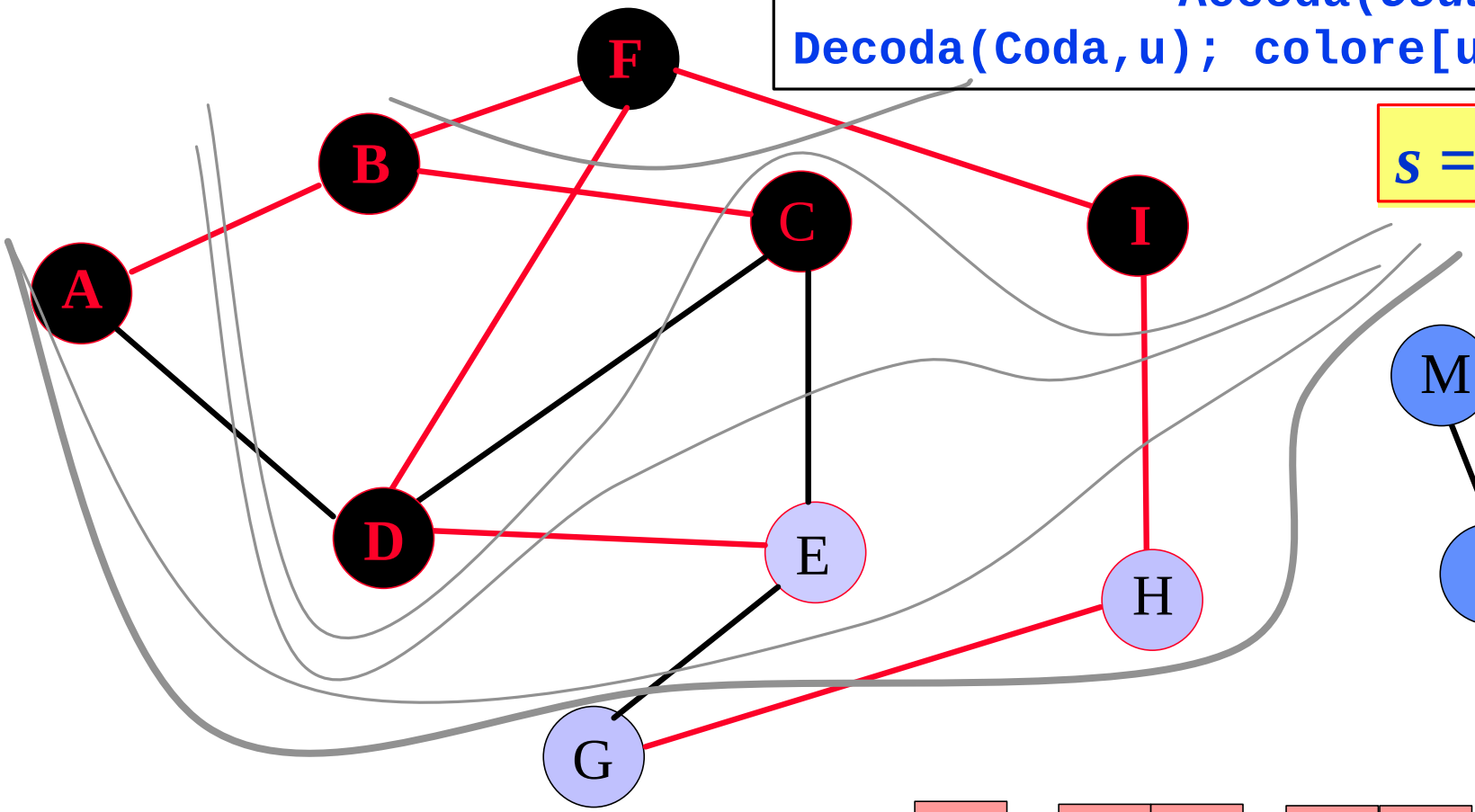


Coda:{A,H,E,G}

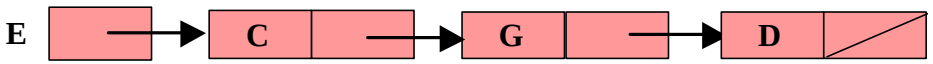
$u = A$

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



s = F

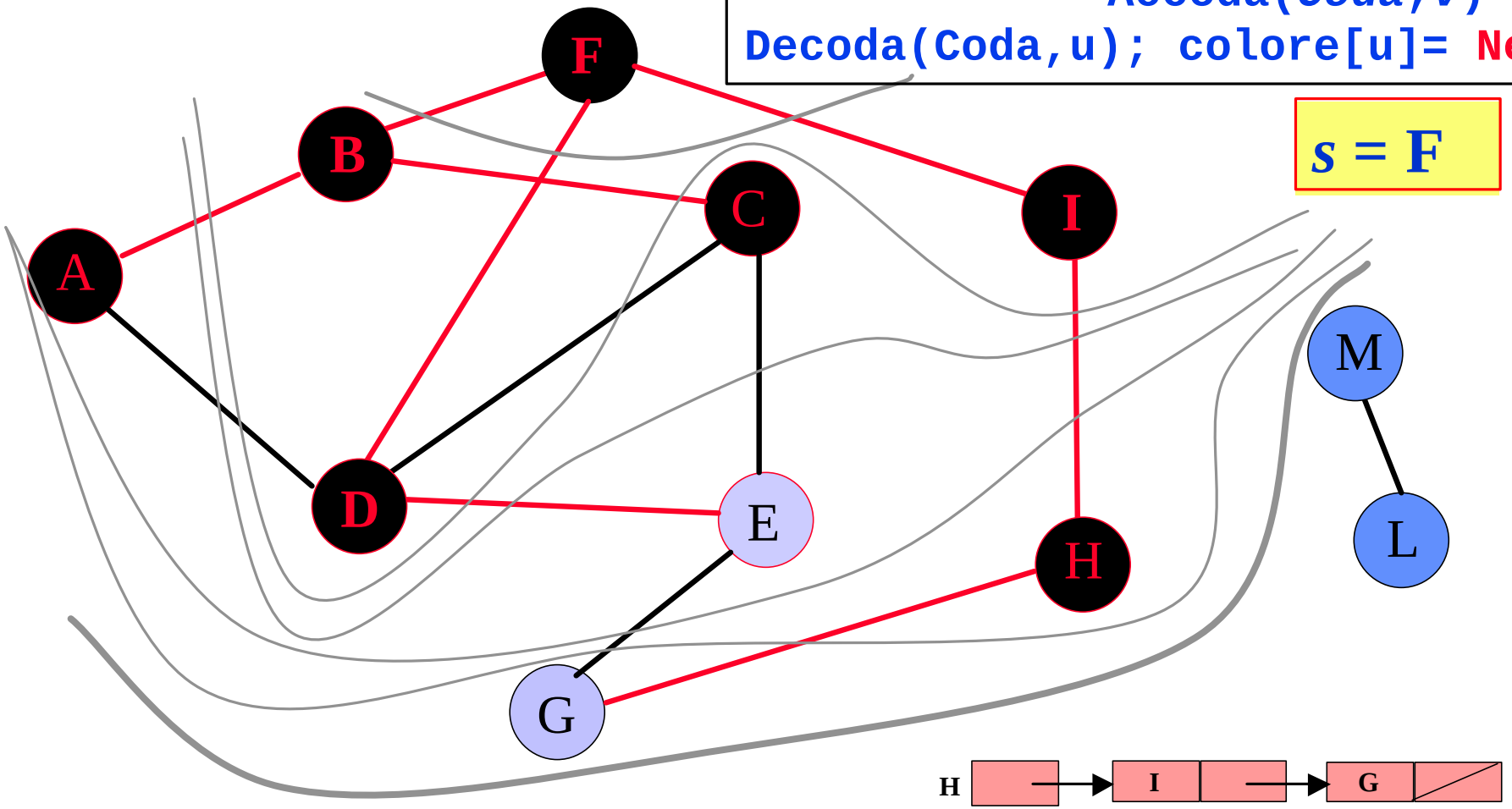


Coda:{H,E,G}

u = E

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



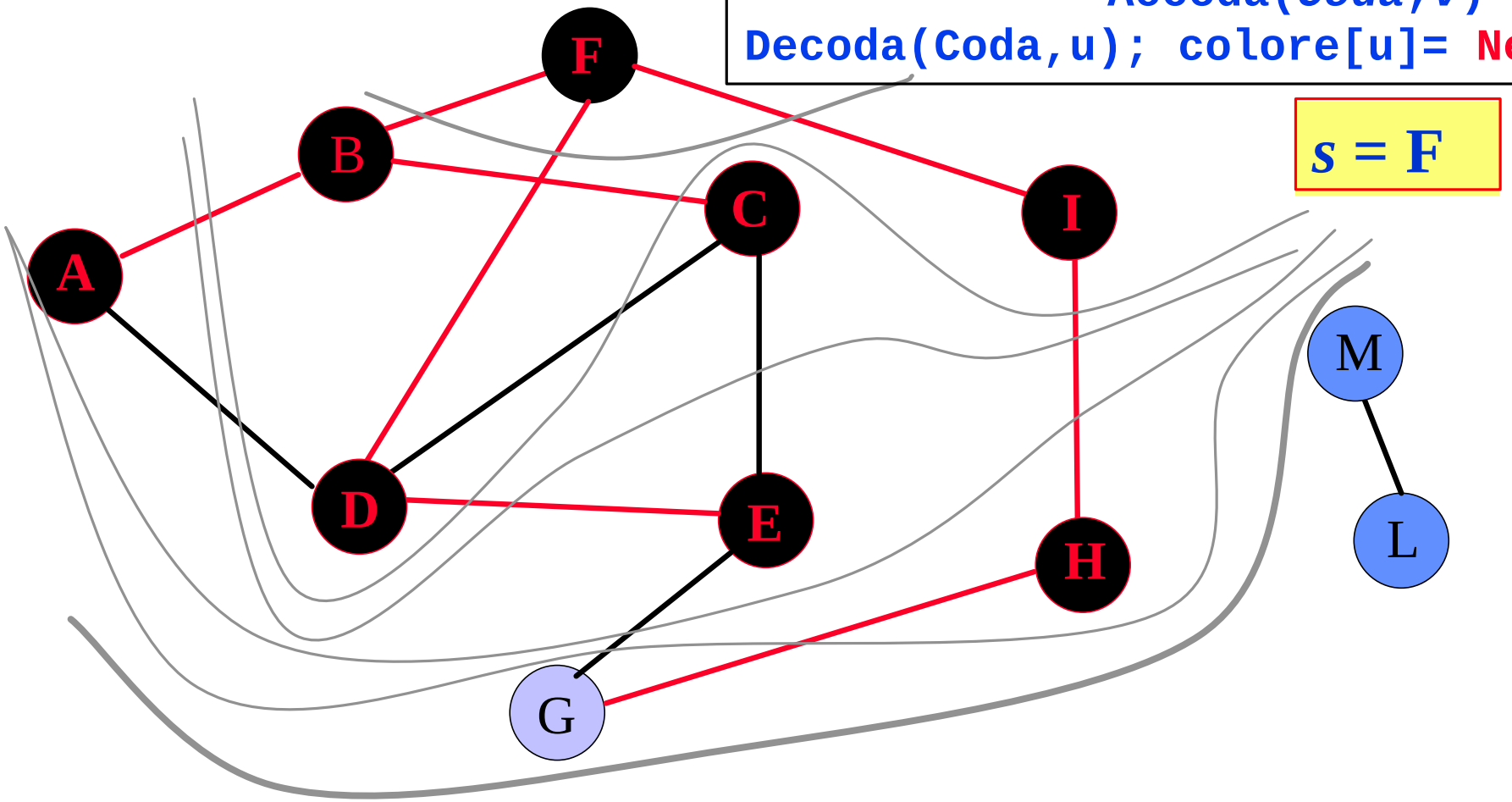
s = F

Coda:{E,G}

u = H

Algoritmo BFS II

```
for each v Î Adiac(u)
   do if colore[v] = Bianco
         then colore[v] = Grigio
              pred[v] = u
              Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```

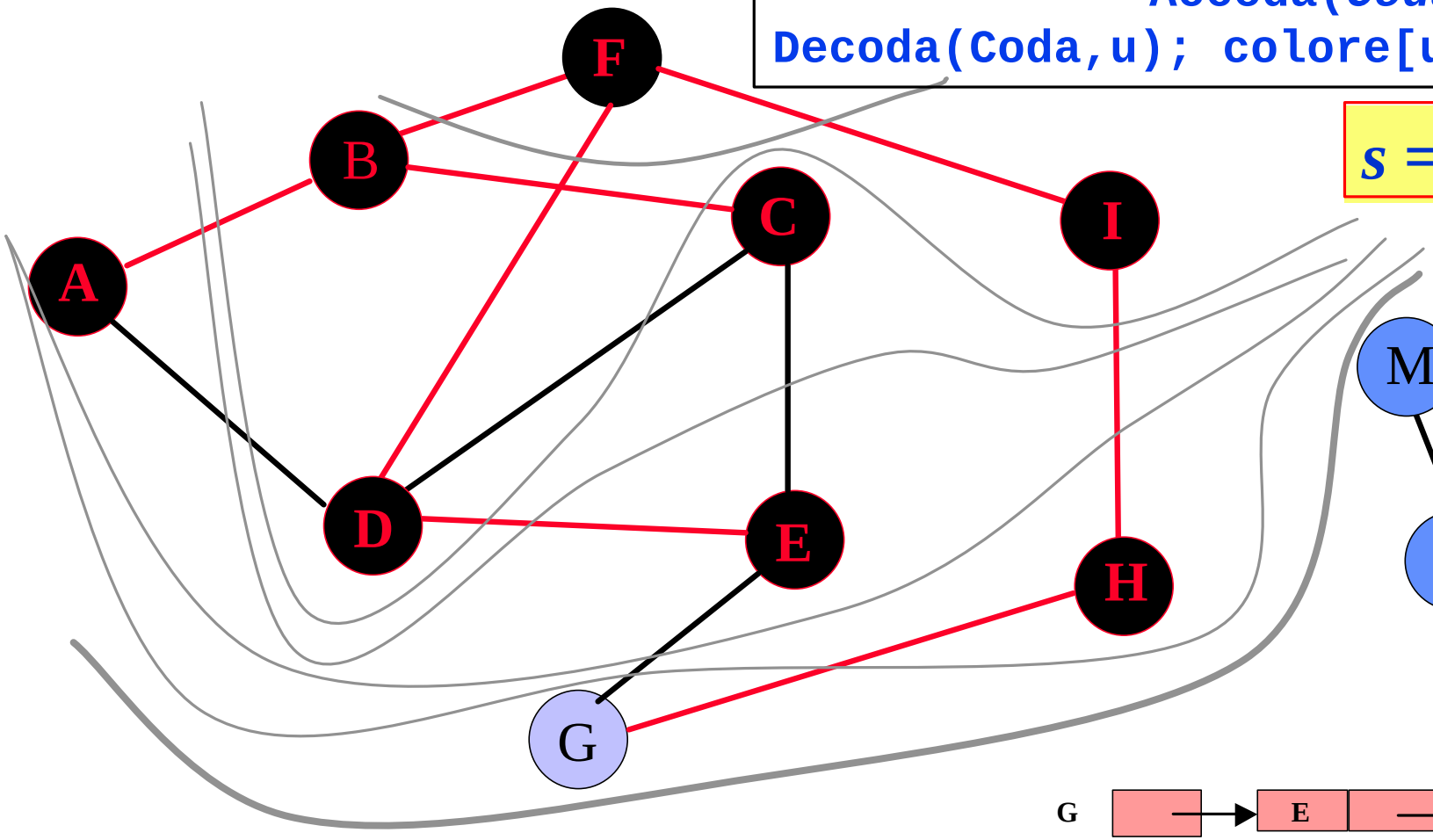


$s = F$

Coda:{G}

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



s = F

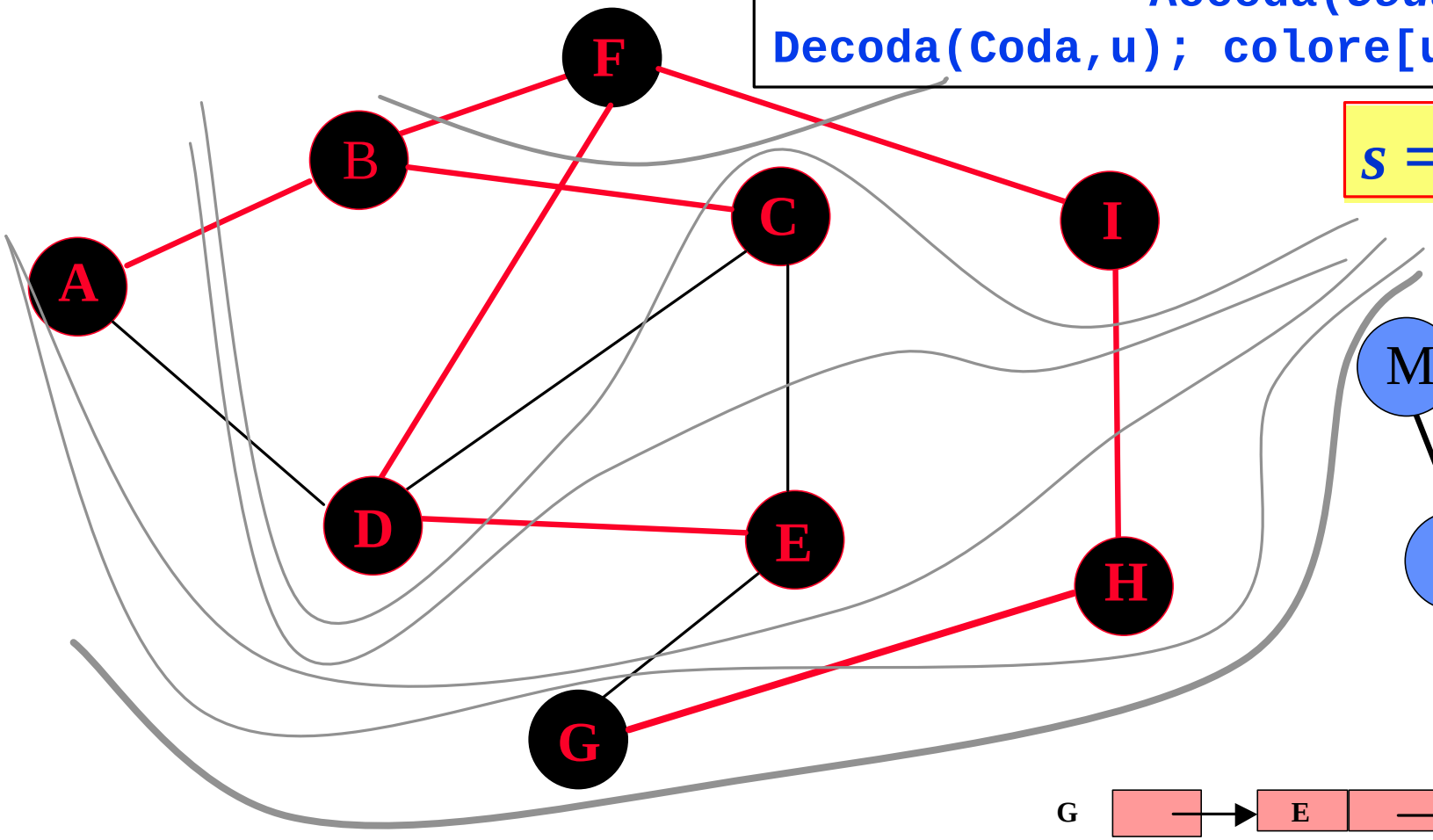
G → E → H

Coda:{G}

u = G

Algoritmo BFS II

```
for each v Î Adiac(u)
    do if colore[v] = Bianco
          then colore[v] = Grigio
               pred[v] = u
               Accoda(Coda,v)
Decoda(Coda,u); colore[u]= Nero
```



$s = F$

G → E → H

Coda:{}

$u = G$

Algoritmo BFS II

```
BSF(G:grafo, s:vertice)
  for each vertice u Î V(G) - {s}
     do colore[u] = Bianco
        pred[u] = Nil
  colore[s] = Grigio
  pred[s] = Nil
  Coda = {s}
  while Coda ¹ ∅
     do u = Testa[Coda]
        for each v Î Adiac(u)
           do if colore[v] = Bianco
                then colore[v] = Grigio
                     pred[v] = u
                     Accoda(Coda,v)
        Decoda(Coda)
        colore[u] = Nero
```

Inizializzazione

Accodamento dei soli nodi non visitati

Algoritmo BFS II: complessità

```
BSF(G:grafo, s:vertice)
  for each vertice u Î V(G) - {s}
     do colore[u] = Bianco
        pred[u] = Nil
  colore[s] = Grigio
  pred[s] = Nil
  Coda = {s}
  while Coda ¹ ∅
     do u = Testa[Coda]
        for each v Î Adiac(u)
           do if colore[v] = Bianco
                then colore[v] = Grigio
                     pred[v] = u
                     Accoda(Coda,v)
        Decoda(Coda)
        colore[u] = Nero
```

$O(|V|)$

$O(|E_u|)$

E_u = lunghezza della lista di adiacenza di u

Algoritmo BFS II: complessità

```
BSF(G:grafo, s:vertice)
  for each vertice u Î V(G) - {s}
     do colore[u] = Bianco
        pred[u] = Nil
  colore[s] = Grigio
  pred[s] = Nil
  Coda = {s}
  while Coda ¹ ∅
     do u = Testa[Coda]
        for each v Î Adiac(u)
            do if colore[v] = Bianco
                  then colore[v] = Grigio
                       pred[v] = u
                       Accoda(Coda,v)
        Decoda(Coda)
        colore[u] = Nero
```

$O(|V|)$

$O(|E|)$

E = dimensione delle liste di adiacenza. Numero di archi

Algoritmo BFS II: complessità

L'algoritmo di visita in *Breadth-First* impiega *tempo* proporzionale alla *somma* del *numero di vertici* e del *numero di archi* (dimensione delle liste di adiacenza).

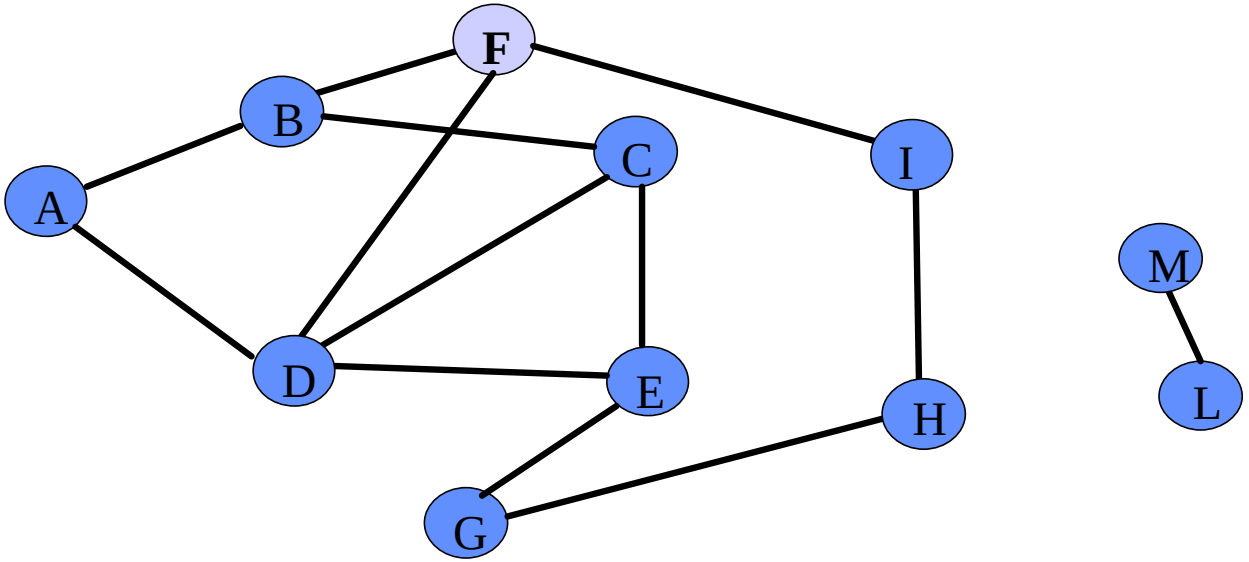
$$T(V,E) = O(|V|+|E|)$$

Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

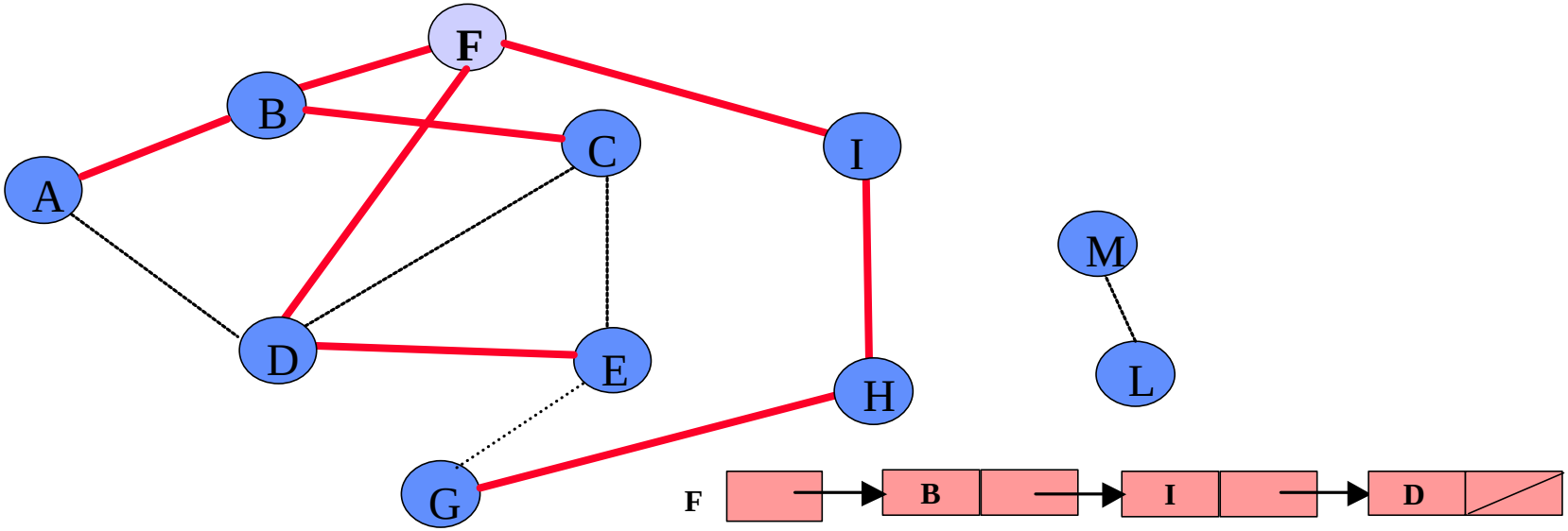


Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

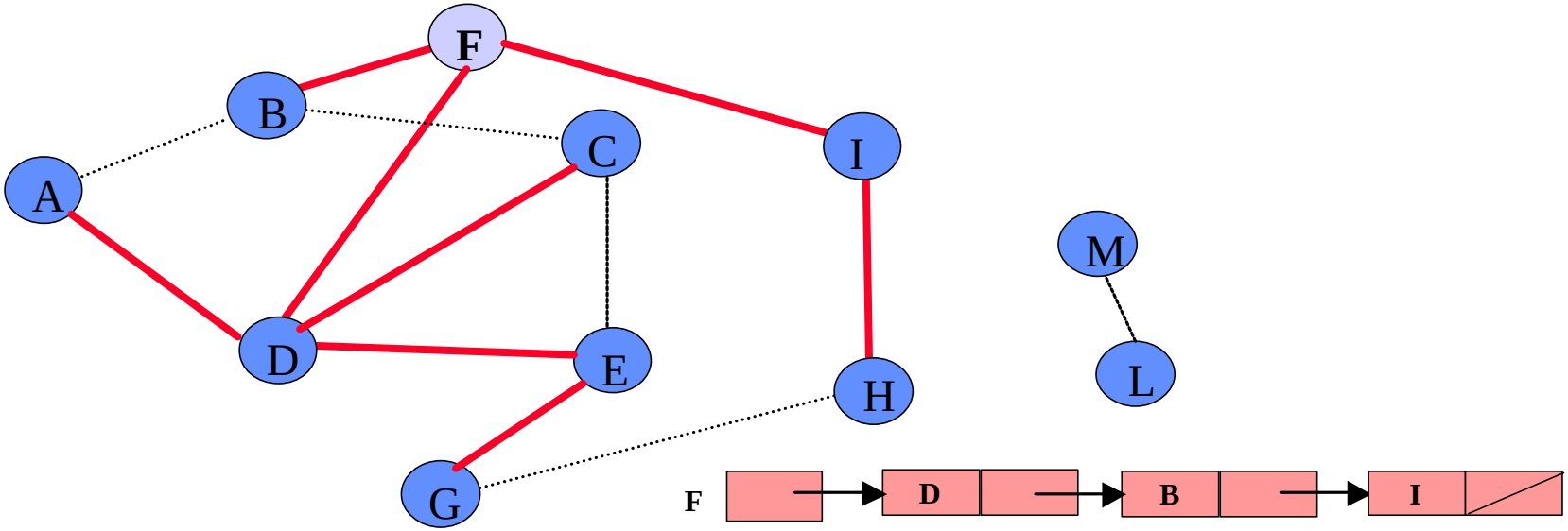


Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

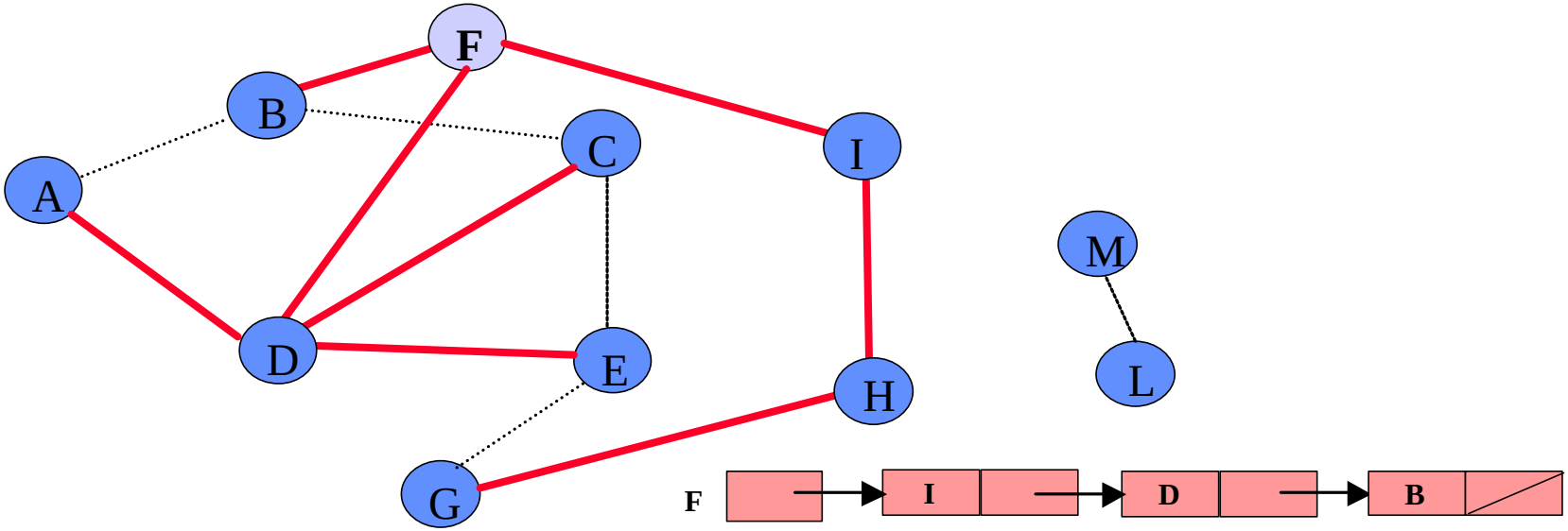


Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

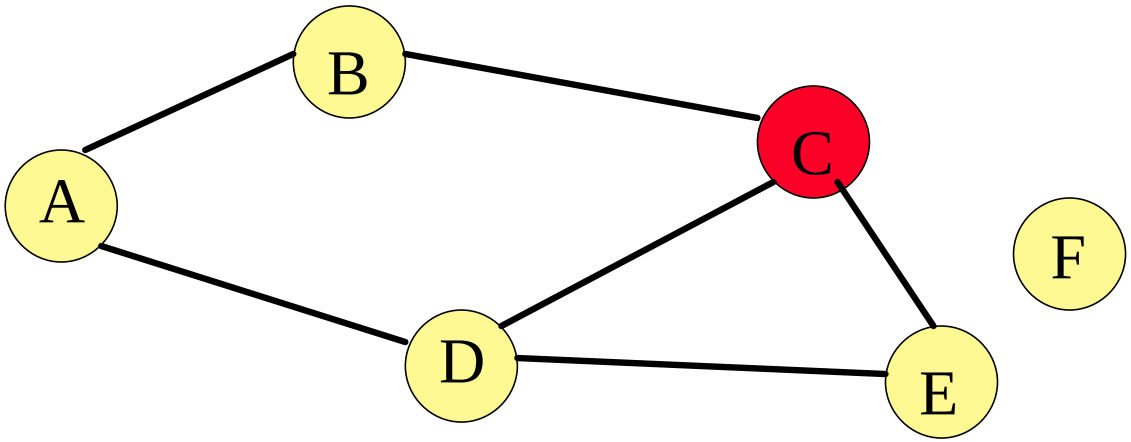


Definizione del problema

Attraversamento di un grafo

Dato un grafo *G*=<*V*,*E*> ed un vertice *s* di *V* (detto *sorgente*), esplorare *ogni vertice raggiungibile* nel grafo dal vertice *s*

Calcolare anche la *distanza* da *s* di tutti i vertici raggiungibili



***s* = C**

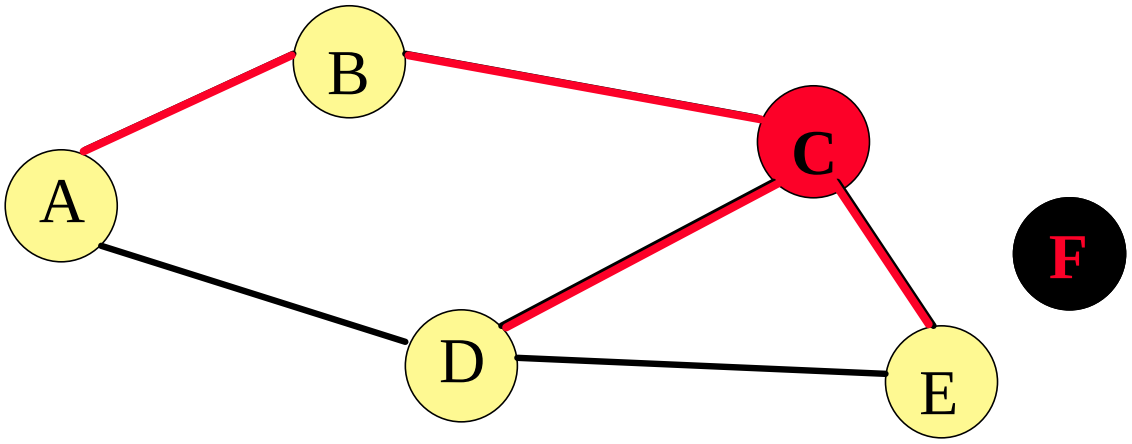
Numero minimo di archi tra i vertici

Definizione del problema

Attraversamento di un grafo

Dato un grafo *G*=<*V*,*E*> ed un vertice *s* di *V* (detto *sorgente*), esplorare *ogni vertice raggiungibile* nel grafo dal vertice *s*

Calcolare anche la *distanza* da *s* di tutti i vertici raggiungibili



***s* = C**

***dist*[*B*]=*dist*[*E*]=**
=*dist*[*D*]=1
***dist*[*A*]=2**
***dist*[*F*]=∞**

Algoritmo BFS III

```
BSF(G:grafo, s:vertice)
  for each vertice u Î V(G) - {s}
     do colore[u] = Bianco
        dist[u] = ∞
        pred[u] = Nil
  colore[s] = Grigio
  pred[s] = Nil
  dist[s] = 0
  Coda = {s}
  while Coda ¹ ∅
     do u = Testa[Coda]
        for each v Î Adiac(u)
            do if colore[v] = Bianco
                  then colore[v] = Grigio
                       dist[v] = dist[u] + 1
                       pred[v] = u
                       Accoda(Coda,v)
        Decoda(Coda)
        colore[u] = Nero S
```

Inizializzazione

Aggiorna-mento delle distanze

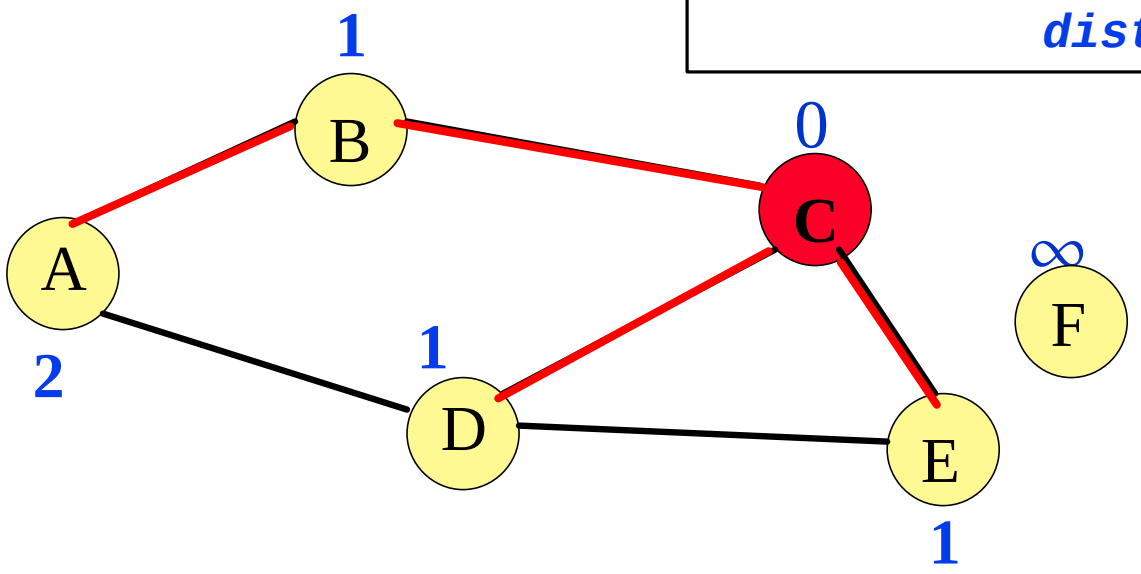
Algoritmo BFS III: calcolo distanze

Inizializzazione

```
for each vertice u ∈ V(G) - {s}
          do  ...
              dist[u] = ∞
              ...
  ...
  dist[s] = 0
```

Aggiornamento delle distanze

```
for each v ∈ Adiac(u)
    do if colore[v] = Bianco
       then ...
            dist[v]=dist[u]+1
```



Correttezza di BFS III

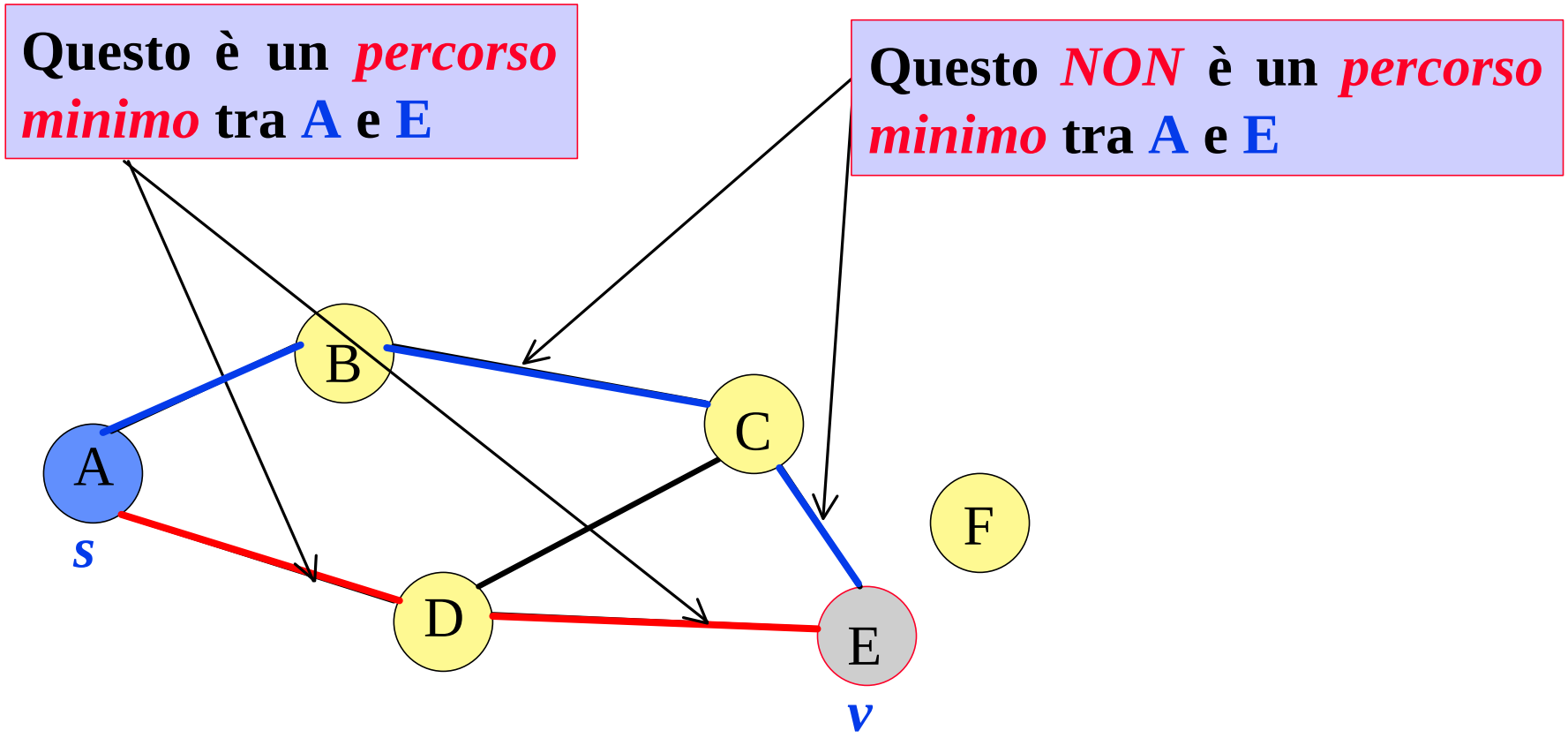
Sia dato un *grafo* $G=<V,E>$ (orientato o non) e un vertice sorgente s:

- **durante l'esecuzione dell'algoritmo *BFS(G,s)*, vengono esaminati *tutti i vertici* di V *raggiungibili* da s;**

- **Prima di dimostrare la correttezza di *BFS*, dimostreremo alcune proprietà dei *percorsi minimi*.**

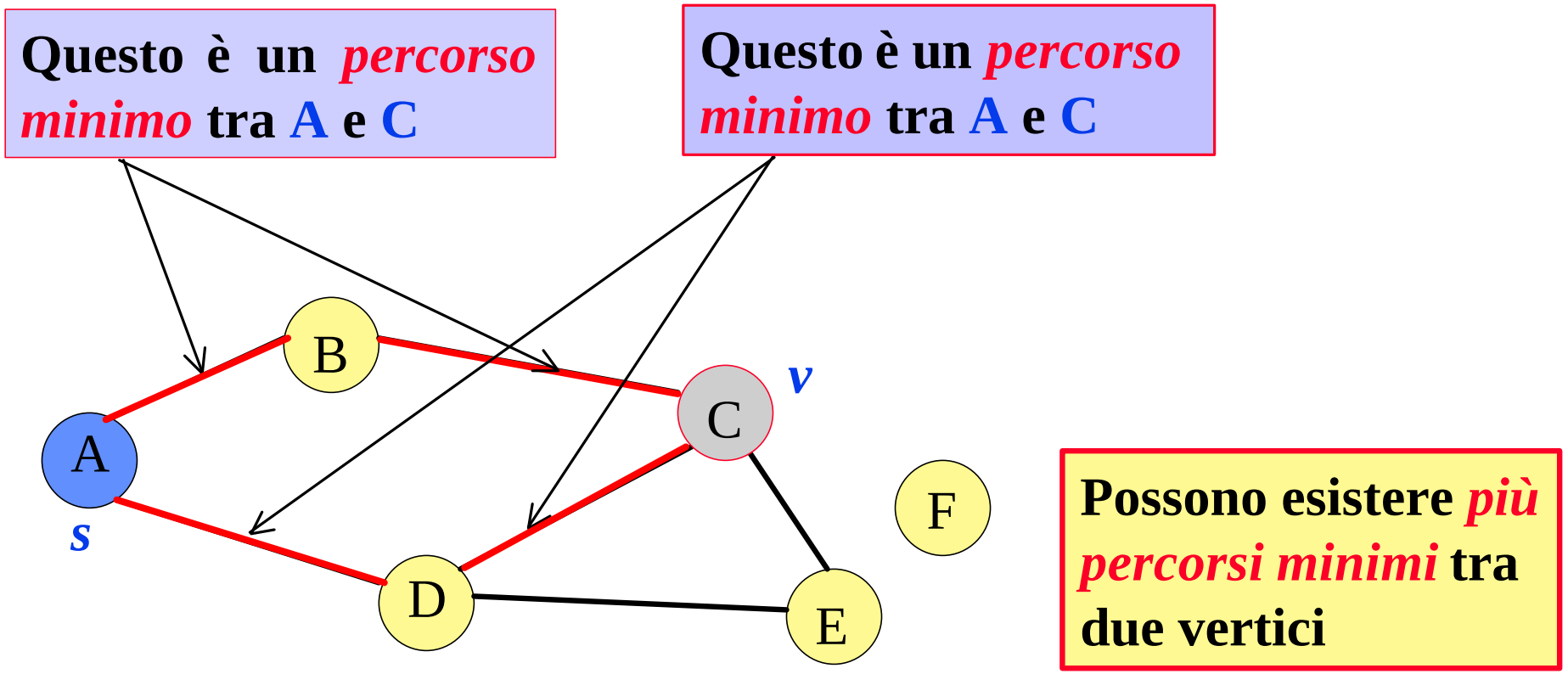
Percorsi minimi

Un *percorso minimo* in un *grafo* $G=<V,E>$ tra due vertici s e v è un percorso da s a v che contiene il minimo numero di archi.



Percorsi minimi

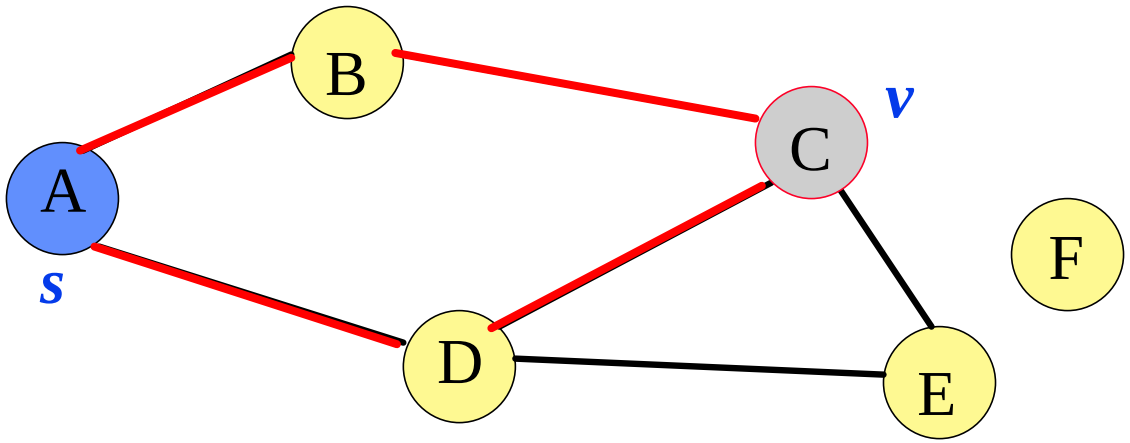
Un *percorso minimo* in un *grafo* $G=<V,E>$ tra due vertici s e v è un percorso da s a v che contiene il minimo numero di archi.



Percorsi minimi

Un *percorso minimo* in un *grafo* $G=<V,E>$ tra due vertici s e v è un percorso da s a v che contiene il minimo numero di archi.

La *distanza* $d(s,v)$ tra due vertici s e v è la *lunghezza* (numero di archi) di un *percorso minimo* tra s e v.



d(A,C) = 2

La *distanza minima* tra due vertici è unica

Percorsi minimi: proprietà I

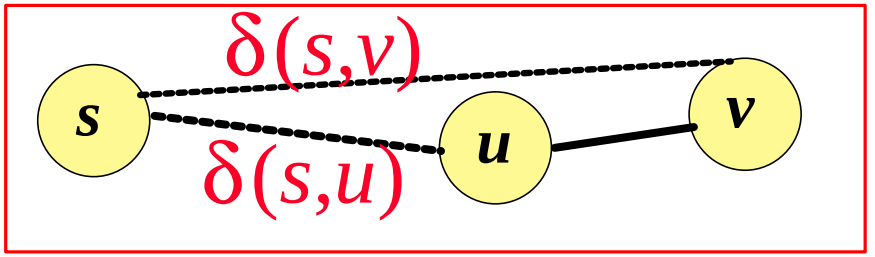
Sia *G=<V,E>* un *grafo* (orientato o non) e *s* un vertice di *G*. Allora per ogni arco *(u,v)* di *E*, vale quanto segue:

$$d(s,v) \leq d(s,u) + 1$$

Proprietà I: dimostrazione

Ci sono 2 casi:

- ***u* è raggiungibile da *s***
- ***u* non è raggiungibile da *s***



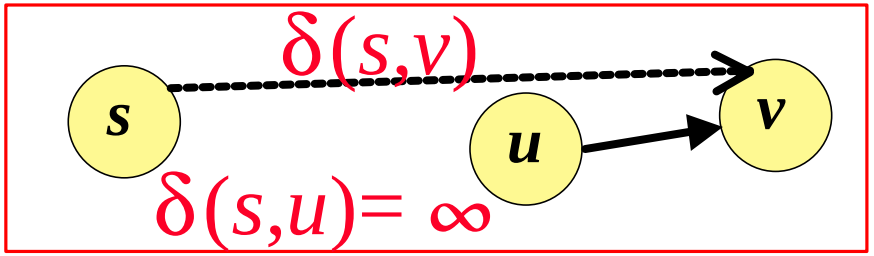
- ***u* è raggiungibile da *s*:**

  Allora anche *v* lo è.

  Il *percorso minore* da *s* a *v* in tal caso *non* può essere *più lungo* del percorso minore da *s* a *u* seguito dall'arco (*v*,*u*), e quindi d (*s*,*v*) £ d (*s*,*u*) + 1

- ***u* non è raggiungibile da *s***

  Allora d (*s*,*u*)=∞, e nuovamente la disuguaglianza vale.



Percorsi minimi: proprietà II

Sia *G=<V,E>* un *grafo* (orientato o non). Supponiamo di eseguire *BFS(G,s)*. Al termine dell'algoritmo, per ogni vertice *v* di *V*, vale:

$$dist[v] \geq d(s,v)$$

Proprietà II: dimostrazione

Induzione sul *numero di inserimenti* di vertici nella coda, con la seguente *Ipotesi Induttiva*: "per ogni accodamento precedente a quello corrente, *dist*[*v*] ³ d(*s*,*v*) per tutti i vertici *v*".

***Passo Base*: è quando *s* viene posto nella coda. Poiché *dist*[*s*] = 0 = d(*s*,*s*) e *dist*[*v*] = ∞ ³ d(*s*,*v*) per ogni altro vertice *v*, la tesi è verificata!**

```
BSF(G:grafo, s:vertice)
  for each vertice u Î V(G) - {s}
     do  colore[u] = Bianco
         dist[u] = ∞
         pred[u] = Nil
  dist[s] = 0
  Coda = {s}
  ...
```

Proprietà II: dimostrazione

***Passo Induttivo*: un vertice bianco *v* viene posto in coda scorrendo la lista di adiacenza di un vertice *u*. Per ipotesi induttiva *dist*[*u*] ³ d(*s*,*u*).**

Dall'assegnamento e dalla *proprietà I* risulta che:

$$dist[v] = dist[u] + 1$$
$$\geq d(s,u) + 1$$
$$\geq d(s,v)$$

```
BSF(G:grafo, s:vertice)
  …
while Coda ¹ ∅
   do u = Testa[Coda]
      for each v Î Adiac(u)
        do if colore[v] = Bianco
           then colore[v] = Grigio
                dist[v] = dist[u] + 1
                Accoda(Coda,v)
      Decoda(Coda)
```

Percorsi minimi: proprietà III

Sia $G=<V,E>$ **un** ***grafo*** **(orientato o non). Supponiamo di eseguire** $BFS(G,s)$ **e che in** ***coda*** **siano presenti i vertici** $[v_1,\ldots,v_n]$ **(**v_1 **è la testa). Allora:**

$$dist[v_n] \leq dist[v_1] + 1$$

$$dist[v_i] \leq dist[v_{i+1}]$$

per ogni $i=1,\ldots,n-1$

Proprietà III: dimostrazione

Dimostriamo per induzione sul numero di operazioni sulla coda.

- ***Passo Base*: Inizialmente (*1 operazione sulla coda*), quando la coda contiene solo *s*, la proprietà certamente vale.**

Proprietà III: dimostrazione

Ipotesi Induttiva

Dobbiamo dimostrare che la proprietà vale sia per le operazioni di accodamento che di estrazione dalla coda di un vertice.

Denotiamo con $[v_1\ v_2\ \ldots\ v_r]$ coda, dove v_1 è la testa.

Supponiamo (ipotesi induttiva) che la proprietà valga alla (k-1)-esima operazione sulla coda, che sarà $[v_1\ v_2\ \ldots\ v_r]$, cioè:

$$dist[v_r] \leq dist[v_1] + 1$$

$$dist[v_i] \leq dist[v_{i+1}]$$

Proprietà III: dimostrazione

- ***Passo Induttivo** consideriamo la **k**-esima operazione*

  1) **quando v_1 viene estratto, v_2 diventa la nuova testa (quando si svuota la proprietà vale banalmente). Allora, poiché si ha $dist[v_1] \leq dist[v_2]$, risulta**

  $$dist[v_r] \leq dist[v_1]+1 \leq dist[v_2]+1$$

  e il resto delle disuguaglianze resta identico.

  Quindi la proprietà vale con v_2 come testa

Proprietà III: dimostrazione

- ***Passo Induttivo** consideriamo la k-esima operazione*

  2) **quando si accoda a $[v_1\ v_2\ \ldots\ v_r]$ il vertice v (nel codice) diventa il nuovo v_{r+1}, $[v_1\ v_2\ \ldots\ v_r\ v_{r+1}]$, mentre il vertice v_1 è il vertice u la cui lista di adiacenza viene esaminata (nel codice). Allora vale $dist[v_{r+1}] = dist[v] £ dist[u]+1 = dist[v_1]+1$ e**

  $dist[v_r] £ dist[v_1]+1 = dist[u]+1 = dist[v] = dist[v_{r+1}]$

Le altre uguaglianze restano invariate…

… e la proprietà vale!

```
u = Testa[Coda]
for each v Î Adiac(u)
    do if colore[v] = Bianco
        then dist[v] = dist[u] + 1
             Accoda(Coda,v)
Decoda(Coda)
```

Correttezza di BFS III

Sia dato un *grafo* $G=<V,E>$ (orientato o non) e un vertice sorgente s:

- **durante l'esecuzione dell'algoritmo *BFS(G,s)*, vengono esaminati *tutti i vertici* di V *raggiungibili* da s;**
- **al termine $dist[v] = \delta(s,v)$ per ogni $v \in V$;**
- **se $v \neq s$, *uno dei percorsi minimi* tra s e v è il *percorso minimo* da s a $pred[v]$ seguito dall'arco $(pred[v],v)$.**

Correttezza di BFS III: dimostrazione

Dimostrazione: **consideriamo il caso in cui il vertice v sia raggiungibile da s (vedere sul libro di testo il caso in cui v non è raggiungibile).**

- **Sia V_k l'insieme dei vertici a distanza (minima) k da s (cioè $V_k=\{v \in V: d(s,v) = k\}$).**
- **La dimostrazione procede per *induzione su* k, cioè sulla distanza di un nodo v da s.**

Ipotesi induttiva: **per ogni $j < k$, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:**

- **colorata v di *grigio***
- **assegna *a* $dist[v]$ il valore j**
- **se $v \neq s$, allora assegna a $pred[v]$ il valore u, per qualche $u \in V_{j-1}$**
- **inserisce v nella coda**

Correttezza di BFS III: dimostrazione

Caso Base: **Per $k=0$, $V_0=\{s\}$ (unico vertice a distanza 0 da s):**

- **l'inizializzazione colora s di grigio;**
- ***dist*[*s*] viene posto a 0;**
- ***s* è messo nella coda.**

Quindi l'ipotesi induttiva vale!

Ipotesi induttiva: **per ogni $j < k$, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:**

- **colorata *v* di *grigio***
- **assegna *a dist*[*v*] il valore *j***
- **se $v \neq s$, allora assegna a *pred*[*v*] il valore *u*, per qualche $u \in V_{j-1}$**
- **inserisce *v* nella coda**

Correttezza di BFS III: dimostrazione

Caso induttivo:

- **La coda non è mai vuota fino al termine.**
- **Una volta inserito un vertice *u* nella coda, né *dist*[*u*] né *pred*[*u*] cambiano il loro valore.**
- **Per il teorema precedente, se i vertici sono inseriti nell'ordine $v_1, v_2, \ldots, v_r$, la sequenza delle distanze è crescente monotonicamente ($d[v_i] \leq d[v_{i+1}]$)**

Ipotesi induttiva: **per ogni j < k, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:**

- **colorata *v* di *grigio***
- **assegna *a dist*[*v*] il valore *j***
- **se $v \neq s$, allora assegna a *pred*[*v*] il valore *u*, per qualche $u \in V_{j-1}$**
- **inserisce *v* nella coda**

Correttezza di BFS III: dimostrazione

Caso induttivo:

- Sia ora $v \in V_k$ ($k \geq 1$).
- Dalla proprietà di monotonicità, dal fatto che $dist[v] \geq k$ (per un teorema precedete) e dall'ipotesi induttiva, segue che v (se viene visitato) deve essere visitato dopo che tutti i vertici nell'insieme V_{k-1} sono stati accodati.
- Poiché $d(s,v)=k$, c'è un percorso di $k-1$ archi da s a u, e quindi anche un vertice $u \in V_{k-1}$ tale che $(u,v) \in E$, cioè v adiacente a u.

Ipotesi induttiva: per ogni $j < k$, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:

- colorata v di *grigio*
- assegna *a* $dist[v]$ il valore j
- se $v \neq s$, allora assegna a $pred[v]$ il valore u, per qualche $u \in V_{j-1}$
- inserisce v nella coda

Correttezza di BFS III: dimostrazione

Caso induttivo:

- **Supponiamo che *u* sia il primo dei vertici a cui *v* è adiacente che viene colorato di grigio (per induzione tutti i vertici in V_{k-1} sono grigi).**
- **As un certo momento, verrò *esaminata la lista di adiacenza* di *u* e *v* viene scoperto (ciò non accade prima perché *v* sta in V_k e non è adiacente a vertici in V_j e con $j < k-1$, e *u* è il primo adiacente per l'ipotesi in alto)**

Ipotesi induttiva: **per ogni $j < k$, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:**

- **colorata *v* di *grigio***
- **assegna *a dist*[*v*] il valore *j***
- **se $v \neq s$, allora assegna a *pred*[*v*] il valore *u*, per qualche $u \in V_{j-1}$**
- **inserisce *v* nella coda**

Correttezza di BFS III: dimostrazione

Caso induttivo:

- **Quindi *v* viene colorato di grigio da BFS.**
- **Viene assegnato *dist*[*v*]=*dist*[*u*]+1= (k-1) + 1=*k***
- **Viene eseguito *pred*[*v*]=*u* e sappiamo (per ipotesi induttiva) che $u \in V_{k-1}$**
- **Viene messo *v* in coda.**

Essendo *v* un vertice arbitrario in V_k, l'ipotesi induttiva è dimostrata per *k*!

Ipotesi induttiva: **per ogni j < k, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:**

- **colorata *v* di *grigio***
- **assegna *a dist*[*v*] il valore *j***
- **se $v \neq s$, allora assegna a *pred*[*v*] il valore *u*, per qualche $u \in V_{j-1}$**
- **inserisce *v* nella coda**

Correttezza di BFS III: dimostrazione

Quindi, se $v \in V_k$, allora certamente $pred[v] \in V_{k-1}$

È quindi possibile ottenere il *percorso minimo da s a v* edstendendo il percorso minimo da *s* a $pred[v]$ con l'arco $(pred[v],v)$.

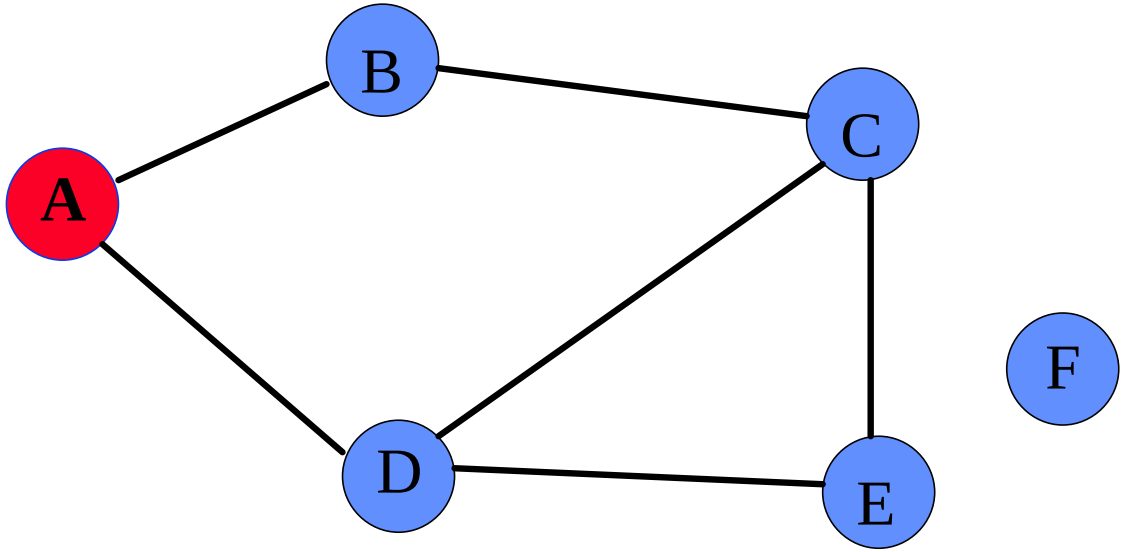
Ipotesi induttiva: **per ogni $j < k$, per ogni $v \in V_j$, c'è solo un istante in cui l'algoritmo di BFS:**

- **colorata *v* di *grigio***
- **assegna *a* $dist[v]$ il valore *j***
- **se $v \neq s$, allora assegna a $pred[v]$ il valore *u*, per qualche $u \in V_{j-1}$**
- **inserisce *v* nella coda**

Alberi breadth-first

Un ***albero breadth-first*** di un grafo non orientato ***G =<V,E>*** con ***sorgente s***, è un ***albero libero G'=<V',E'>*** tale che:

- ***G'*** è un ***sottografo del grafo non orientato sottostante*** ***G***
- ***v*** Î ***V'*** se e solo se ***v*** è raggiungibile da ***s***
- per ogni ***v*** Î ***V'***, il percorso da ***s*** a ***v*** è ***minimo***

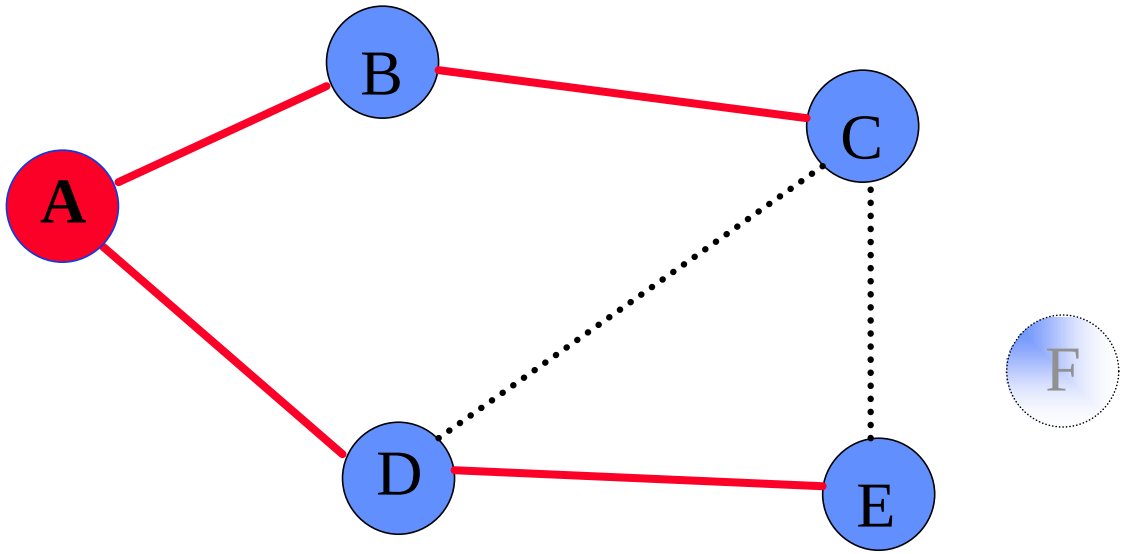


***s* = A**

Alberi breadth-first

Un ***albero breadth-first*** di un grafo non orientato ***G =<V,E>*** con ***sorgente s***, è un ***albero libero G'=<V',E'>*** tale che:

- ***G'*** è un ***sottografo del grafo non orientato sottostante*** ***G***
- ***v*** Î ***V'*** se e solo se ***v*** è raggiungibile da ***s***
- per ogni ***v*** Î ***V'***, il percorso da ***s*** a ***v*** è ***minimo***



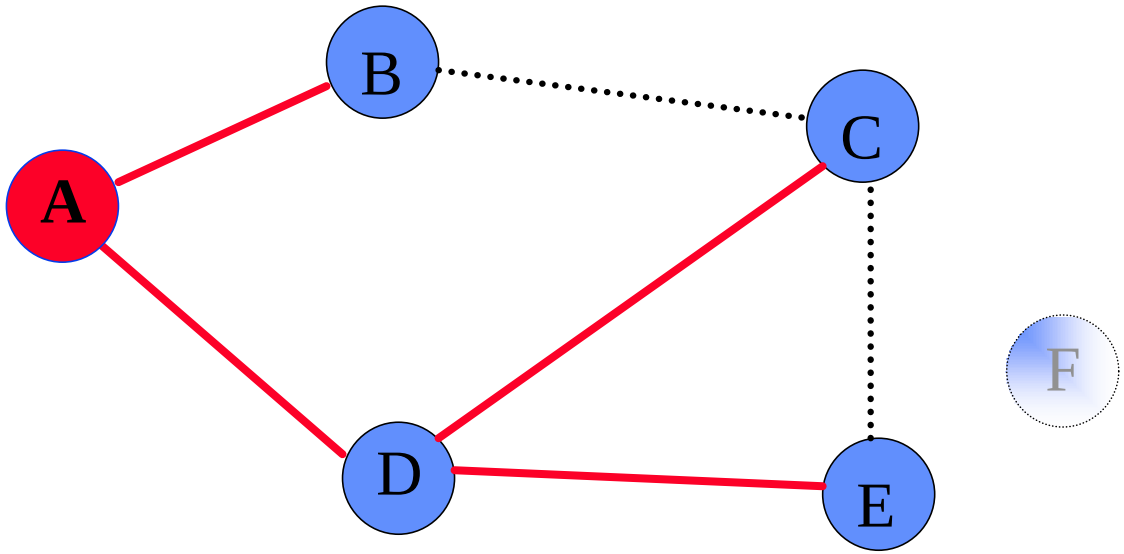
***s* = A**

Questo è un albero breadth-first

Alberi breadth-first

Un ***albero breadth-first*** di un grafo non orientato ***G =<V,E>*** con ***sorgente s***, è un ***albero libero G'=<V',E'>*** tale che:

- ***G'*** è un ***sottografo del grafo non orientato sottostante*** ***G***
- ***v*** Î ***V'*** se e solo se ***v*** è raggiungibile da ***s***
- per ogni ***v*** Î ***V'***, il percorso da ***s*** a ***v*** è ***minimo***



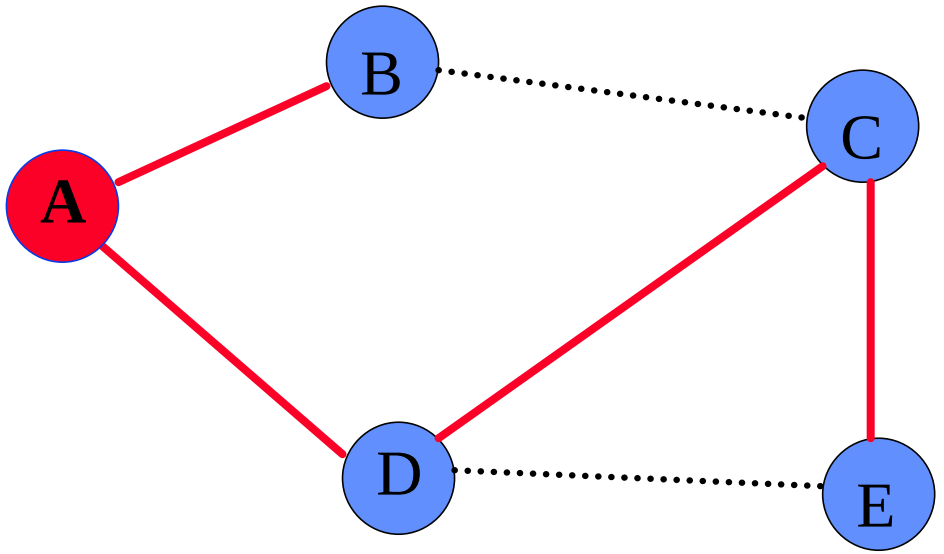
***s* = A**

Questo è un altro albero breadth-first

Alberi breadth-first

Un ***albero breadth-first*** di un grafo non orientato ***G =<V,E>*** con ***sorgente s***, è un ***albero libero G'=<V',E'>*** tale che:

- ***G'*** è un ***sottografo del grafo non orientato sottostante*** ***G***
- ***v*** Î ***V'*** se e solo se ***v*** è raggiungibile da ***s***
- per ogni ***v*** Î ***V'***, il percorso da ***s*** a ***v*** è ***minimo***



s = A

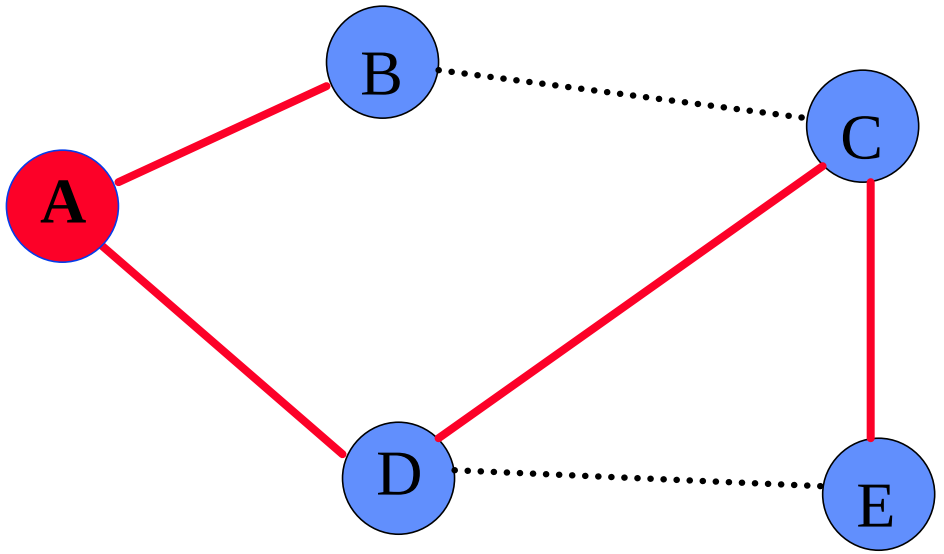
F

Questo NON è un albero breadth-first! *Perché*?

Alberi breadth-first

Un ***albero breadth-first*** di un grafo non orientato ***G =<V,E>*** con ***sorgente s***, è un ***albero libero G'=<V',E'>*** tale che:

- ***G'*** è un ***sottografo del grafo non orientato sottostante*** ***G***
- ***v*** Î ***V'*** se e solo se ***v*** è raggiungibile da ***s***
- per ogni ***v*** Î ***V'***, il percorso da ***s*** a ***v*** è ***minimo***



s = A

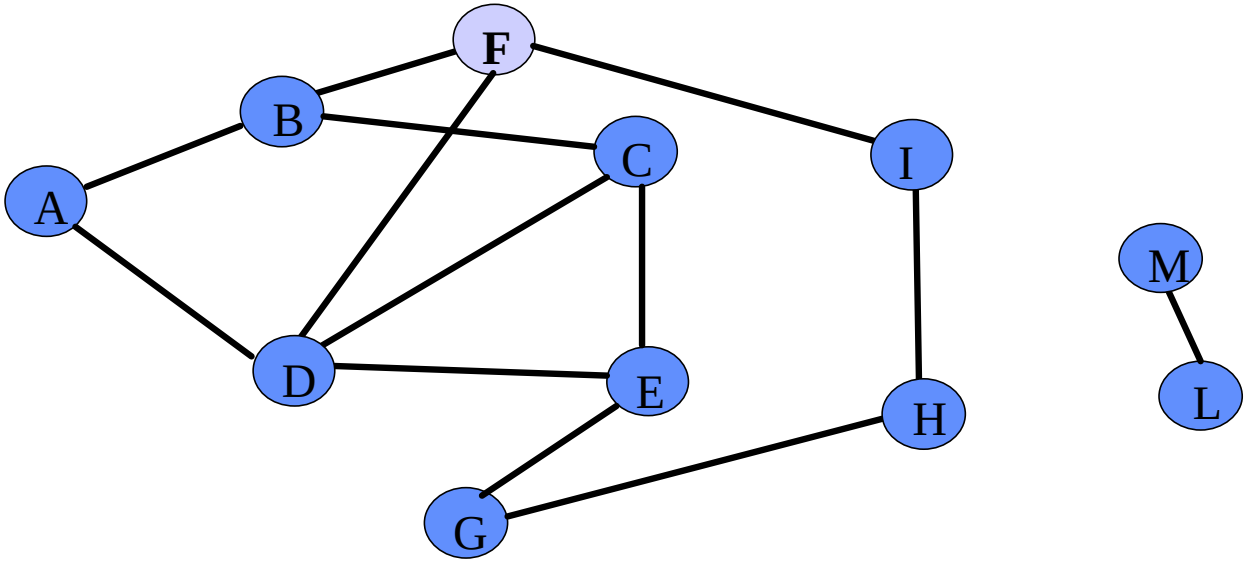
Perché il *percorso* da A ad E *NON* è *minimo*

Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

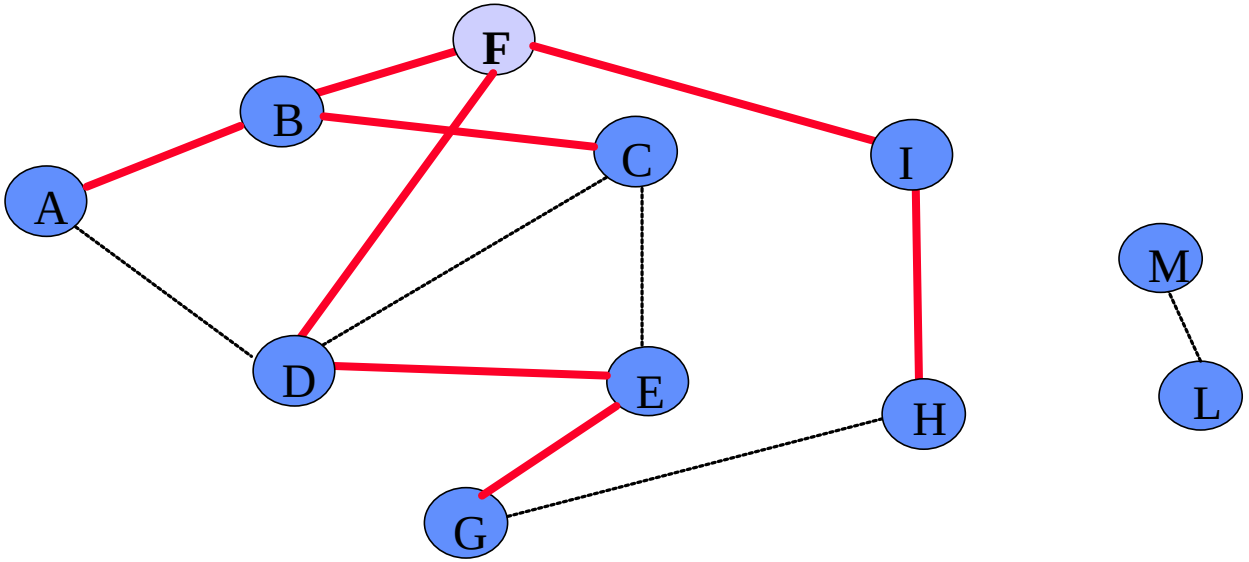


Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

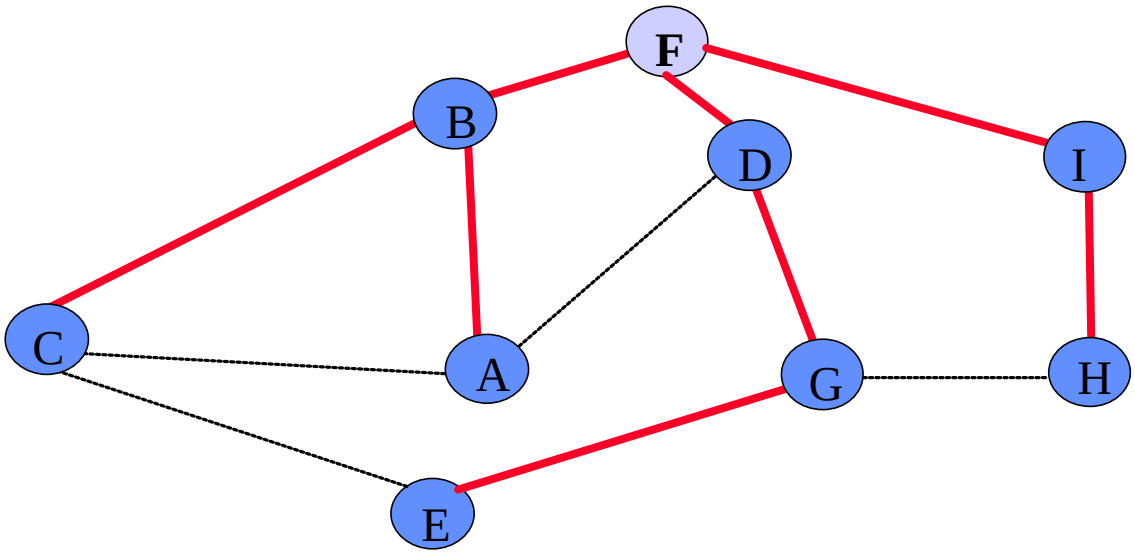


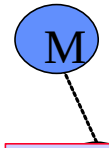
Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$





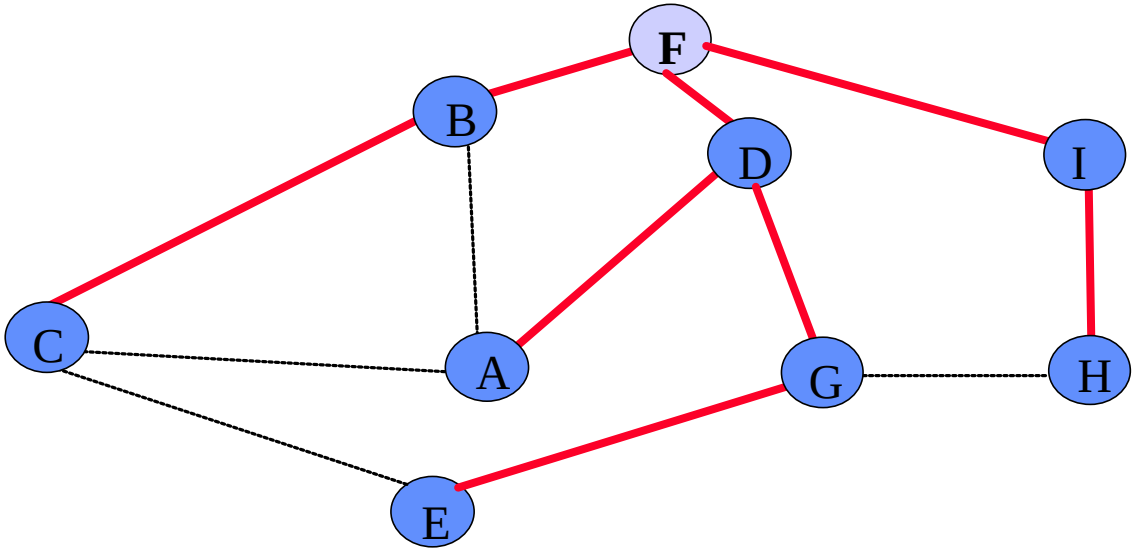
Questo è *un albero breadth-first* di *G*

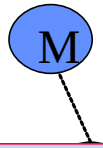
Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$





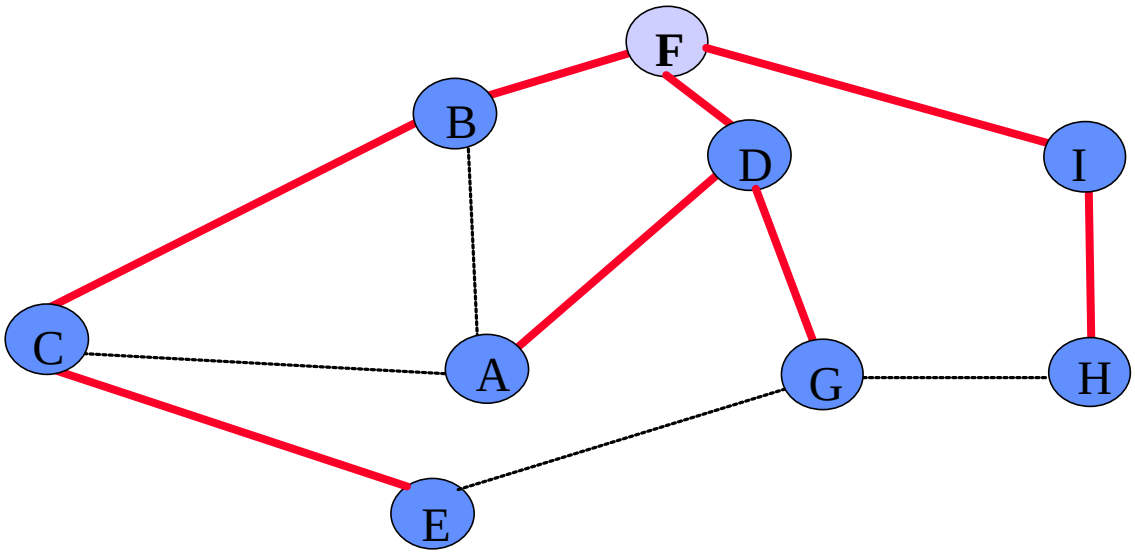
Questo è *un altro albero breadth-first* di *G*

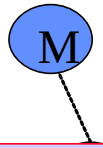
Sottografo dei predecessori e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce (nell'array $pred[]$) il *sottografo dei predecessori* denotato con $G_{pred}=<V_{pred},E_{pred}>$, dove:**

$$V_{pred} = \{ v \in V : pred[v] \neq \mathbf{Nil} \} \cup \{s\}$$

$$E_{pred} = \{ (pred[v],v) \in E : v \in V_{pred} - \{s\} \}$$

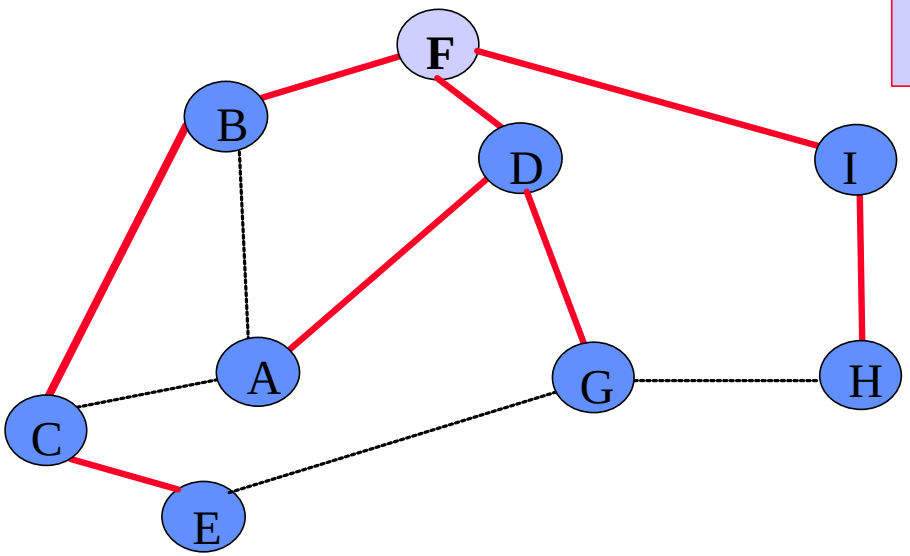




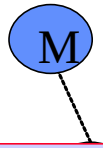
Questo è *un altro albero breadth-first* di *G*

Alberi breadth-first e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce in *pred*[] il *sottografo dei predecessori* $G_{pred}=<V_{pred},E_{pred}>$ in modo tale che G_{pred} sia un *albero breadth-first* di G.**



Questo è *un grafo dei predecessori* di *G* costruito da *BFS*

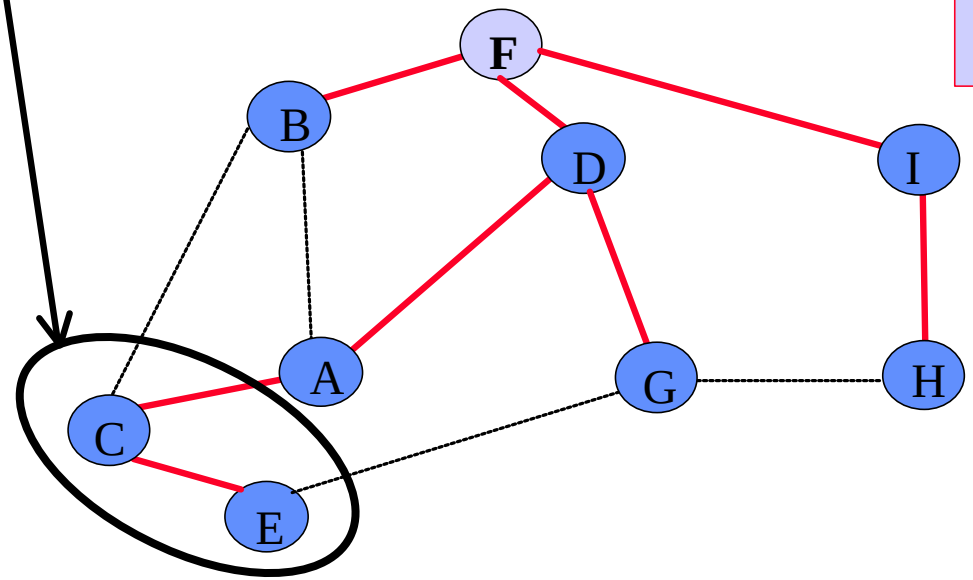


Questo è *un albero breadth-first* di *G*

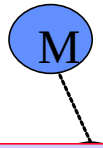
Alberi breadth-first e BFS

- **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce in *pred*[] il *sottografo dei predecessori* $G_{pred}=<V_{pred},E_{pred}>$ in modo tale che G_{pred} sia un *albero breadth-first* di G.**

I percorsi da F a C e da F ad E *non sono minimi*



Questo *non è un grafo dei predecessori* di *G* costruito da *BFS*



Questo *non è un albero breadth-first* di *G*

Alberi breadth-first e BFS

Teorema: **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce in *pred*[] il *sottografo dei predecessori* $G_{pred}=<V_{pred},E_{pred}>$ in modo tale che G_{pred} sia un *albero breadth-first* di G.**

Dimostrazione: **BFS assegna $pred[v]=u$ solo se (u,v) Î E e $d(s,v)<¥$ (solo se v è raggiungibile da s).**

Quindi V_{pred} consiste di vertici in V tutti raggiungibili da s

Poiché G_{pred} è un albero, contiene un unico percorso da s ad ogni vertice in V_{pred}

Usando *induttivamente* il *teorema di correttezza* (parte finale), segue che ognuno di questi percorsi è minimo.

Alberi breadth-first e BFS

Teorema: **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce in *pred*[] il *sottografo dei predecessori* $G_{pred}=<V_{pred},E_{pred}>$ in modo tale che G_{pred} sia un *albero breadth-first* di G.**

Dimostrazione: **Usando *induttivamente* il *teorema di correttezza* (parte finale), segue che ognuno di questi percorsi è minimo. Induzione sulla distanza k di v da s.**

Passo Base: **Se $k=0$ segue banalmente.**

Sia dato un *grafo* $G=<V,E>$ (orientato o non) e un vertice sorgente s:

- **se v ^{1} s, *uno dei percorsi minimi* tra s e v è il *percorso minimo* da s a *pred*[v] seguito dall'arco (*pred*[v],v).**

Alberi breadth-first e BFS

Teorema: **L'*algoritmo BFS* eseguito su un grafo $G=<V,E>$ costruisce in *pred*[] il *sottografo dei predecessori* $G_{pred}=<V_{pred},E_{pred}>$ in modo tale che G_{pred} sia un *albero breadth-first* di G.**

Dimostrazione: **Usando *induttivamente* il *teorema di correttezza* (parte finale), segue che ognuno di questi percorsi è minimo. Induzione sulla distanza k di v da s.**

Passo Induttivo: **Il percorso tra s e *pred*[v] è minimo per induzione**

Sia dato un *grafo* $G=<V,E>$ (orientato o non) e un vertice sorgente s:

- **se v ^{1} s, *uno dei percorsi minimi* tra s e v è il *percorso minimo* da s a *pred*[v] seguito dall'arco (*pred*[v],v).**

Ma allora per il *teorema di correttezza* lo è anche il *percorso* da s a *pred*[v] seguito dall'arco (*pred*[v],v).

Applicazione di BFS: calcolo del percorso minimo tra due vertici

Definizione del problema:

Dato un grafo *G* e due vertici *s* e *v*, stampare il *percorso minimo* che congiunge *s* e *v*.

Sfruttando le *proprietà* di *BFS* che abbiamo dimostrato fin qui, possiamo facilmente definire un algoritmo che utilizza *BFS* opportunamente e che risolve il problema.

Stampa del percorso minimo

```
Stampa-percorso(G:grafo, s,v:vertice)
 BFS(G,s)
 if v = s
  then stampa s
  else if pred[v] = NIL
         then stampa “non esiste alcun cammino
                          tra s e v”
         else Stampa-percorso(G,s,pred[v])
         stampa v
```