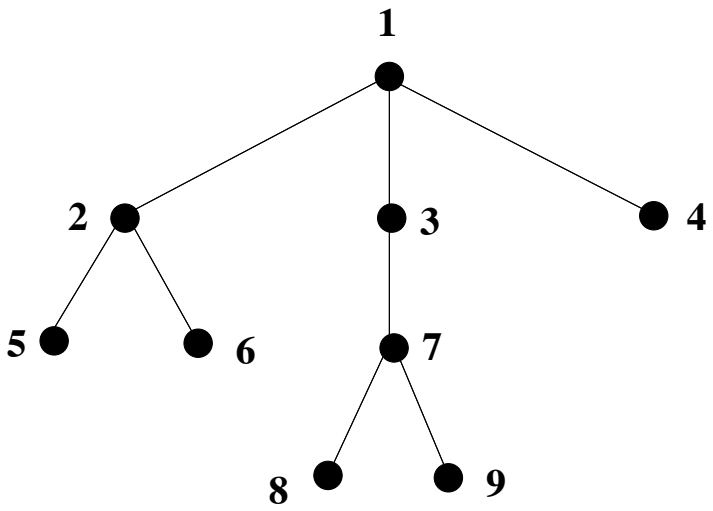
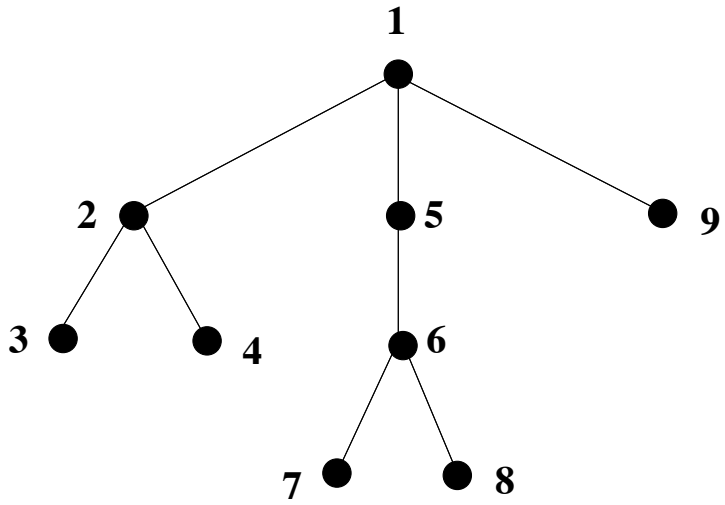
Visite di grafi

Gli algoritmi di visita di un grafo hanno come obiettivo l'esplorazione di tutti i nodi e gli archi del grafo. Vi sono due modi principali per esplorare un grafo:

- **visita in ampiezza** (**breadth-first visit**): mi muovo il più possible in ampiezza (di fratello in fratello) e ritorno sui miei passi solo quando non posso più spingermi oltre;
- **visita in profondità** (**depth-first visit**): mi muovo il più possible in profondità (di padre in figlio) e ritorno sui miei passi solo quando non posso più spingermi oltre.

Le due visite a confronto





Visita in ampiezza

I dati di ingresso dell'algoritmo **BFS** (**Breadth-First Search**) sono un grafo $G = (V, E)$ e un nodo sorgente $s \in V$. L'algoritmo:

- esplora tutti i **vertici raggiungibili** a partire da s;
- calcola e memorizza le **distanze** (la lunghezza del cammino minimo) da s a tutti i nodi raggiungibili da s;
- costruisce **l'albero dei cammini minimi** (BF-albero) da s a tutti i nodi raggiungibili da s.

Visita in ampiezza

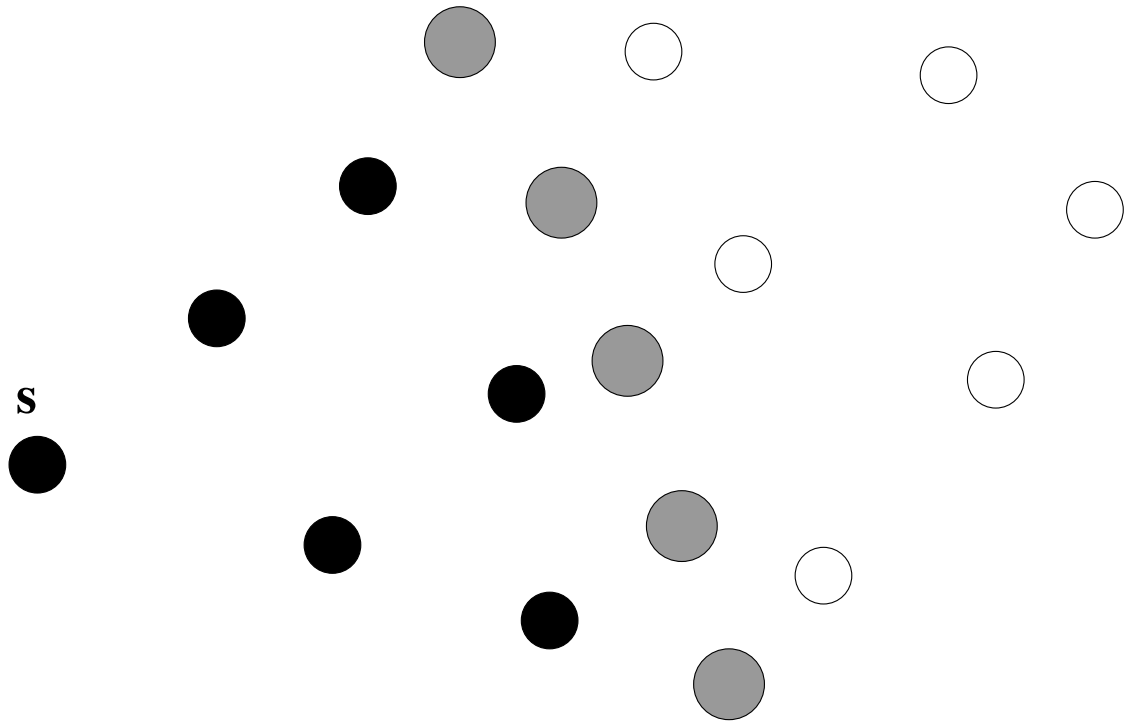
Ogni nodo viene **scoperto** quanto è incontrato la prima volta dalla procedura e **terminato** quando tutti i suoi successori sono stati scoperti.

Per distinguere i nodi visitati da quelli ancora da visitare, BFS colora i nodi di bianco, di grigio e di nero, con il seguente significato:

- **nodi bianchi** corrispondono a nodi ancora da scoprire;
- **nodi grigi** corrispondono a nodi scoperti ma non ancora terminati;
- **nodi neri** corrispondono a nodi terminati.

Quindi i nodi grigi formano la **frontiera** tra nodi scoperti (zona nera) e nodi ancora da scoprire (zona bianca).

Zona bianca, grigia e nera



BFS: strutture di dati

L'algoritmo BFS assume che il grafo G sia rappresentato mediante **liste di adiacenza**. Esso usa le seguenti strutture di dati:

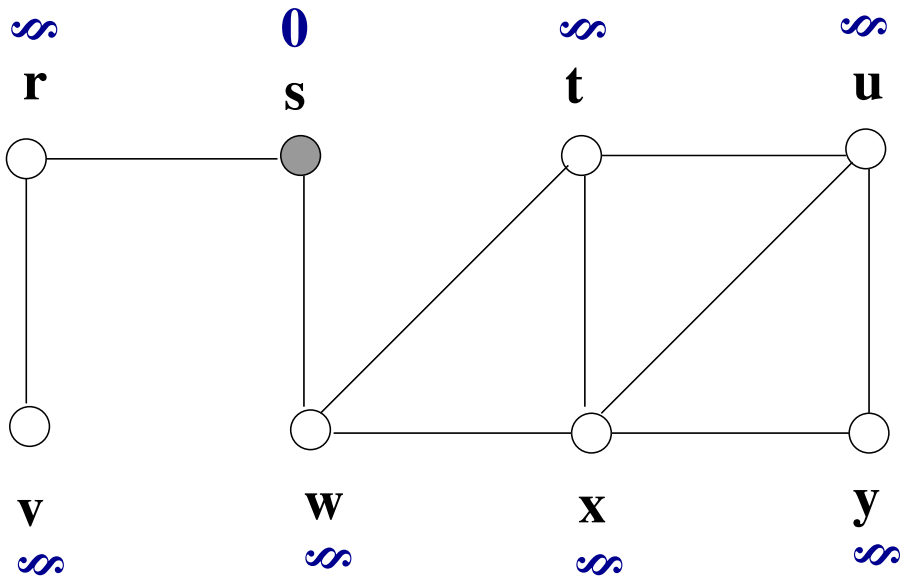
- un vettore *color* tale che $color[u]$ è il colore del nodo u;
- un vettore π tale che $\pi[u]$ è il padre del nodo u;
- un vettore d tale che $d[u]$ è la distanza di u dalla sorgente s;
- una coda Q per memorizzare i nodi grigi (la frontiera).

Visita in ampiezza

1: $BFS(G, s)$
2: **for** each $u \in V[G] \setminus \{s\}$ **do**
3: $color[u] \leftarrow$ WHITE
4: $d[u] \leftarrow \infty$
5: $\pi[u] \leftarrow$ NIL
6: **end for**
7: $color[s] \leftarrow$ GRAY
8: $d[s] \leftarrow 0$
9: $\pi[s] \leftarrow$ NIL
10: $Q \leftarrow \emptyset$
11: $Enqueue(Q, s)$

```
12: while not QueueEmpty(Q) do
13:     u ← Dequeue(Q)
14:     for each v ∈ Adj[u] do
15:         if color[v] = WHITE then
16:             color[v] ← GRAY
17:             d[v] ← d[u] + 1
18:             π[v] ← u
19:             Enqueue(Q, v)
20:         end if
21:     end for
22:     color[u] ← BLACK
23: end while
```

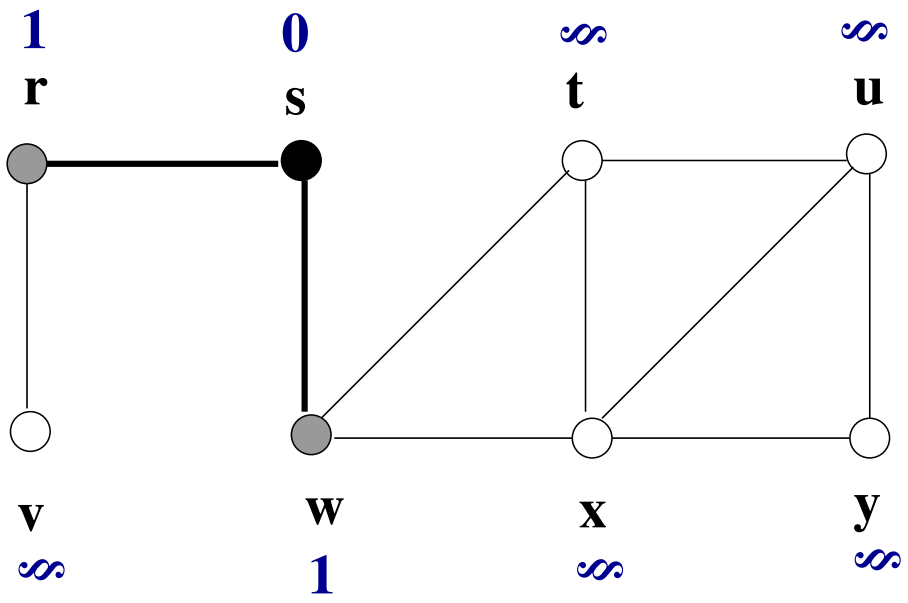
(a)



Q

s		

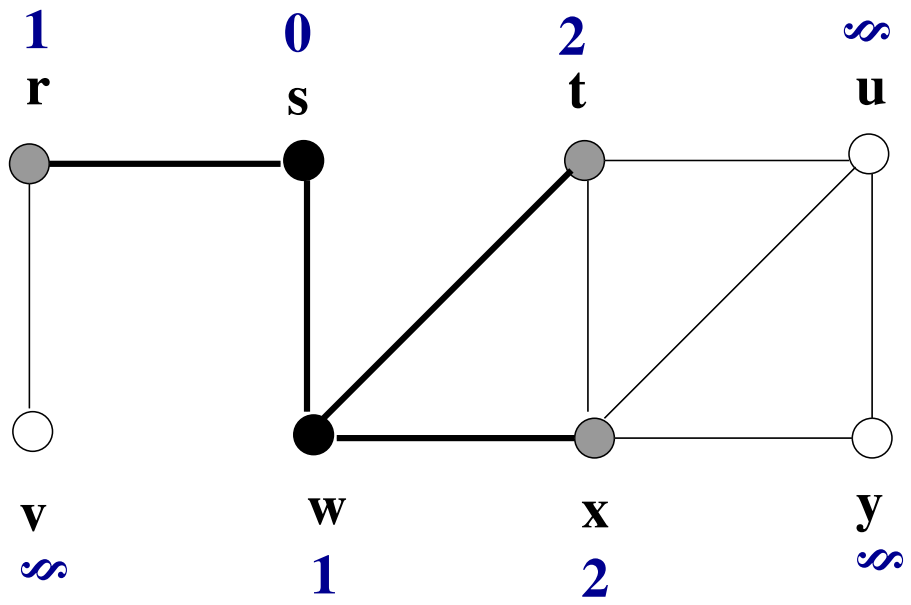
(b)



Q

w	r	

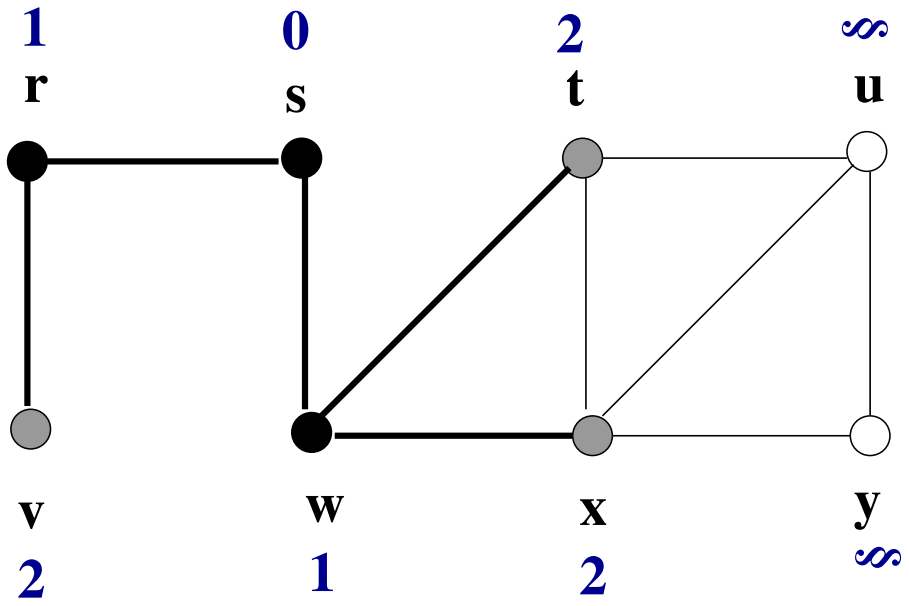
(c)



Q

r	t	x

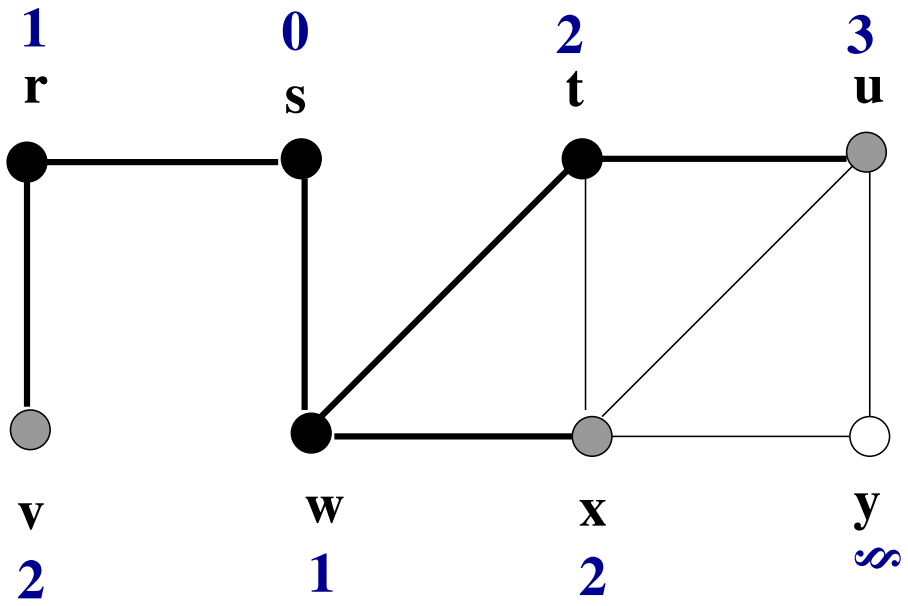
(d)



Q

t	x	v

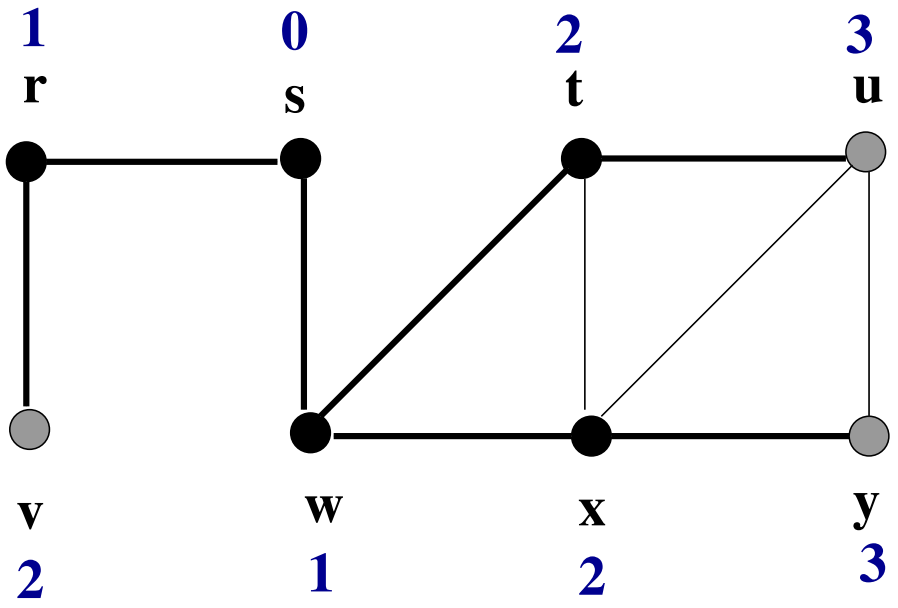
(e)



Q

x	v	u

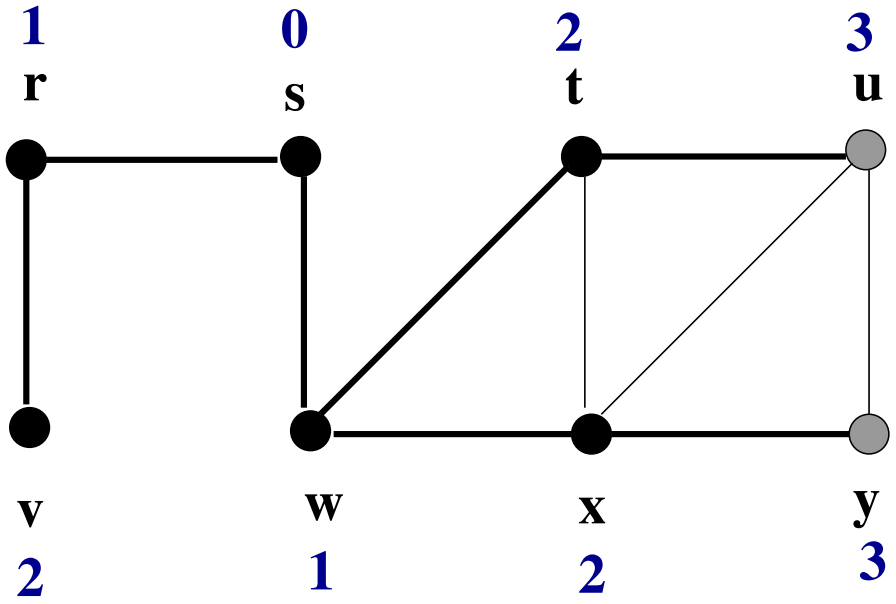
(f)



Q

v	u	y

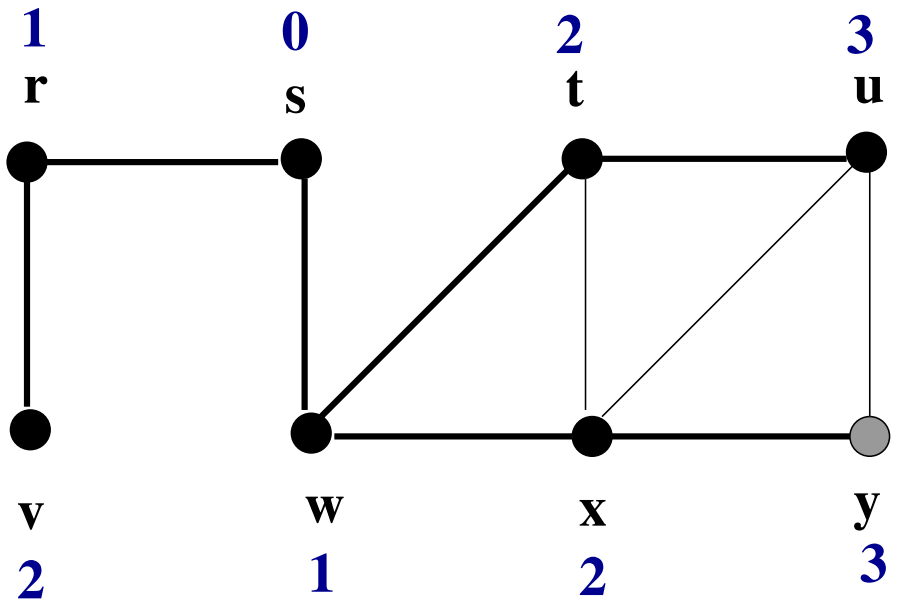
(g)



Q

u	y	

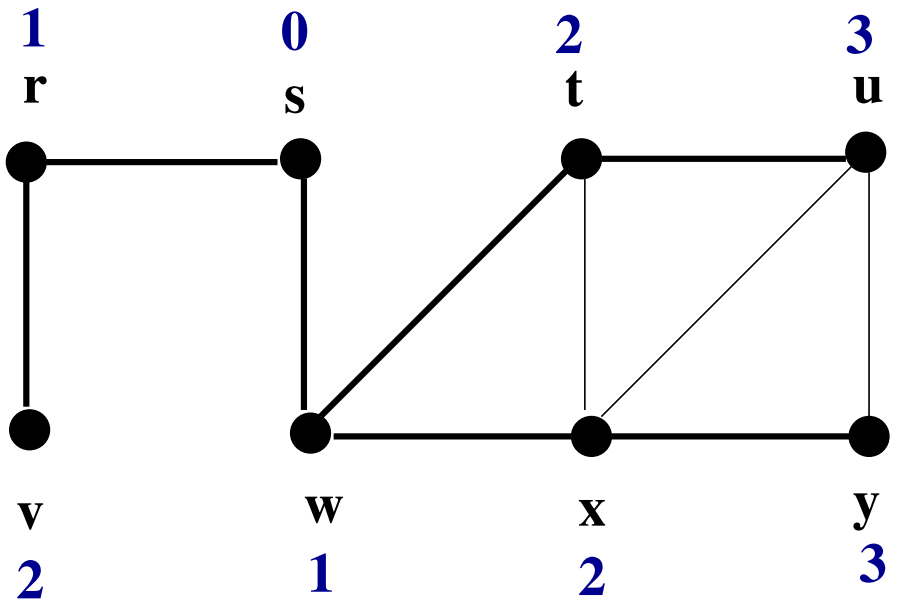
(h)



Q

y		

(i)



Q

Complessità di BFS

1: $BFS(G, s)$
2: **for** each $u \in V[G] \setminus \{s\}$ **do**
3: $color[u] \leftarrow$ WHITE
4: $d[u] \leftarrow \infty$
5: $\pi[u] \leftarrow$ NIL
6: **end for**
7: $color[s] \leftarrow$ GRAY
8: $d[s] \leftarrow 0$
9: $\pi[s] \leftarrow$ NIL
10: $Q \leftarrow \emptyset$
11: $Enqueue(Q, s)$

```
12: while not QueueEmpty(Q) do
13:     u ← Dequeue(Q)
14:     for each v ∈ Adj[u] do
15:         if color[v] = WHITE then
16:             color[v] ← GRAY
17:             d[v] ← d[u] + 1
18:             π[v] ← u
19:             Enqueue(Q, v)
20:         end if
21:     end for
22:     color[u] ← BLACK
23: end while
```

Complessità di BFS

L'algoritmo consiste di una fase di inizializzazione e di un ciclo while.

- La complessità della fase di inizializzazione è $O(n)$.
- La complessità del ciclo while si valuta così:
  - **Quante volte viene eseguito il ciclo?** Ogni nodo raggiungibile da s viene caricato in Q esattamente una volta. Ad ogni iterazione scarico esattamente un nodo da Q. Quindi il ciclo viene eseguito al massimo n volte (tutti i nodi sono raggiungibili da s);

- **Quanto costa ogni iterazione del ciclo?** Ad ogni iterazione del ciclo:
  - visito la lista di adiacenza di un nodo;
  - eseguo altre operazioni di costo costante (esempio, Enqueue e Dequeue).

  Dato che la somma delle lunghezze di tutte le liste di adiacenza è $O(m)$, la complessità del ciclo while vale $O(n + m)$.

In totale, BFS costa

$$O(n + (n + m)) = O(n + m)$$

cioé è **lineare** nella dimensione dell'input.

Esercizio. Si scriva una procedura che stampa il cammino minimo da s a v assumendo che $BFS(G, s)$ sia già stata invocata per calcolare il BF-albero con radice s.

PrintPath(G, s, v)

```
1: if v = s then
2:    print s
3: else
4:    if π[v] = NIL then
5:       print  "no path exists"
6:    else
7:       PrintPath(G, s, π[v])
8:       print v
9:    end if
10: end if
```

PrintPath(G, s, v)

```
 1: if v = s then
 2:    print s
 3: else
 4:    if π[v] = NIL then
 5:       print  "no path exists"
 6:    else
 7:       x ← v
 8:       repeat
 9:          Push(S, x)
10:          x ← π[x]
11:       until x = NIL
12:       while not StackEmpty(S) do
13:          print Pop(S)
14:       end while
15:    end if
16: end if
```

Visita in profondità

L'algoritmo **DFS** (**Depth-First Search**) ha in input un grafo $G = (V, E)$. L'algoritmo:

- esplora tutti i **vertici del grafo**. L'algoritmo parte da una sorgente qualsiasi e cambia sorgente quanto tutti i nodi raggiungibili dalla sorgente sono stati esplorati;
- registra i nodi visitati in un insieme di alberi, chiamato **DF-foresta**. Ogni albero ha come radice una sorgente;
- calcola e memorizza i **tempi di scoperta** e i **tempi di terminazione** di ogni nodo.

Visita in profondità

Per distinguere i nodi visitati da quelli ancora da visitare, DFS colora i nodi di bianco, di grigio e di nero:

- ogni nodo è inizialmente **bianco**;
- quando un vertice viene **scoperto**, cioè incontrato per la prima volta, il suo colore diventa **grigio**;
- quando un nodo viene **terminato**, cioè quando tutti i suoi **discendenti** sono stati esaminati, il colore diventa **nero**.

DFS: strutture di dati

L'algoritmo DFS assume che il grafo G sia rappresentato mediante **liste di adiacenza**. Esso usa le seguenti strutture di dati:

- un vettore *color* tale che $color[u]$ è il colore del nodo u;
- un vettore π tale che $\pi[u]$ è il padre del nodo u;
- un vettore d tale che $d[u]$ contiene l'istante in cui il nodo u viene scoperto;
- un vettore f tale che $f[u]$ contiene l'istante in cui il nodo u viene terminato.

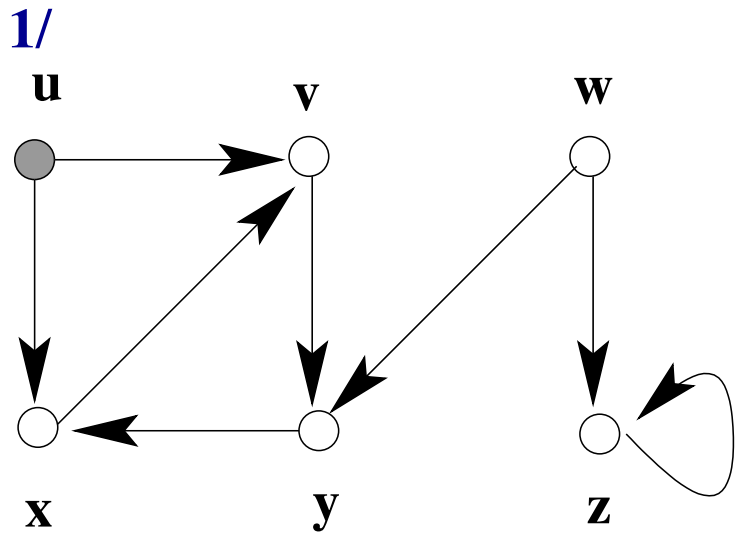
$DFS(G)$

1: **for** each $u \in V[G]$ **do**
2: $color[u] \leftarrow$ WHITE
3: $\pi[u] \leftarrow$ NIL
4: **end for**
5: $time \leftarrow 0$
6: **for** each $u \in V[G]$ **do**
7: **if** $color[u] =$ WHITE **then**
8: $DFSVisit(u)$
9: **end if**
10: **end for**

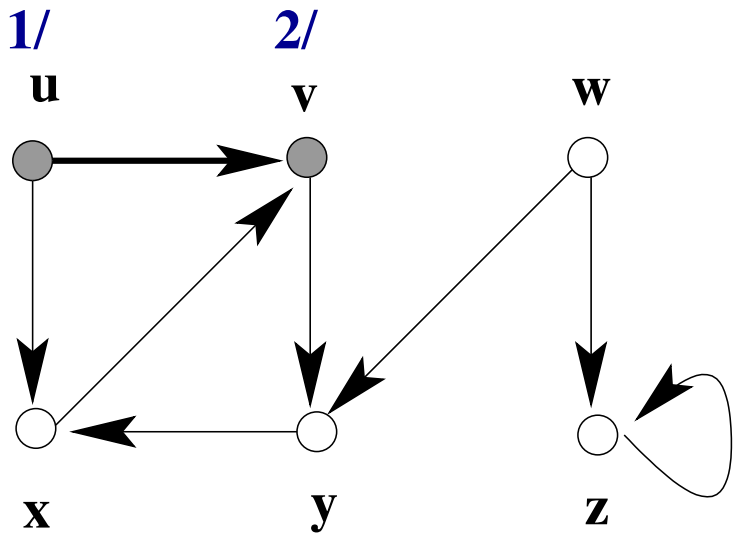
$DFSVisit(u)$

1: $color[u] \leftarrow$ GRAY
2: $time \leftarrow time + 1$
3: $d[u] \leftarrow time$
4: **for** each $v \in Adj[u]$ **do**
5: **if** $color[v] =$ WHITE **then**
6: $\pi[v] \leftarrow u$
7: $DFSVisit(v)$
8: **end if**
9: **end for**
10: $color[u] \leftarrow$ BLACK
11: $time \leftarrow time + 1$
12: $f[u] \leftarrow time$

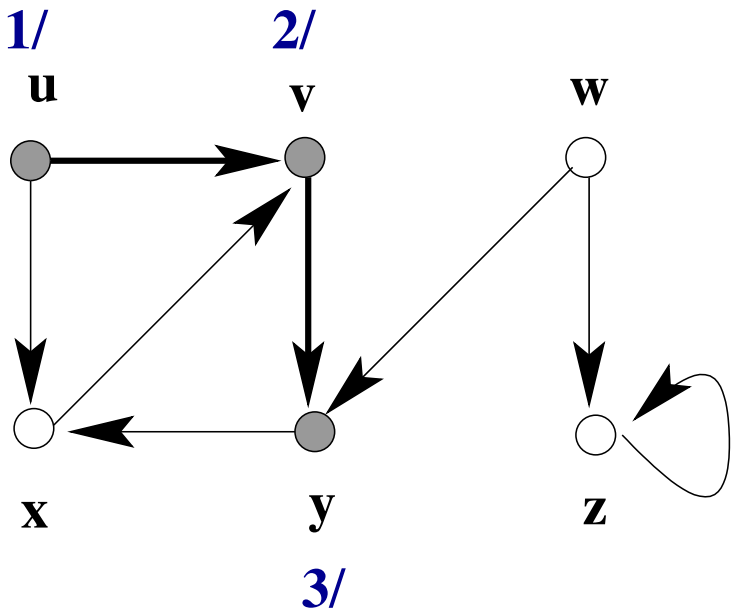
(a)



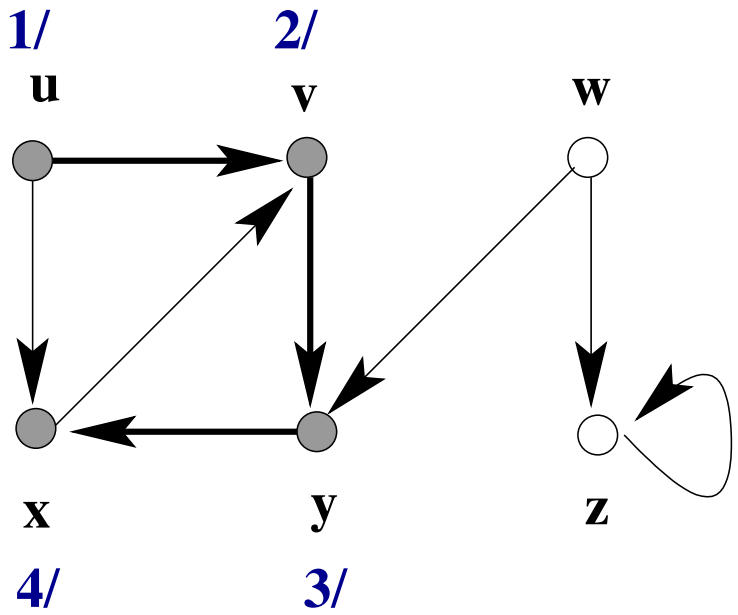
(b)



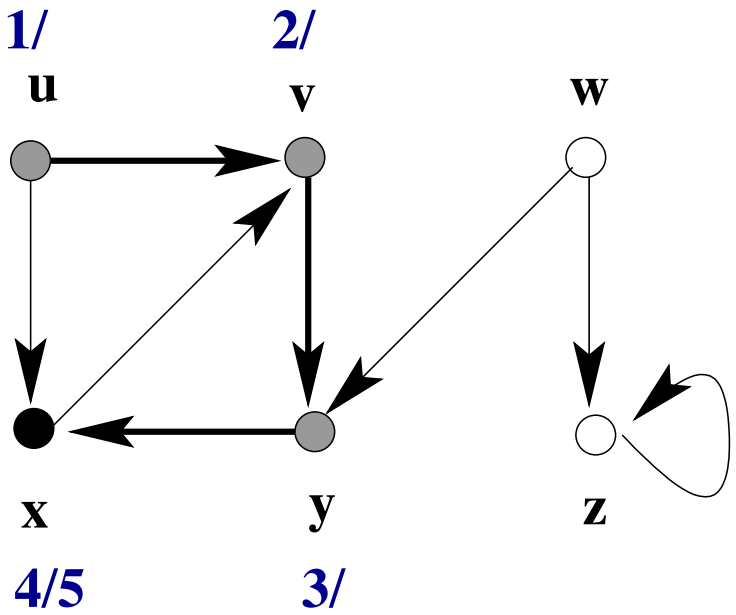
(c)



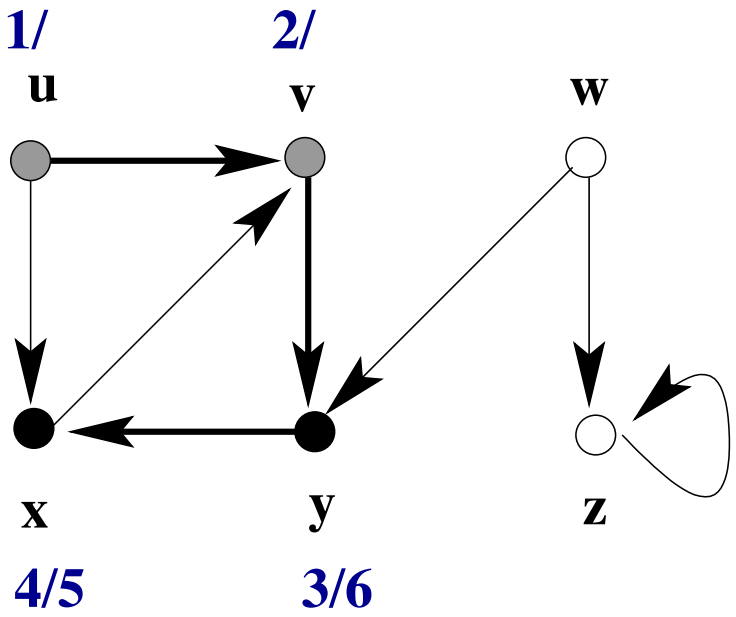
(d)



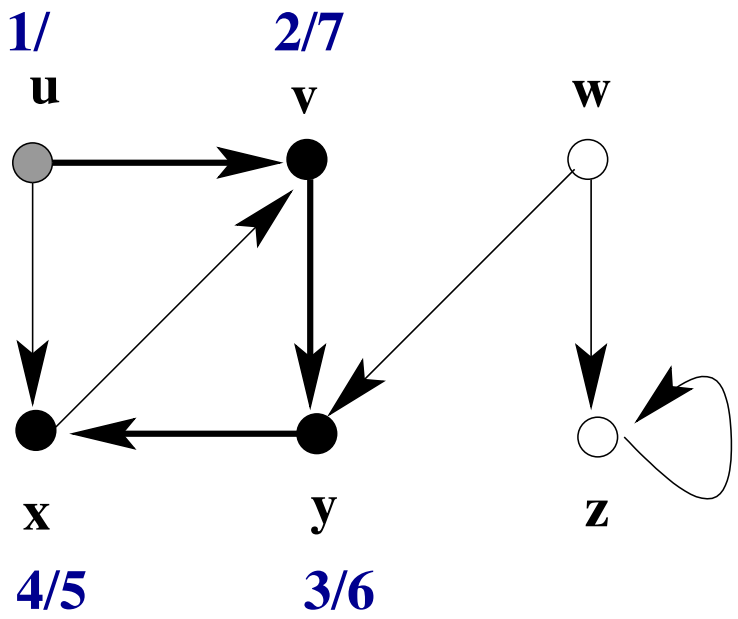
(e)



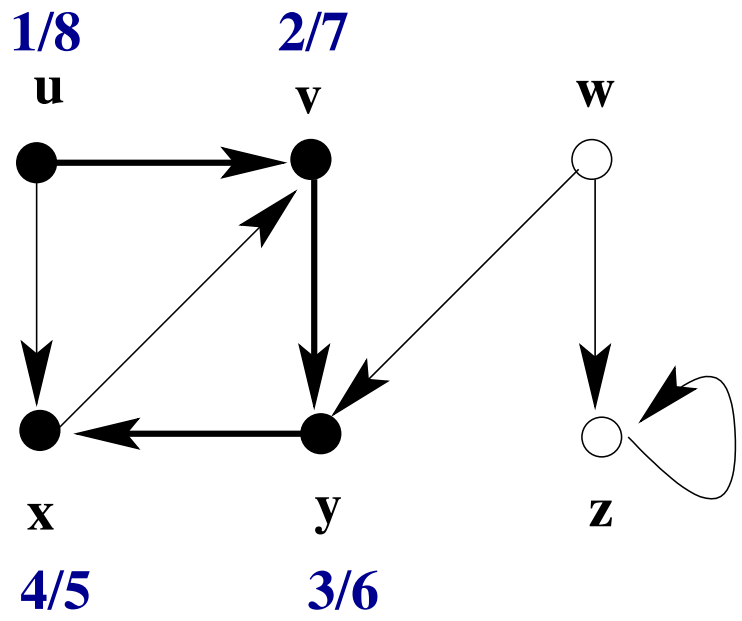
(f)



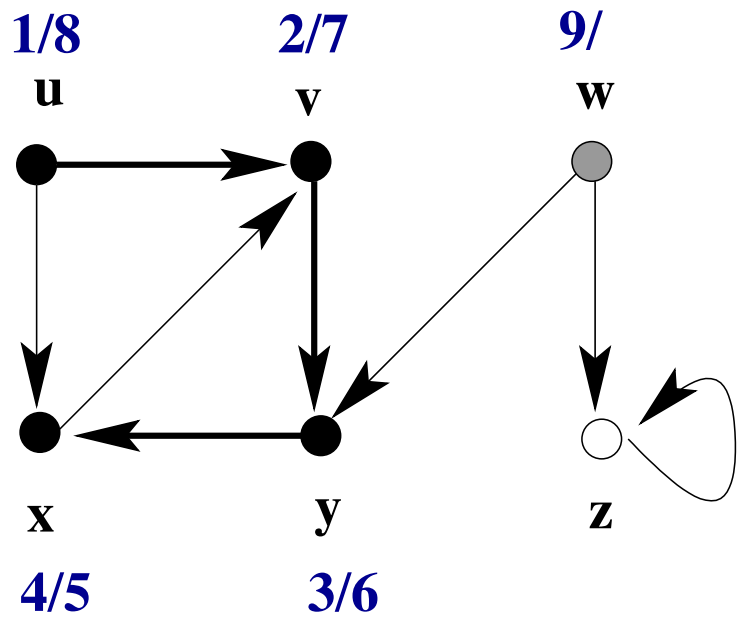
(g)



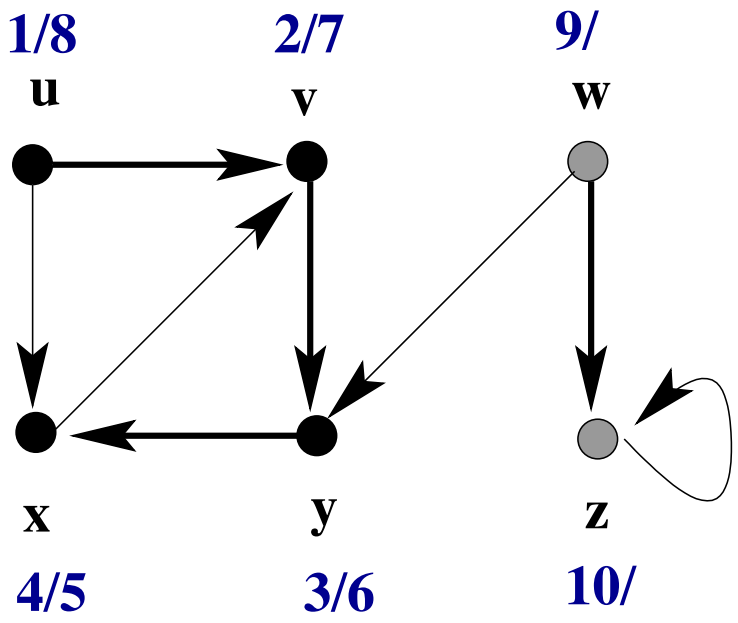
(h)



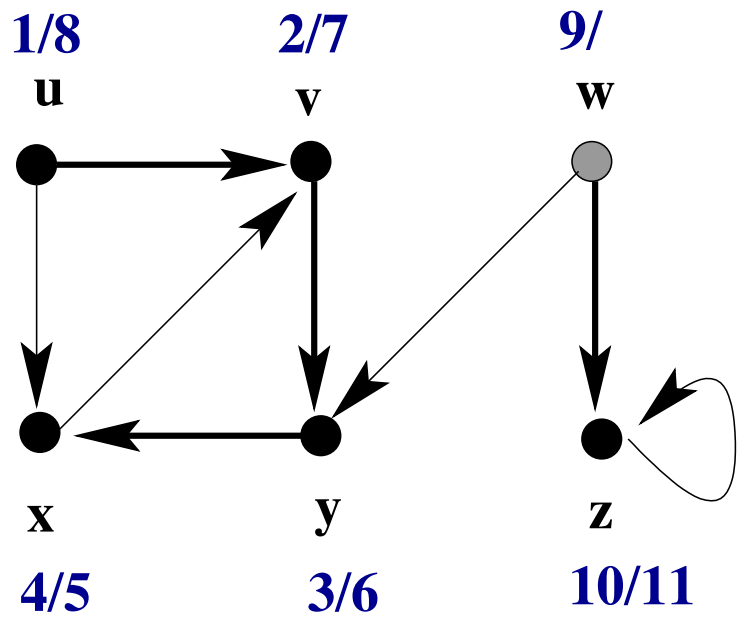
(i)



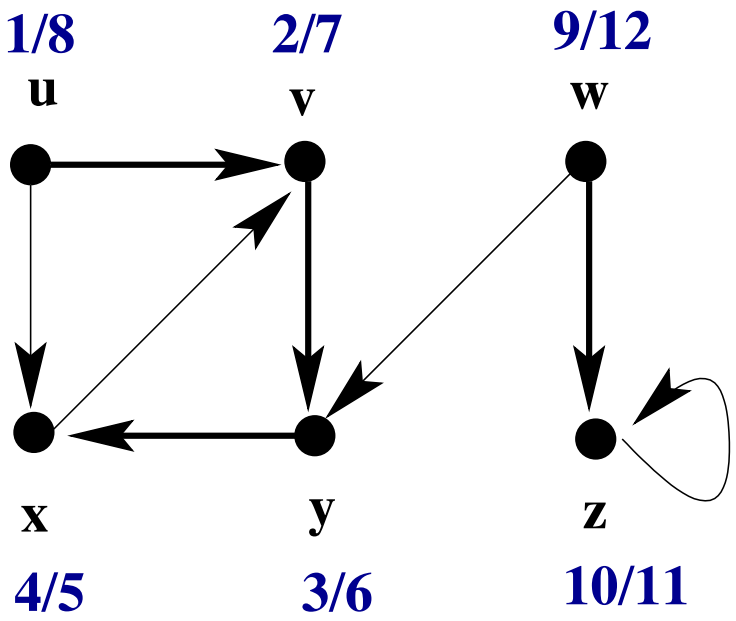
(j)



(k)



(l)



Complessità di DFS

$DFS(G)$

```
 1: for each u ∈ V[G] do
 2:     color[u] ← WHITE
 3:     π[u] ← NIL
 4: end for
 5: time ← 0
 6: for each u ∈ V[G] do
 7:     if color[u] = WHITE then
 8:         DFSVisit(u)
 9:     end if
10: end for
```

$DFSVisit(u)$

1: $color[u] \leftarrow$ GRAY
2: $time \leftarrow time + 1$
3: $d[u] \leftarrow time$
4: **for** each $v \in Adj[u]$ **do**
5: **if** $color[v] =$ WHITE **then**
6: $\pi[v] \leftarrow u$
7: $DFSVisit(v)$
8: **end if**
9: **end for**
10: $color[u] \leftarrow$ BLACK
11: $time \leftarrow time + 1$
12: $f[u] \leftarrow time$

Complessità di DFS

- La procedura DFS impiega un tempo $O(n)$ senza considerare il tempo di esecuzione delle chiamate di DFSVisit;
- La procedura DFSVisit viene chiamata esattamente una volta per ogni nodo, quindi al più n volte;
- DFSVisit visita la lista di adiacenza del nodo su cui è chiamata, oltre a fare altre operazioni di costo costante;

Quindi tutte le chiamata a DFSVisit costano $O(n + m)$, e il costo totale di DFS è

$$O(n + (n + m)) = O(n + m)$$

quindi **lineare** nella dimensione dell'input.

Proprietà della visita in profondità

Per ogni nodo u, al termine di DFS, vale che:

$$1 \leq d[u] < f[u] \leq 2n$$

in quanto esiste un unico evento di scoperta e un unico evento di fine per ognuno degli n vertici.

Proprietà della visita in profondità

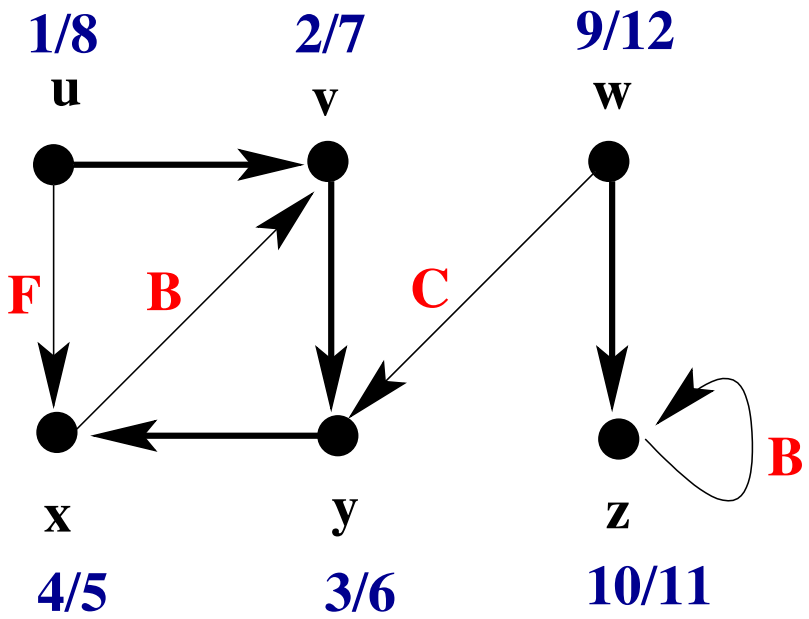
Teorema delle parentesi. Per ogni visita di profondità di un grafo $G = (V, E)$ (diretto o indiretto), per ogni due vertici distinti u e v, esattamente una delle seguenti condizione è vera:

1. gli intervalli $[d[u], f[u]]$ e $[d[v], f[v]]$ sono disgiunti, e nessuno tra u e v è un discendente dell'altro nella DF-foresta;
2. l'intervallo $[d[u], f[u]]$ è contenuto nell'intervallo $[d[v], f[v]]$ e u è un discendente di v nella DF-foresta;
3. l'intervallo $[d[v], f[v]]$ è contenuto nell'intervallo $[d[u], f[u]]$ e v è un discendente di u nella DF-foresta.

Classificazione degli archi

Possiamo definire quattro tipi di archi:

1. **archi tree**: sono gli archi della DF-foresta;
2. **archi back**: sono archi (u, v) che connettono un vertice u ad un suo ascendente v nello stesso albero appartenente alla DF-foresta;
3. **archi forward**: sono archi non tree (u, v) che connettono un vertice u ad un suo discendente v nello stesso albero appartenente alla DF-foresta;
4. **archi cross**: tutti gli altri archi. Possono connettere vertici nello stesso albero (purché nessuno dei due vertici sia un discendente dell'altro) oppure connettere vertici di alberi diversi.

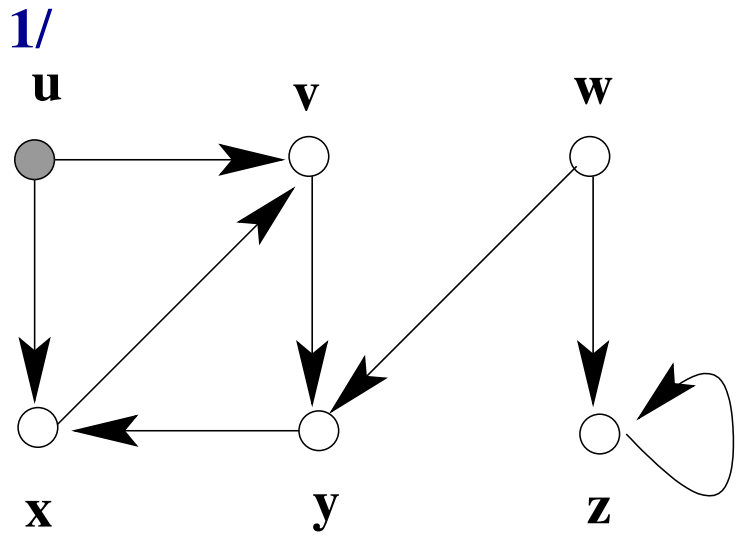

1/8
u
2/7
v
9/12
w
F
B
C
B
x
y
z
4/5
3/6
10/11

Classificazione degli archi

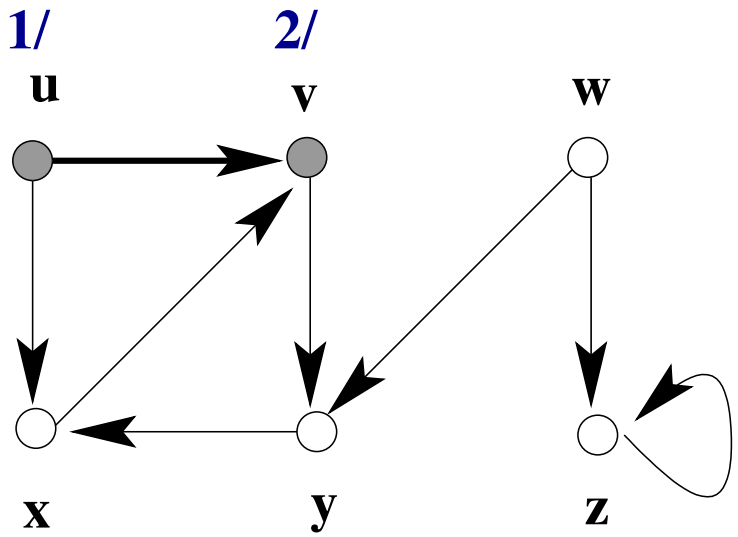
Teorema Sia (u, v) un arco di un grafo G. Sia C il colore del nodo v quando l'arco (u, v) viene esplorato per la prima volta dalla visita in profondità. Allora:

- se C è bianco, allora (u, v) è un arco tree;
- se C è grigio, allora (u, v) è un arco back;
- se C è nero, allora:
  - se $d[u] < d[v]$, allora (u, v) è un arco forward;
  - se $d[u] > d[v]$, allora (u, v) è un arco cross.

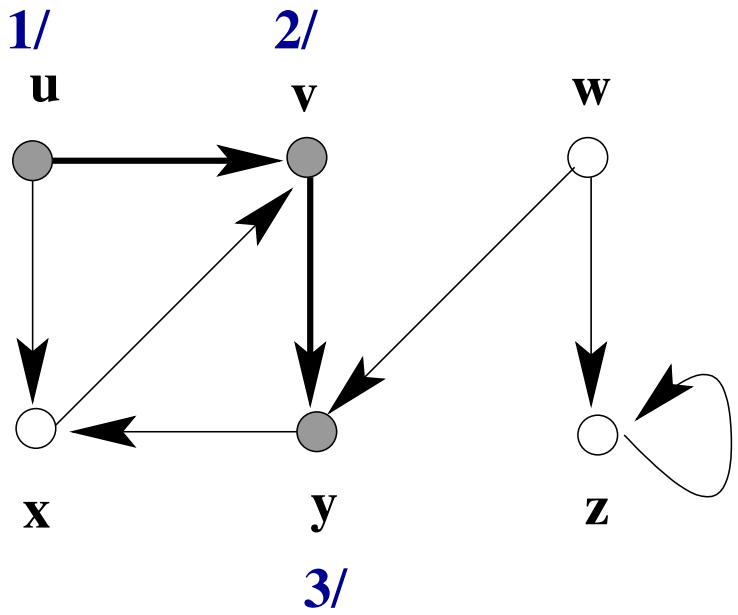
(a)



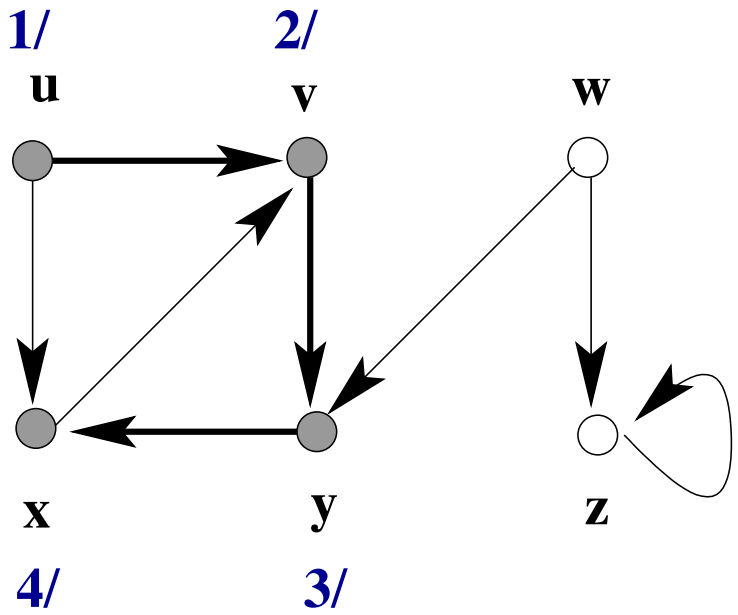
(b)



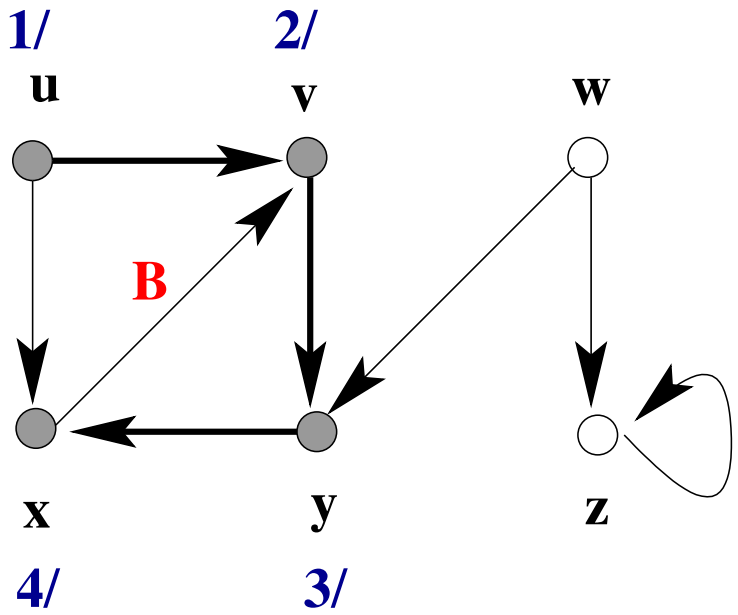
(c)



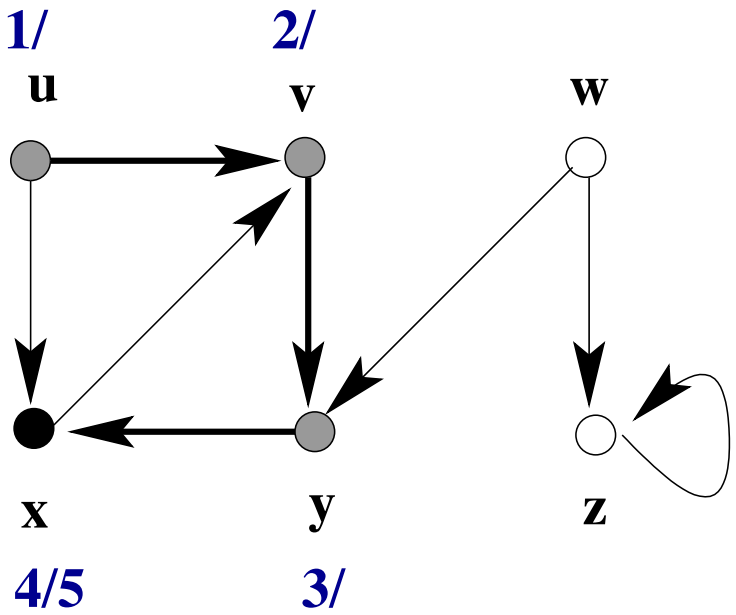
(d)



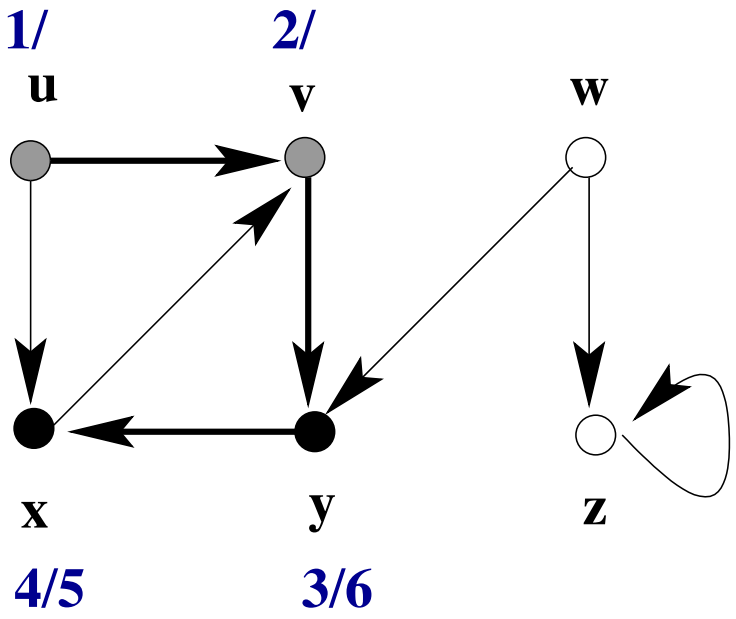
(d)



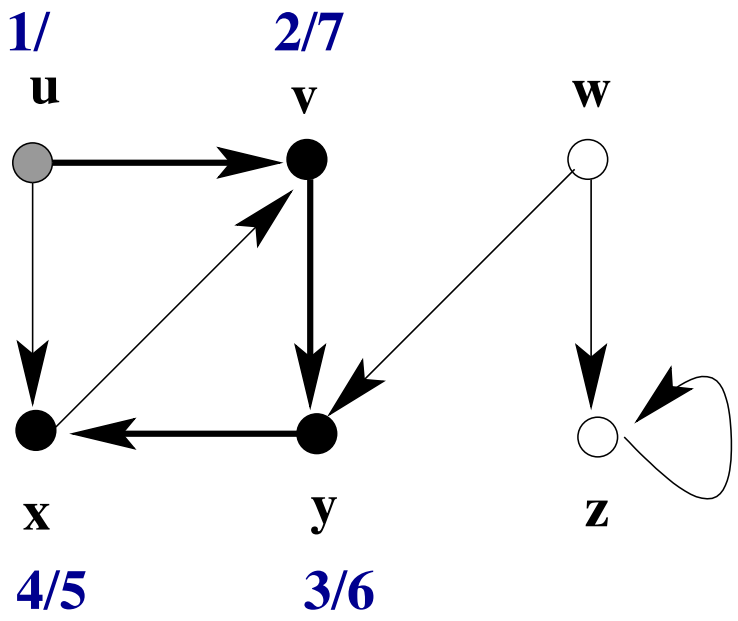
(e)



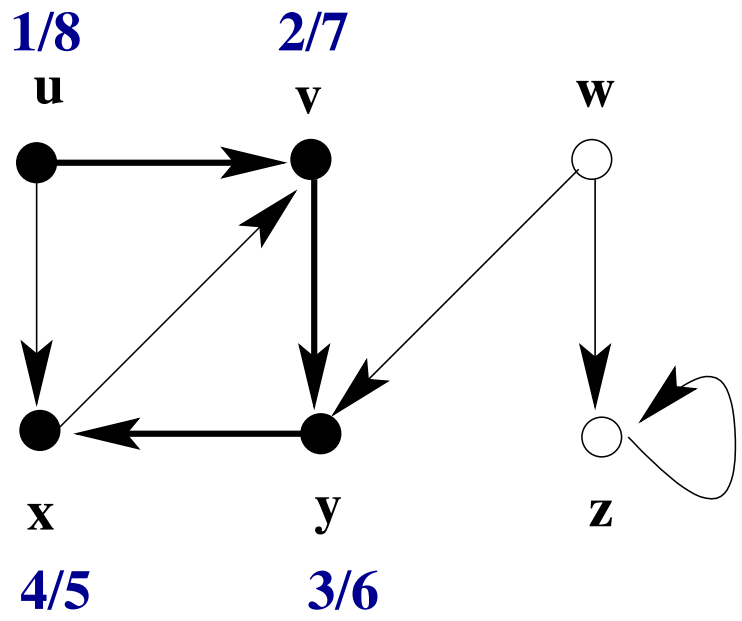
(f)



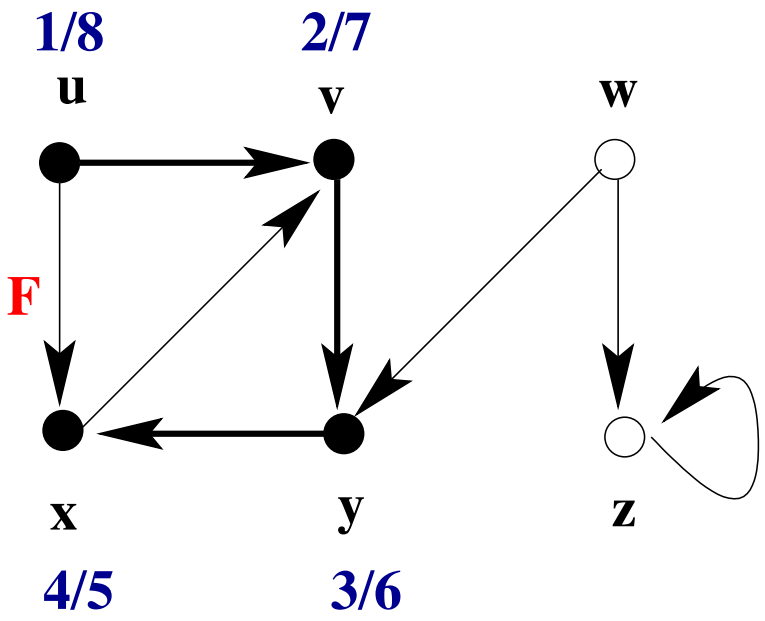
(g)



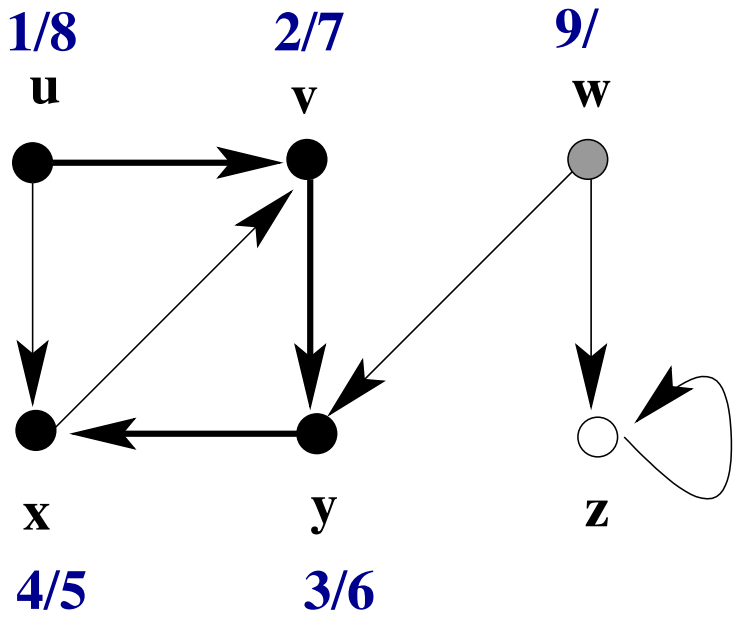
(h)



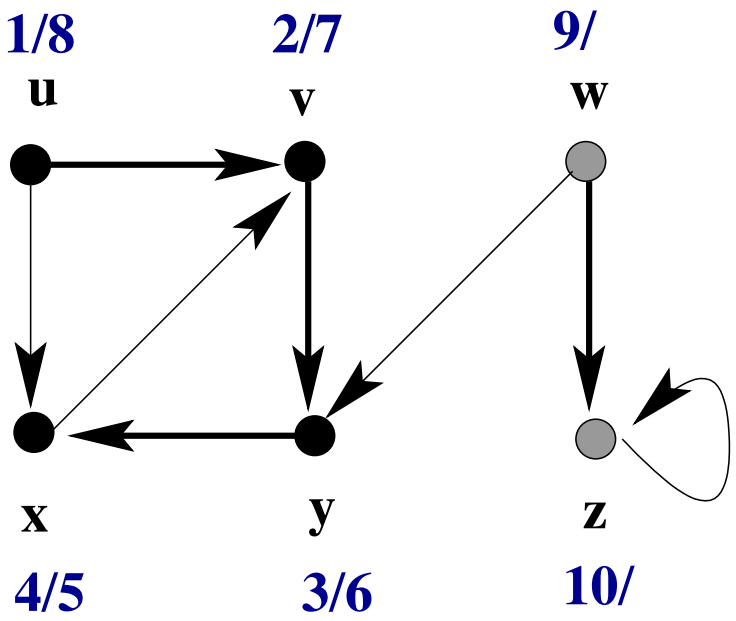
(h)



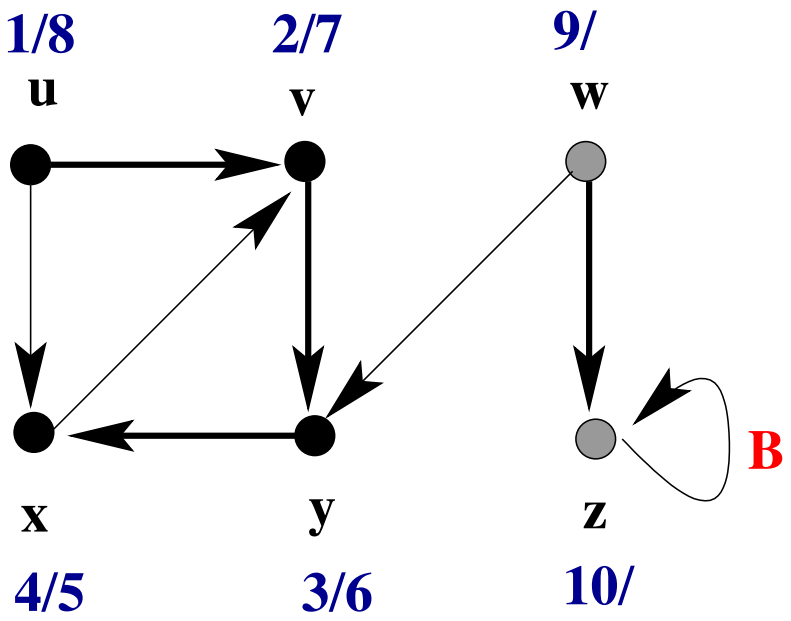
(i)



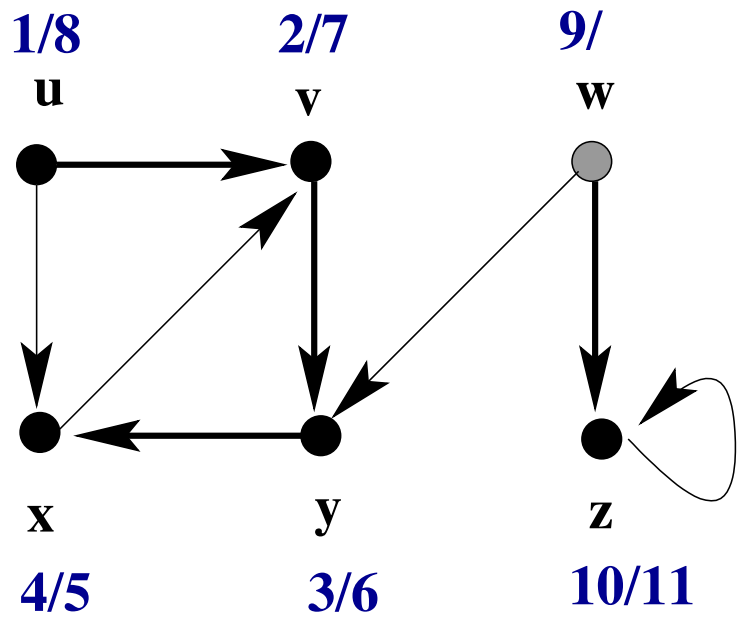
(j)



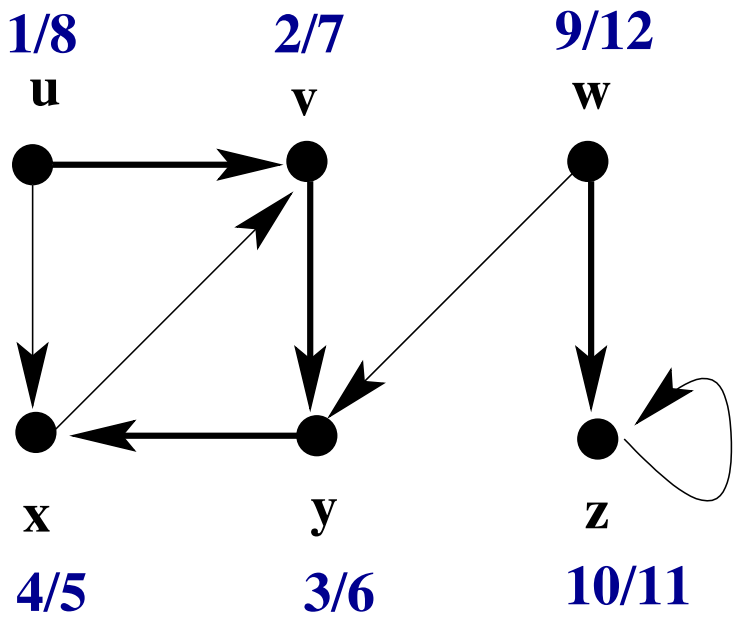
(j)



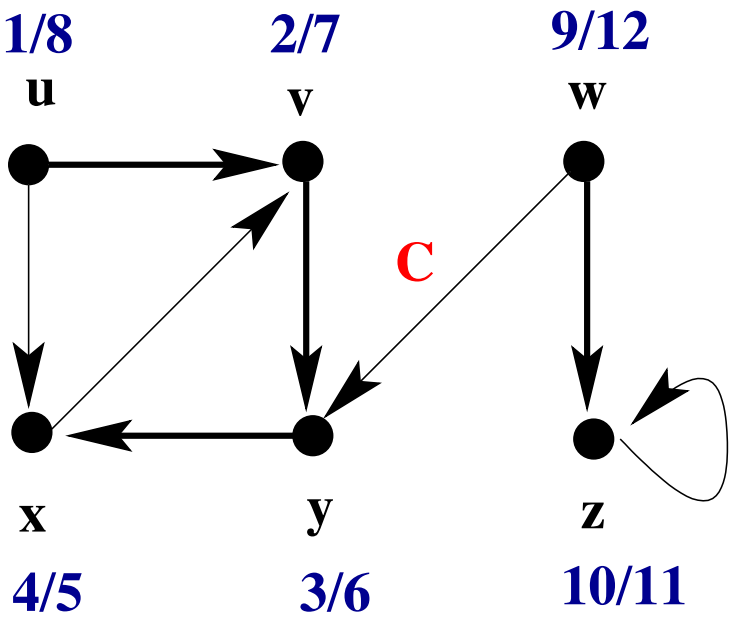
(k)

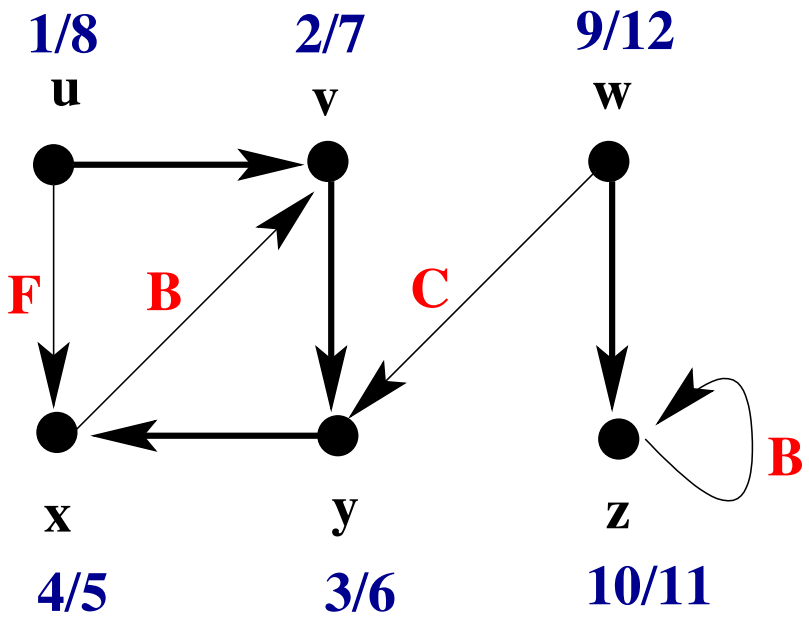


(l)

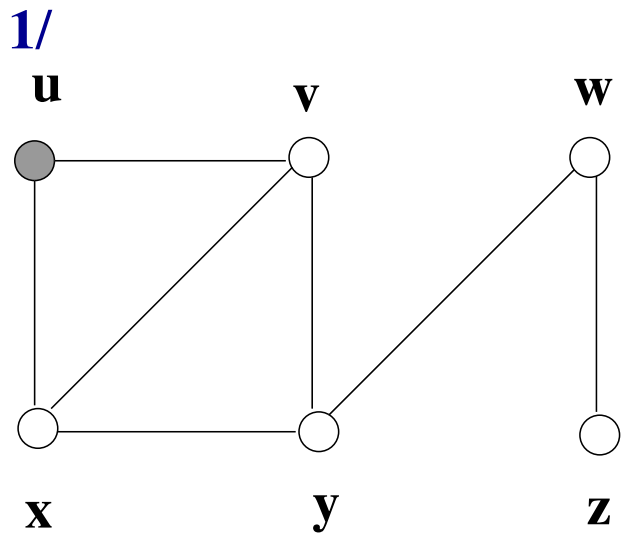


(l)

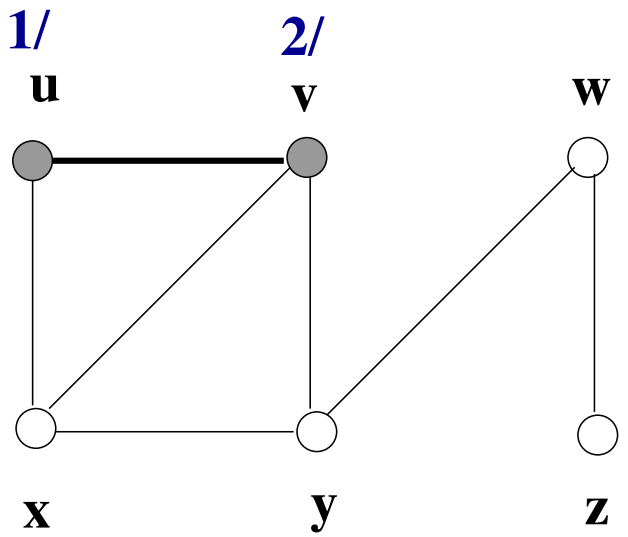



1/8
u
2/7
v
9/12
w
F
B
C
B
x
y
z
4/5
3/6
10/11

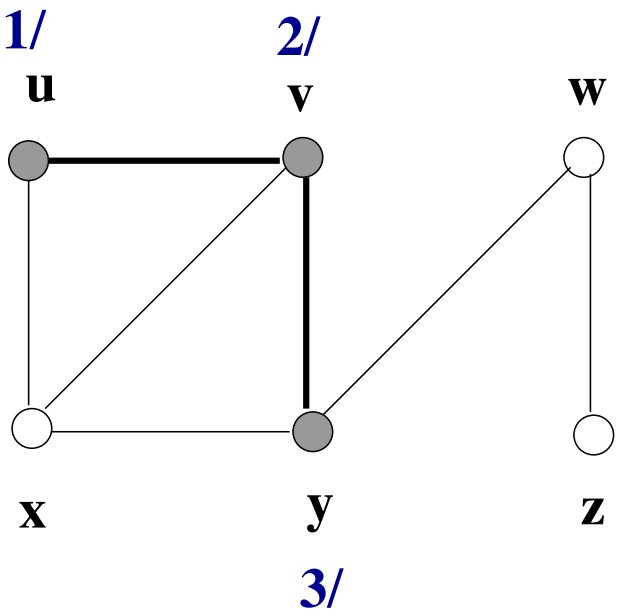
(a)



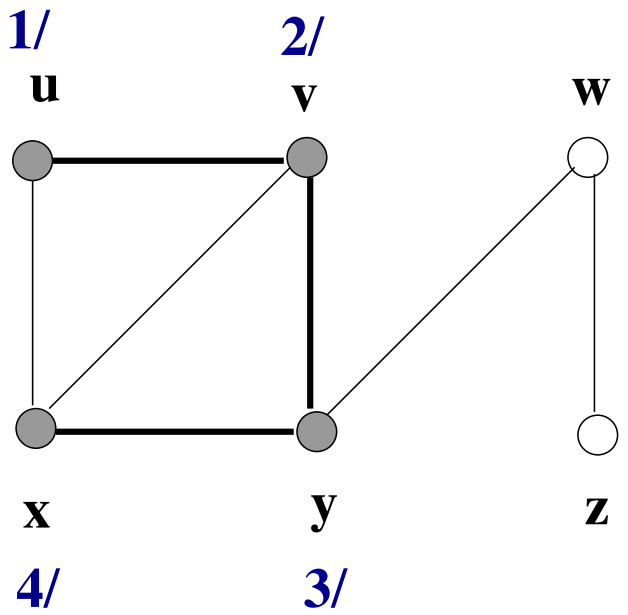
(b)



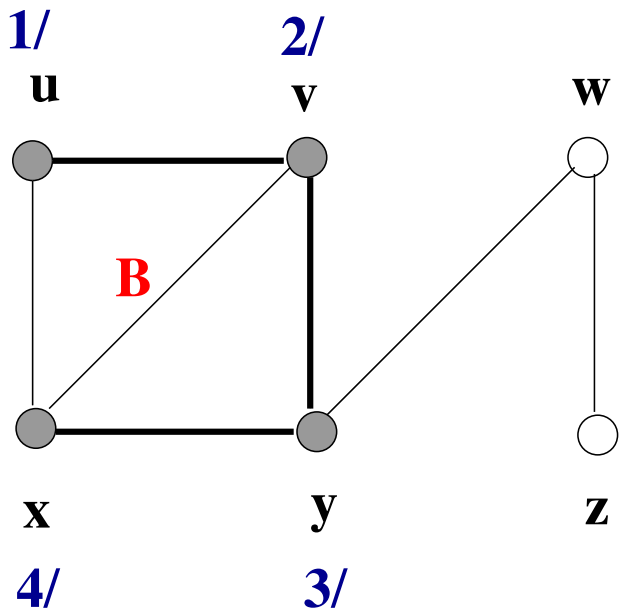
(c)



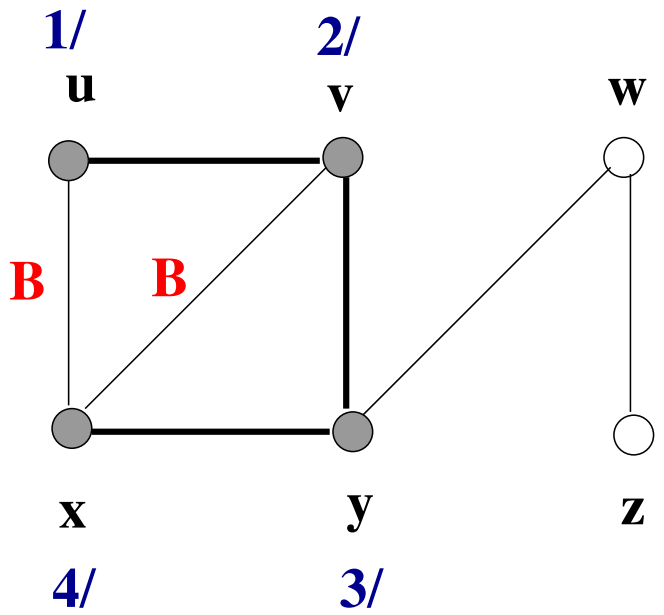
(d)



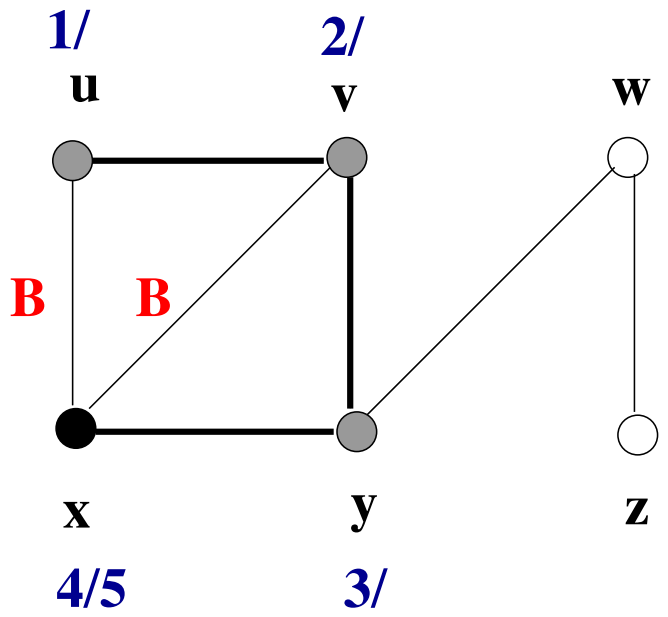
(d)



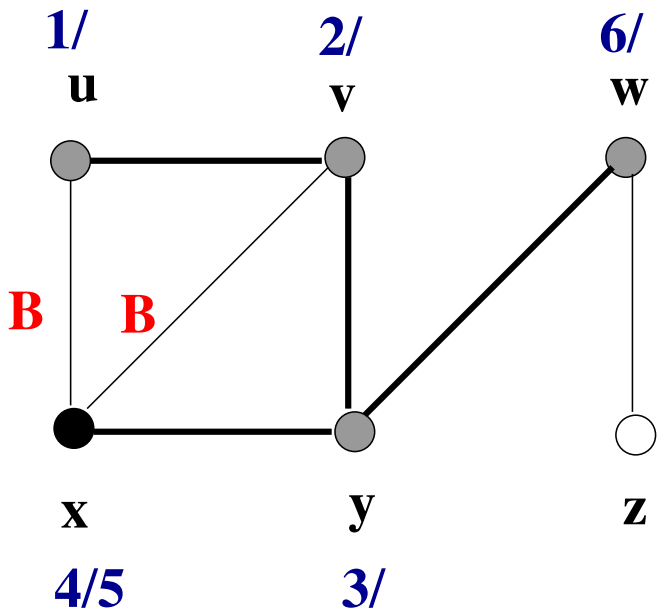
(d)



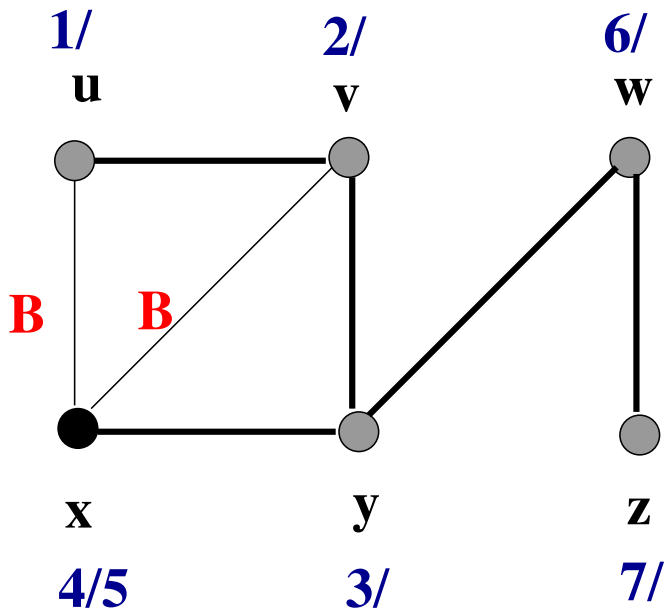
(e)



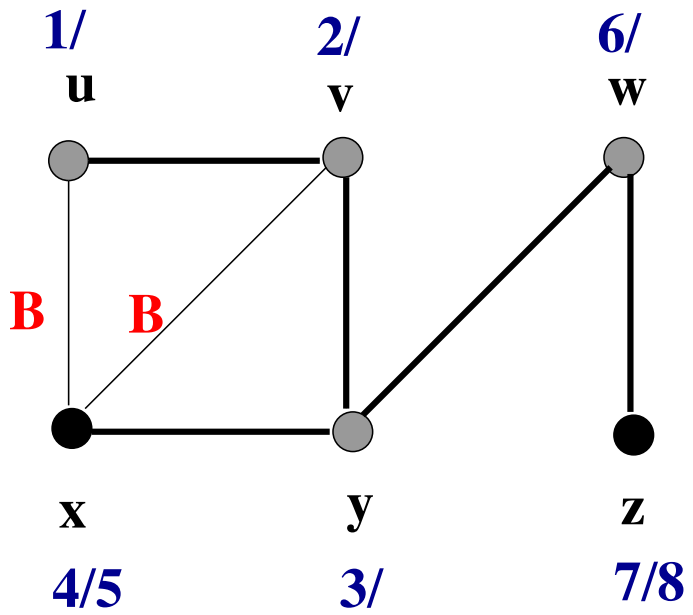
(f)



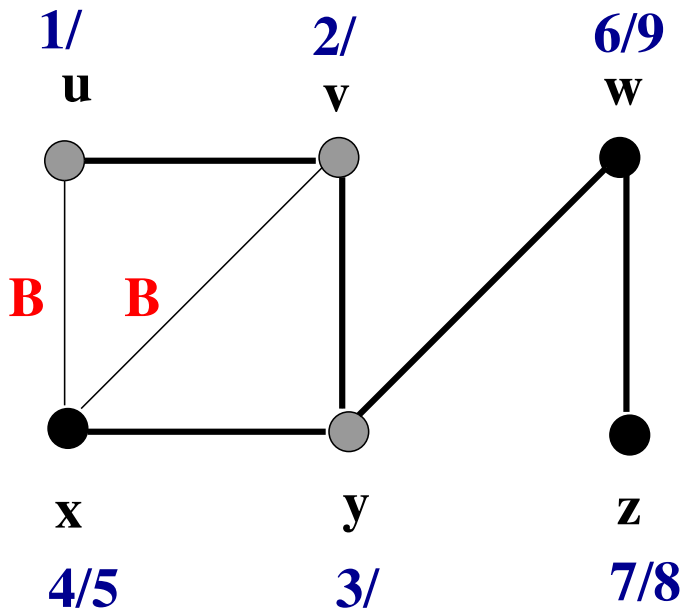
(g)



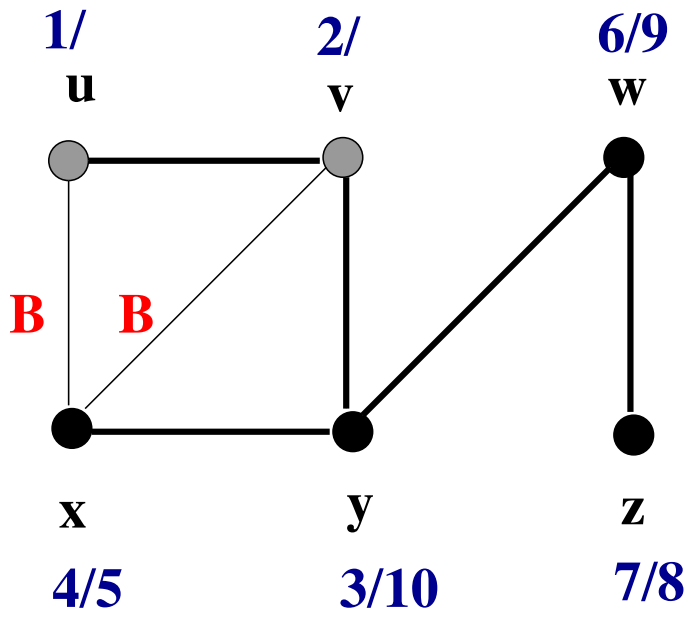
(h)



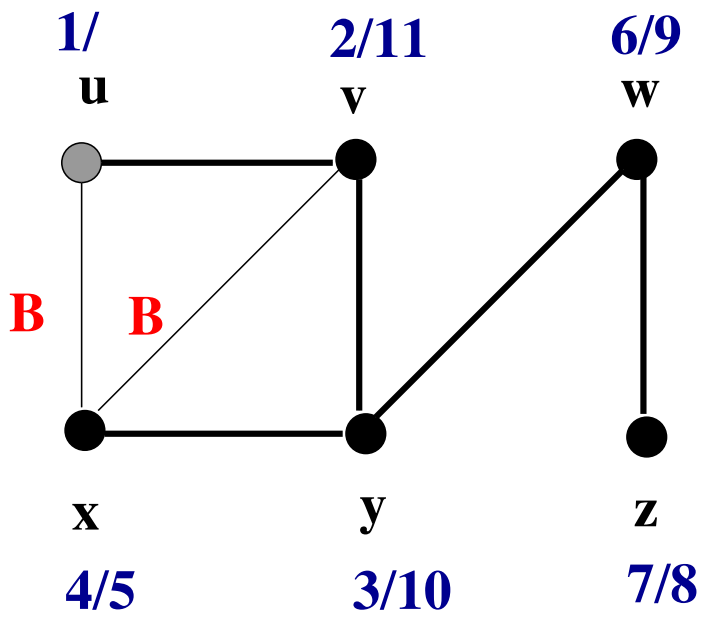
(i)

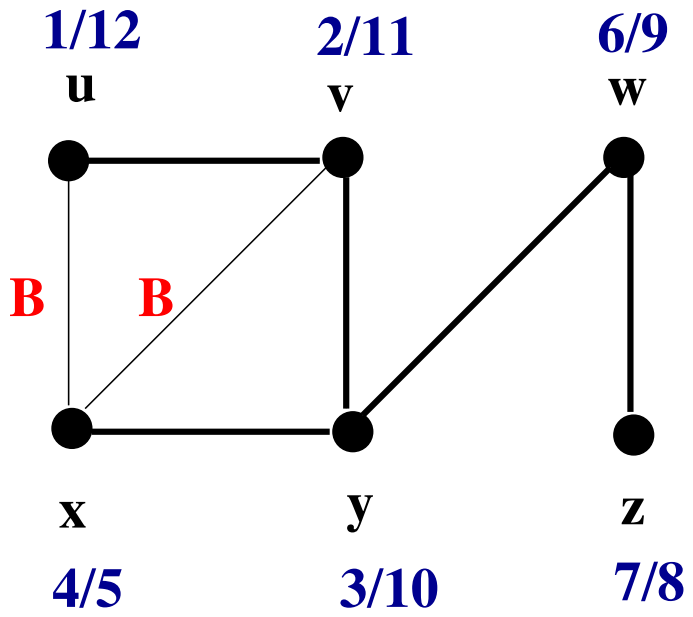


(j)



(k)




1/12
u
2/11
v
6/9
w
B
B
x
y
z
4/5
3/10
7/8

Identificazione di cicli

Teorema. Un grafo G è aciclico se e soltanto se una visita in profondità di G non produce alcun arco di tipo back.