Algoritmi e Strutture Dati

Grafi: Definizione e Algoritmi di visita

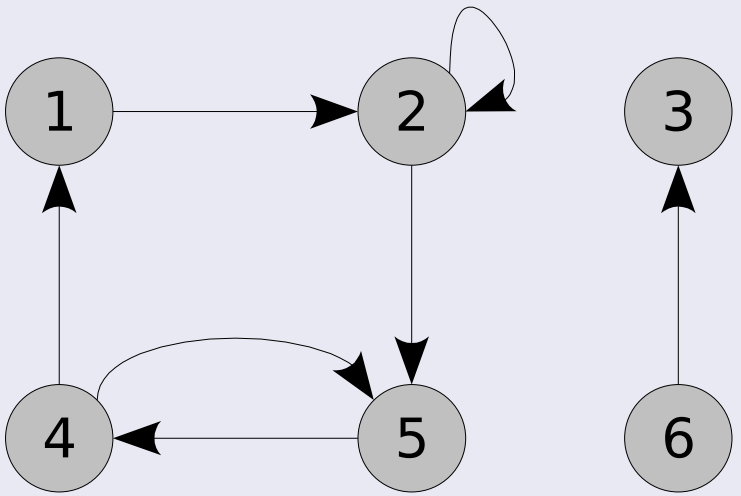
Maria Rita Di Berardini, Emanuela Merelli[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

A.A. 2007/08

Definition (Grafi orientati)

Un **grafo orientato** G è una coppia (V, E) dove V è un insieme finito (non vuoto) di *nodi* (detti anche vertici) ed $E \subseteq V \times V$ è un insieme di coppie ordinate di nodi, detti *archi*

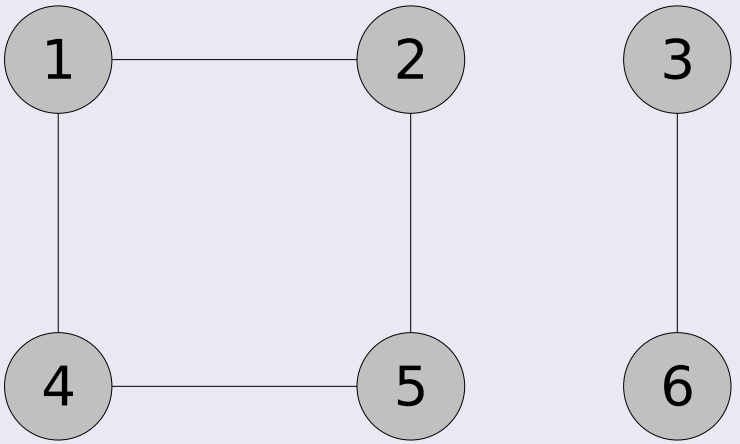


Rappresentazione di un grafo orientato $G = (V, E)$ con insieme di vertici $V = \{1, 2, 3, 4, 5, 6\}$ ed $E = \{(1, 2), (2, 2), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$

Sono ammessi **cappi**, ossia archi che escono ed entrano nello stesso nodo

Definition (Grafi non orientati)

In un **grafo non orientato** $G = (V, E)$ l'insieme di archi E è composto da coppie *non ordinate*, anzichè da coppie ordinate



Rappresentazione di un grafo non orientato $G = (V, E)$ con insieme di vertici $V = \{1, 2, 3, 4, 5, 6\}$ ed $E = \{(1, 2), (2, 2), (2, 5), (4, 1), (6, 3)\}$

Per convenzione, utiliziamo la notazione (u, v) per un arco; le notazioni (u, v) ed (v, u) denotano lo stesso arco. Inoltre in un grafo non orientato non sono ammessi cappi.

Definition (Incidenza e adiacenza)

Se (u, v) è un arco di un *grafo orientato* $G = (V, E)$, diciamo che (u, v) **esce** dal nodo u ed **entra** nel nodo v.

Se (u, v) è un arco di un *grafo non orientato* $G = (V, E)$, diciamo che (u, v) è **incidente** nei nodi u e v.

Se (u, v) è un arco di un *grafo (orientato e non)* $G = (V, E)$, diciamo che il nodo u è **adiacente** al nodo v.

Nota: Per grafi non orientati la relazione di adicenza è simmetrica (ossia se u è adiacente a v, allora anche v è adiacente ad u). Viceversa, per grafi orientati, la relazione di adiacenza non è necessariamente simmetrica (ossia, u adiacente a v non implica necessariamente v adiacente ad u)

Definition (Grado)

Sia $G = (V, E)$ un *grafo non orientato* ed $u \in V$ un nodo di G. Il **grado** di u è il numero di archi che incidono nel nodo

Sia $G = (V, E)$ un *grafo orientato* ed $u \in V$ un nodo di G. Il **grado uscente** di u è il numero di archi che escano dal nodo, mentre il **grado entrante** di u è il numero di archi che entrano nel nodo. Il **grado** di u è la somma del suo grado uscente e del suo grado entrante

Un nodo u si dice **isolato** se il suo grado è zero

Definition (Cammini e cicli)

Un **cammino** di lunghezza k da un nodo u ad un nodo v in un grafo $G = (V, E)$ è una sequenza $\langle v_0, v_1, \ldots, v_k \rangle$ di vertici tali che $u = v_0$, $v = v_k$ e $(v_{i-1}, v_i) \in E$ per $i = 1, 2, \ldots, k$. La lunghezza del cammino è data dal numero di archi nel cammino

Se esiste un cammino p da u a v, diciamo che v è **raggiungibile** da u attraverso p. Questo fatto viene indicato con $u \overset{p}{\rightsquigarrow} v$

Un cammino è **semplice** se tutti i nodi nel cammino sono distinti

Un cammino $\langle v_0, v_1, \ldots, v_k \rangle$ forma un **ciclo** se $v_0 = v_k$ ed il cammino contiene almeno un arco (se la lunghezza del ciclo è 1 allora il ciclo è un cappio). Il ciclo è **semplice** se i vertici $v_1, \ldots, v_k$ (tutti tranne v_0) sono distinti.

Un grafo senza cicli è detto **aciclico**

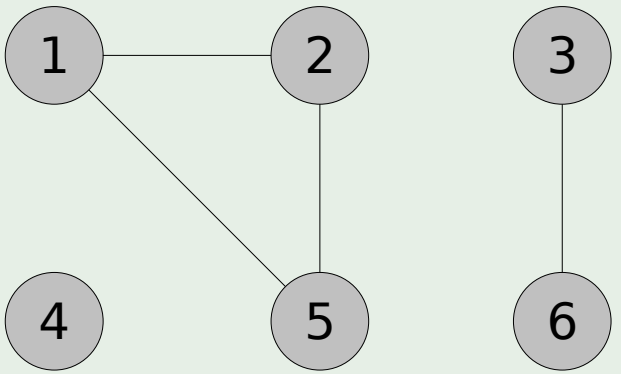
Definition (Grafi connessi)

Un grafo *non* orientato $G = (V, E)$ è **connesso** se ogni coppia di nodi é collegata attraverso un cammino

Le **componenti connesse** di un grafo *non* orientato sono le classi di equivalenza dei vertici secondo la relazione "è raggiungibile da"

Example (Grafi connessi)

Le componenti connesse del seguente grafo sono $\{1, 2, 5\}, \{4\}, \{3, 6\}$



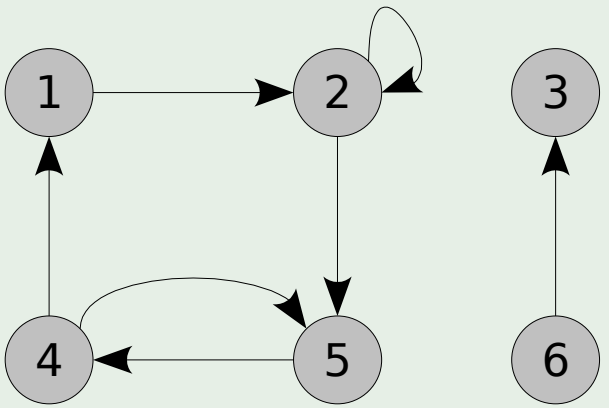
Definition (Grafi fortemente connessi)

Un grafo *orientato* $G = (V, E)$ è **fortemente connesso** se due vertici qualsiasi sono raggiungibili l'uno dall'altro

Le **componenti fortemente connesse** di un grafo *orientato* sono le classi di equivalenza dei vertici secondo la relazione "sono mutuamente raggiungibili"

Example (Grafi fortemente connessi)

Le componenti connesse del seguente grafo sono $\{1, 2, 4, 5\}, \{3\}, \{6\}$



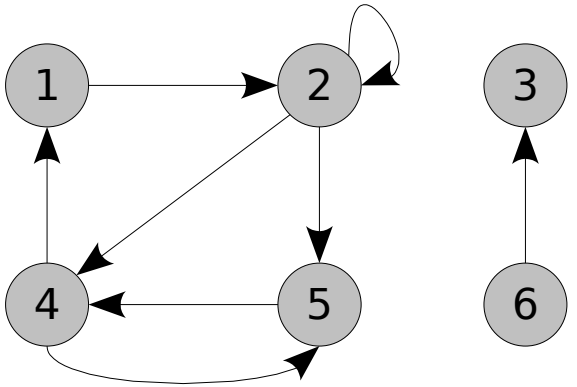
Rappresentazione di grafi

Nella memoria di una macchina un grafo $G = (V, E)$ può essere rappresentato mediante:

- Liste di adiacenza: per ogni nodo $u \in V$ viene costruita la lista dei vertici $v_1, \ldots, v_k$ adiacenti ad u, ossia tali che $u, v_i \in E$ per ogni $i = 1, \ldots, k$
- Matrice di adiacenza: il grafo viene rappresentato mediante una matrice quadrata M di dimensione $n \times n$ (con $n = |V|$, il numero di nodi del grafo) in cui $M_{i,j} = 1$ se $(v_i, v_j) \in E$, $M_{i,j} = 0$ altrimenti

Rappresentazione di grafi

Consideriamo il grafo

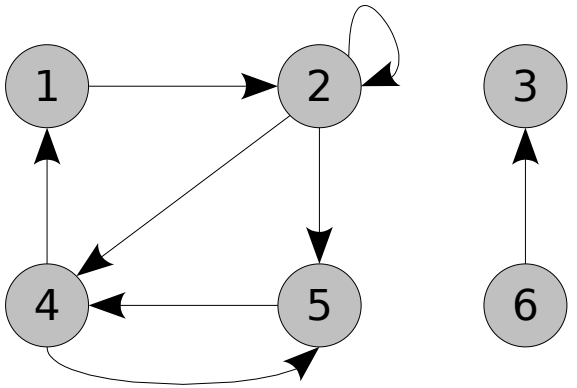


La sua rappresentazione mediante matrice di adiacenza è la seguente

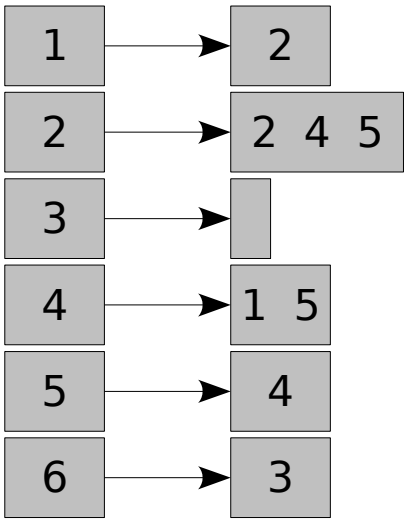
	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

Rappresentazione di grafi

Consideriamo il grafo



La sua rappresentazione mediante liste di adiacenza è la seguente



Rappresentazione di grafi: liste di adiancenza

- Le liste di adiacenza consentono di rappresentare in modo compatto grafi **sparsi**, ossia grafi per cui $|E|$ (il numero di archi) è molto più piccolo di $|V \times V|$
- Se il grafo è orientato, la somma delle lunghezze di tutte le liste di adiacenza è $|E|$, perchè un arco (u, v) viene rappresentato inserendo v in $Adj[u]$ (lista di adiacenza di u)
- Se il grafo è non orientato, la somma delle lunghezze di tutte le liste di adiacenza è $2|E|$, perchè se un arco (u, v) è non orientato, allora v compare nella lista di adiacenza di u e viceversa
- Le liste di adiacenza possono essere adattate per rappresentare **grafi pesati**, cioè grafi per i quali ogni arco ha associato un **peso**. I pesi associati agli archi vengono tipicamente rappresentati mediante una funzione peso $w : E \rightarrow \mathbb{R}$
- Se G è pesato con funzione peso w, il peso $w(u, v)$ dell'arco (u, v) viene memorizzato insieme a v nella lista di adiacenza di u

Rappresentazione di grafi: liste di adiancenza

- L'unico svantaggio è che non esiste un modo veloce per stabile se un certo arco $(u, v) \in E$; possiamo solo cercare v nella lista di adiacenza di u (il che richiede un tempo non necessariamente costante)
- Gli algorimi di visita (in ampiezza ed in profondità) di un grafo assumono una rappresentazione del grafo in input mediante liste di adiacenza

Rappresentazione di grafi: matrici di adiancenza

- La matrice di adiacenza di un grafo richiede una memoria pari a $\Theta(|V|^2)$, indipendentemente dal numero di archi del grafo stesso. Possiamo però stabilire se un certo $(u, v) \in E$ in un tempo costante
- È particolarmente adatta per rappresentare **grafi densi**, ossia grafi in cui $|E|$ è prossimo a $|V \times V|$
- Se G è pesato con funzione peso w, il peso $w(u, v)$ dell'arco (u, v) viene memorizzato nella posizione $M_{u,v}$ della matrice di adiacenza

Parte I

Visite di grafi

Visita in ampiezza (breadth-first search, BFS)

- Dato un grafo $G = (V, E)$ ed un vertice s, detto *sorgente*, ispeziona in maniera sistematica tutti gli archi del grafo per determinare tutti i nodi raggiungibili da s.
- Vengono visitati prima i nodi la cui distanza (intesa come numero minimo di archi) dalla sorgente è 1, poi quelli a distanza 2, 3 e così via.
- L'algoritmo di visita in ampiezza:
  - calcola la distanza da s di ciascun nodo raggiungibile
  - costruisce quello che viene chiamato **breadth-first tree** (BF tree): un albero la cui radice è s e che contiene tutti i nodi raggiungibili da s.
  - per ogni nodo v raggiungibile da s, il cammino nel BF tree corrisponde al cammino minimo (quello con minor numero di archi) da s a v.

Visita in ampiezza (breadth-first search, BFS)

Per tener traccia del lavoro svolto (di quali nodi sono stati visitati e quali no), l'algoritmo colora i nodi di **bianco**, **grigio** e **nero**.

- inizialmente tutti i nodi sono bianchi, poi diventano grigi ed infine neri
- un nodo viene **scoperto** quando viene incontrato per la prima volta durante una visita. Da quel momento in poi cessa di essere un nodo bianco e diventa grigio
- un generico nodo v diventa nero solo quando tutti i nodi adiacenti a v sono stati scoperti (e quindi sono diventati grigi)

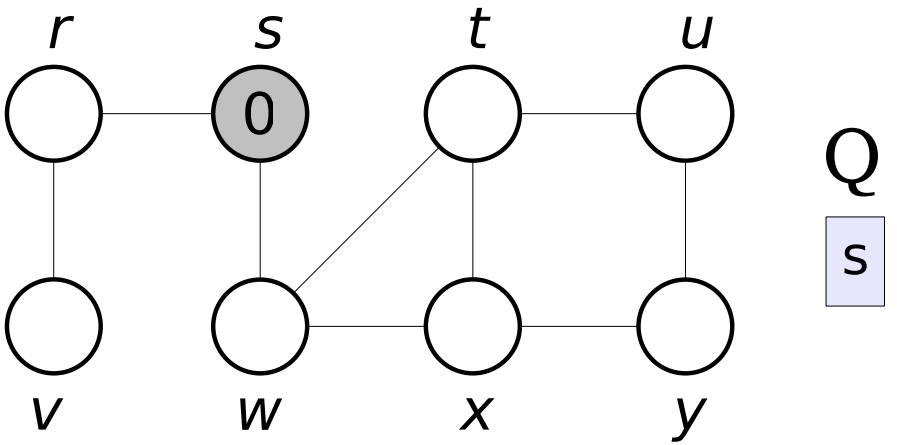
Visita in ampiezza (breadth-first search, BFS)

- Quando un nodo v viene scoperto durante l'ispezione della lista di adiacenza di un dato nodo u, il vertice v e l'arco (u, v) sono aggiunti nel BF tree
- Il nodo u è detto **predecessore** o **padre** di v nel BF tree. Poichè un nodo viene scoperto una sola volta, esso ha al più un padre.
- L'insieme dei nodi scoperti durante una visita viene gestito mediante una coda.

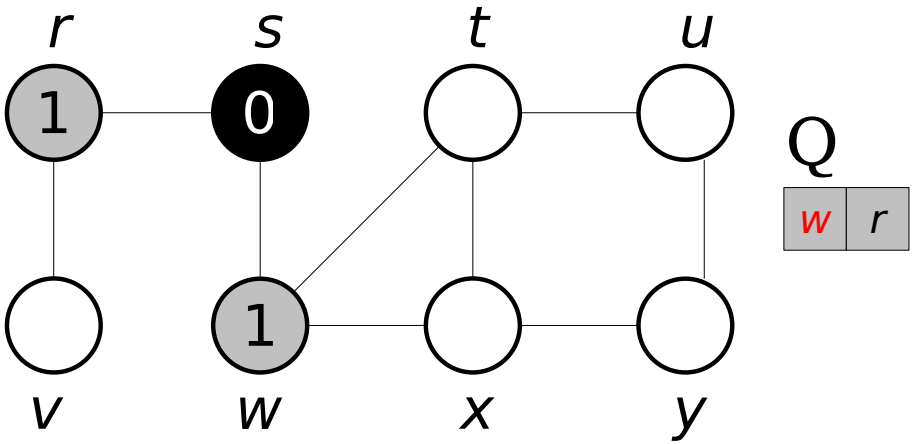
Visita in ampiezza - l'algoritmo

```
     BFS(G, s)
1    for each nodo u ∈ V(G) − {s}
2         color[u] ← WHITE
3         d[u] ← ∞                          ▷ la distanza di u da s
4         π[u] ← ∞                          ▷ il padre di u nel BF tree
5    color[s] ← GRAY
6    d[u] ← 0
7    π[u] ← NIL                             ▷ s è la radice del BF tree
8    Q ← ∅
9    ENQUEUE(Q, s)                          ▷ Q è la coda dei nodi scoperti
10   while Q ≠ ∅
11        do u ← DEQUEUE(Q)
12        for each v ∈ Adj[u]
13             do if color[v] = WHITE
14                  then color[v] ← GRAY
15                       d[v] ← d[u] + 1
16                       π[v] ← u
17                       ENQUEUE(Q, v)
18             color[u] ← BLACK
```

Visita in Ampiezza – un esempio

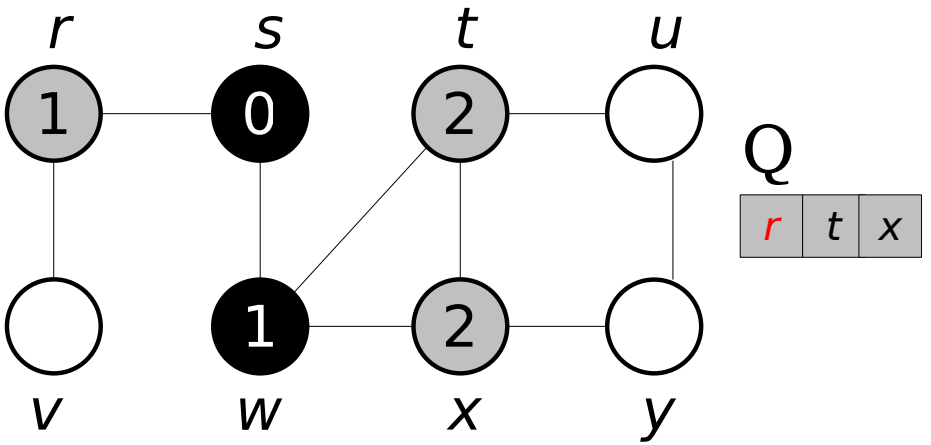


(a)

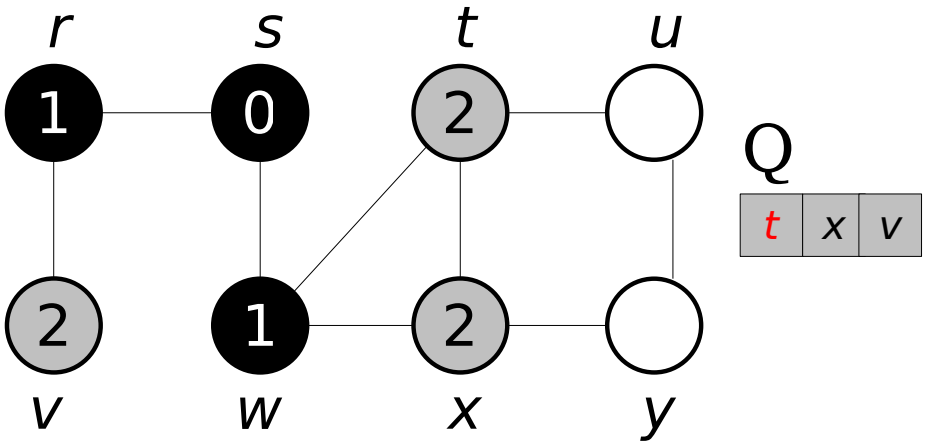


(b)

Visita in Ampiezza – un esempio

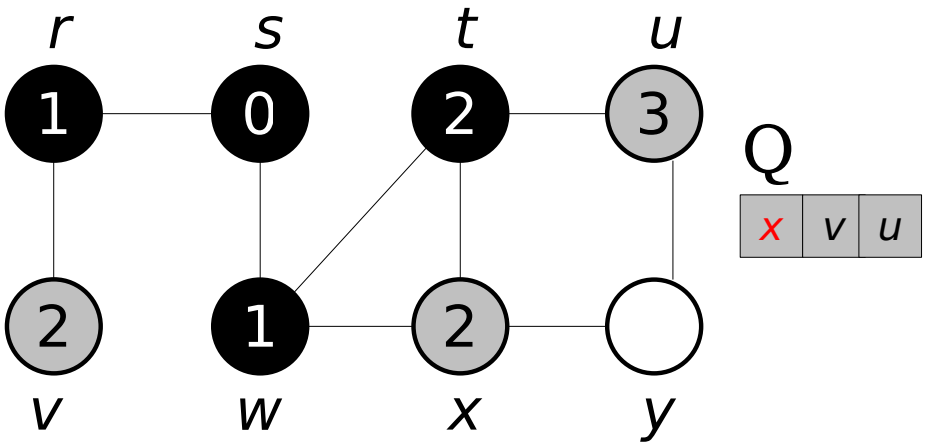


(c)

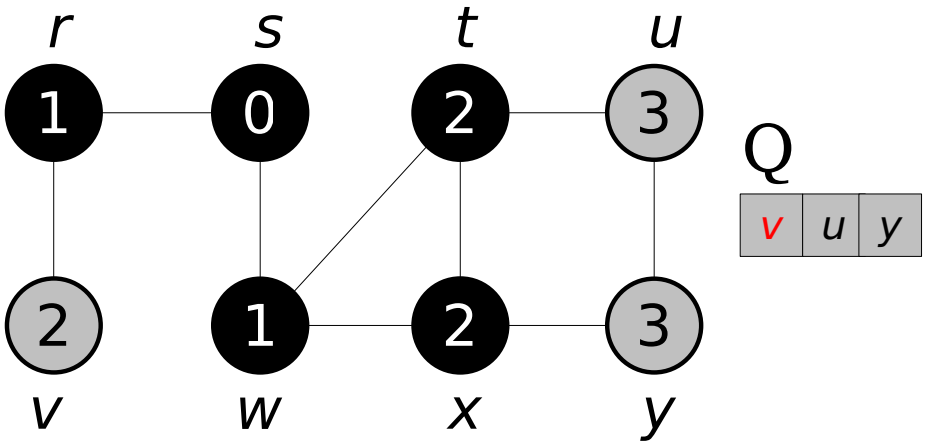


(d)

Visita in Ampiezza – un esempio

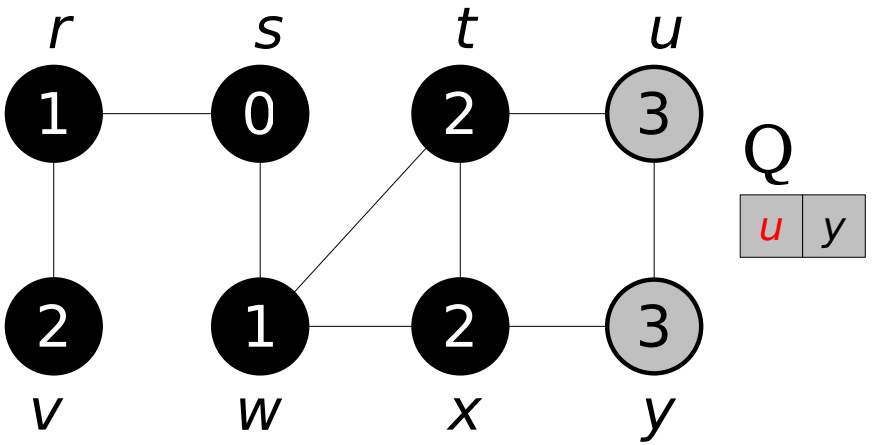


(e)

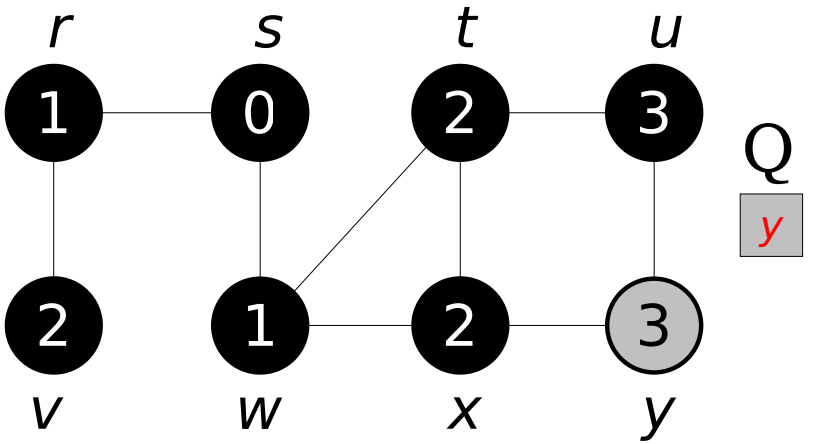


(f)

Visita in Ampiezza – un esempio

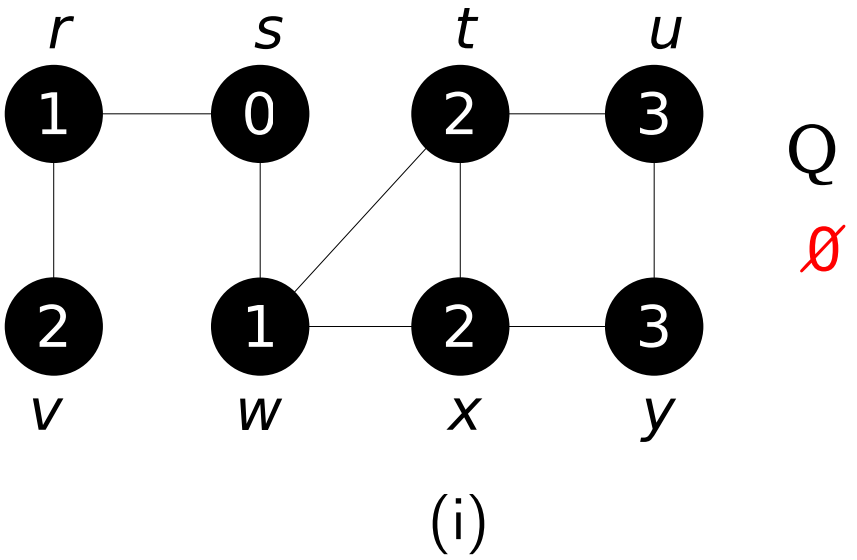


(g)



(h)

Visita in Ampiezza – un esempio



Visita in Ampiezza – analisi

Sia $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. Determiniamo il costo di BFS(G, s) il metodo di aggregazione. Il costo è determinato da:

- costo della fase di inizializzazione (righe 1-9) ossia $O(n)$
- dal costo del ciclo **while** di riga 10. Durante questo ciclo eseguiamo delle operazioni sulla coda (enqueue, dequeue) + una scansione della lista di adiacenza di ogni nodo u estratto dalla coda
- le operazioni di inserimento e cancellazione dalla coda richiedono un tempo $O(1)$. Ogni nodo viene inserito ed estratto dalla coda solo una volta. Il costo complessivo della gestione della coda è $O(n)$
- La lista di adiacenza di generico un nodo viene ispezionata al più una volta (quando il nodo viene estratto dalla coda).
- Poichè la somma delle lunghezze di tutte le liste di adiacenza è $\Theta(m)$ il tempo necessario per ispezionare tutte le liste di adiacenza è $O(m)$

Il costo dell'algoritmo è $O(n) + O(n) + O(m) = O(n + m)$

Visita in Profondità (depth-first search, DFS)

La strategia adottata da questo schema di visita consiste nel visitare il grafo sempre più in "profondità" (fin quando è possibile)

Si comincia con il visitare un nodo qualsiasi v che viene marcato come "scoperto"

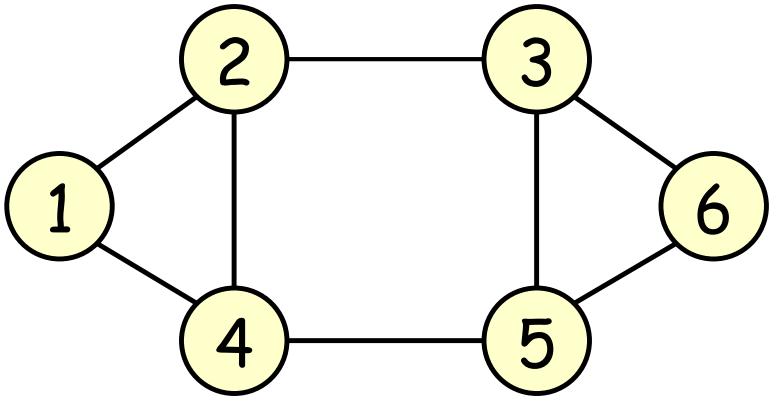
La visita prosegue ispezionando tutti gli archi uscenti dal nodo corrente v alla ricerca di un nodo w collegato ad v e non ancora visitato

Se tale nodo esiste, diventa il prossimo nodo corrente (viene marcato come "scoperto" e si ripente il passo precedente)

Prima o poi, si arriva ad un punto in cui tutti gli archi incidenti il nodo corrente v portano a nodi visitati

Visita in Profondità (depth-first search, DFS)

In questo caso siamo costretti a fare un passo indietro ("backtrack"), tornando al nodo u visitato prima di v; u diventa il nodo corrente e si ripetono i passi precedenti



Lo schema di visita in profondità (a partire dal nodo 1) visita, nell'ordine 1, 2, 3, 6, 5, 4

Visita in Profondità (depth-first search, DFS)

Proprio come la visita in ampiezza, colora i nodi per distinguere i nodi non ancora scoperti (WHITE), da quelli scoperti ma non espansi (GRAY) e da quelli espansi (BLACK)

Di nuovo ad ogni nodo v viene associato un precedessore denotato con $\pi[v]$, è il nodo che ci consente di raggiungere v durante la visita

Inoltre, ogni nodo v ha associate due informazioni temporali:

- il tempo di scoperta $d[v]$ registra il momento in cui il nodo viene scoperto (e diventa grigio)
- il tempo di completamento $f[v]$ registra il momento in cui la visita completa l'ispezione della lista di adiacenza di v (ed il nodo diventa nero)

Per ogni nodo v abbiamo che $d[v] < f[v]$; inoltre: v è WHITE prima di $d[u]$, è GRAY tra $d[u]$ e $f[u]$, e BLACK successivamente

Visita in Profondità (depth-first search, DFS)

DFS(G)

1. **foreach** nodo $u \in V[G]$
2. **do** $color[u] \leftarrow WHITE$
3. $\pi[u] \leftarrow NIL$
4. $time \leftarrow 0$

▷ time è una variabile globale usata per calcolare i tempi $d[u]$ e $f[u]$

5. **foreach** nodo $u \in V[G]$
6. **do if** $color[u] = WHITE$
7. **then DFS-visit**(u)

N.B: Il risultato della visita potrebbe dipendere dall'ordine con cui i nodi vengono ispezionati nella riga 5

In realtà, queste differenze nell'ordine non causano problemi, perchè tutti i possibili risultati della visita sono sostanzialmente equivalenti

Visita in Profondità (depth-first search, DFS)

DFS-visit(u)

1. $color[u] \leftarrow GRAY$
2. $time \leftarrow time + 1$
3. $d[u] \leftarrow time$
4. **foreach** nodo $v \in Adj[u]$
5. **do if** $color[v] = WHITE$ ▷ ispezioniamo l'arco (u, v)
6. **then** $\pi[v] \leftarrow u$
7. **DFS-visit**(v)
8. $color[u] \leftarrow BLACK$
9. $time \leftarrow time + 1$
10. $f[u] \leftarrow time$

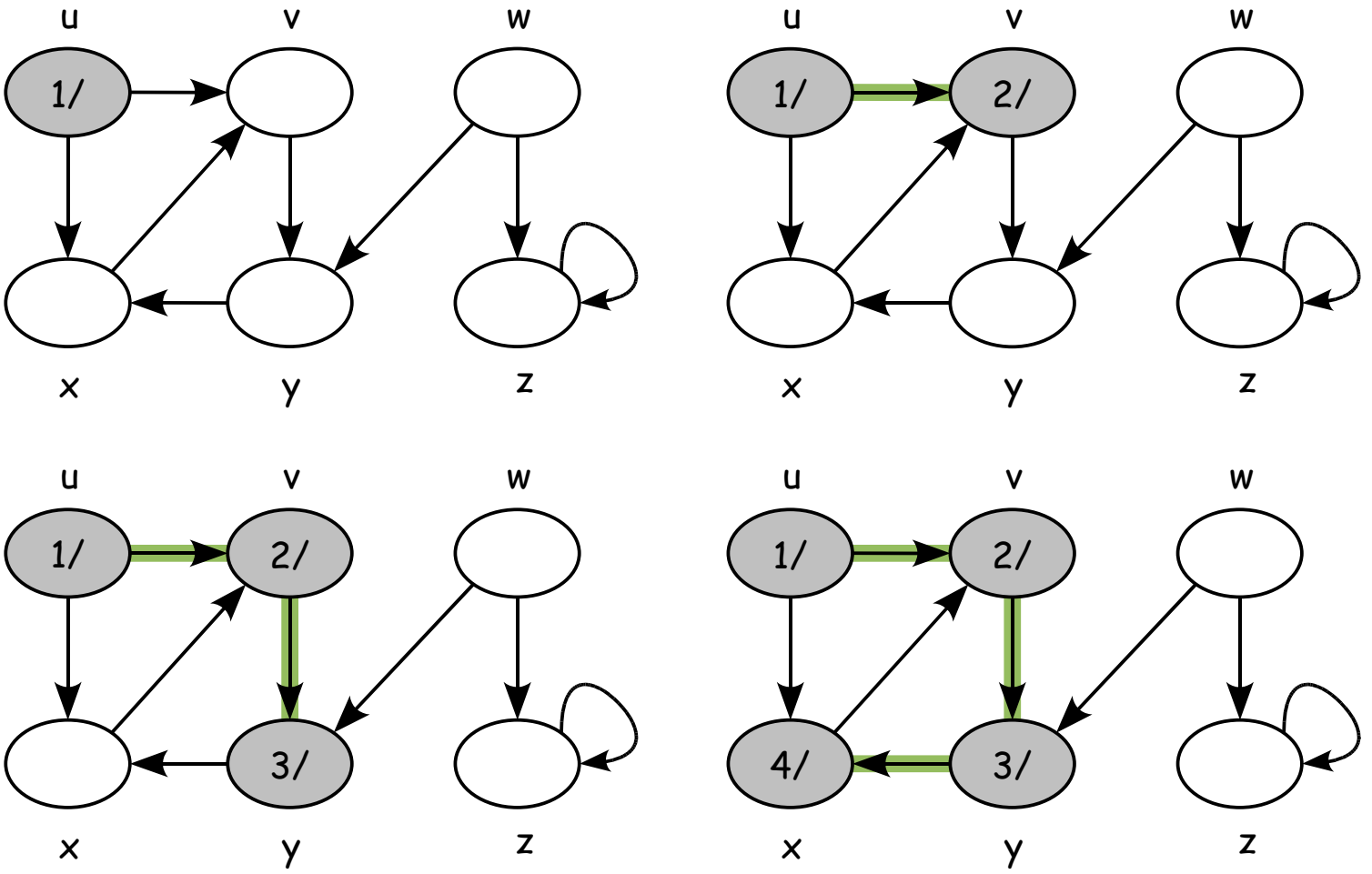
Visita in Profondità (depth-first search, DFS)

Nella fase di inizializzazione, tutti i nodi vengono colorati di bianco ed il loro predecessore viene settato a NIL; la variabile globale *time* viene settato a zero

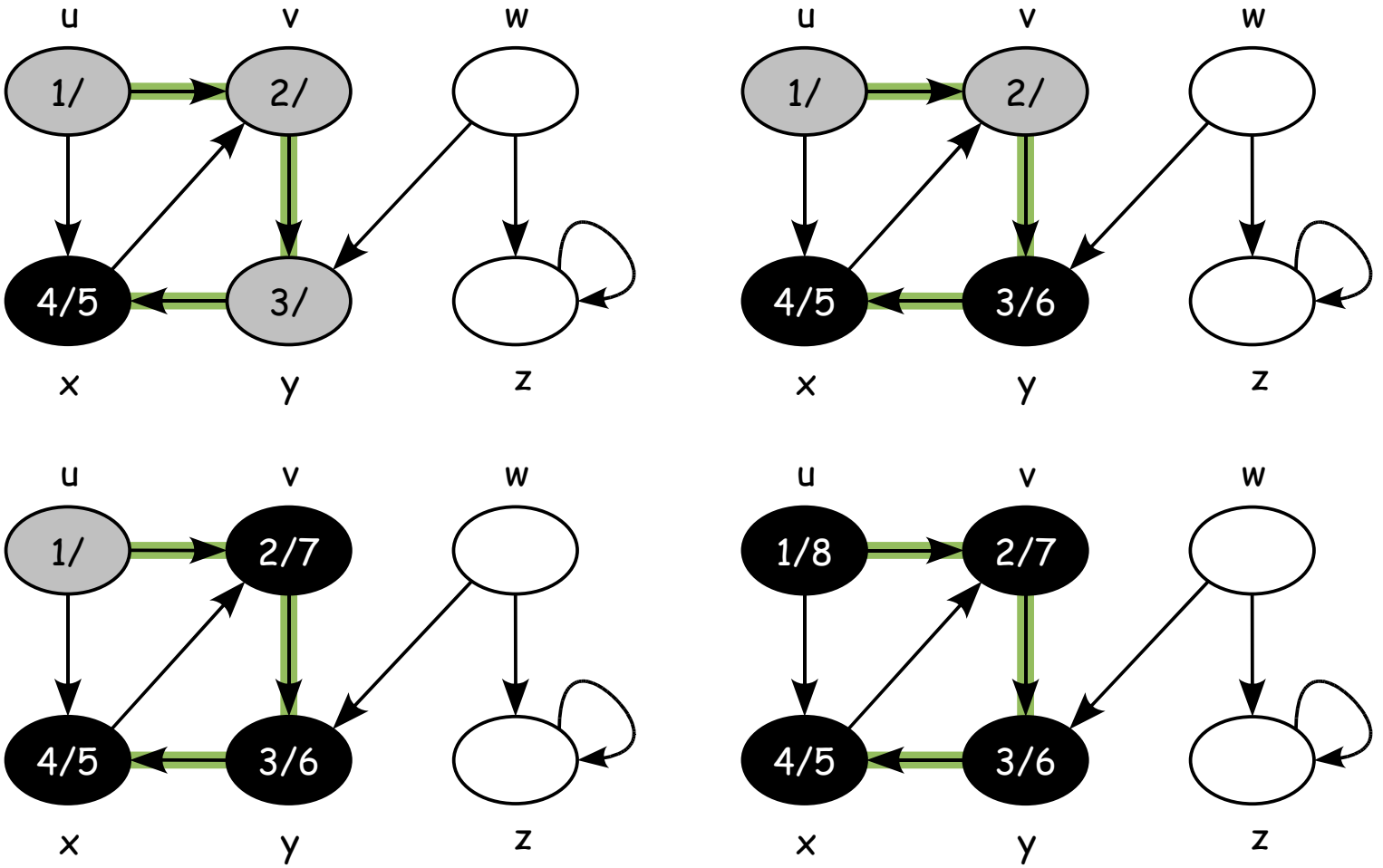
DFS-visit(u) visita (in profondità) tutti i nodi raggiungibili da u

Quando DFS finisce, ogni vertice u ha associato un tempo di scoperta $d[u]$ e un tempo di completamento $f[u]$

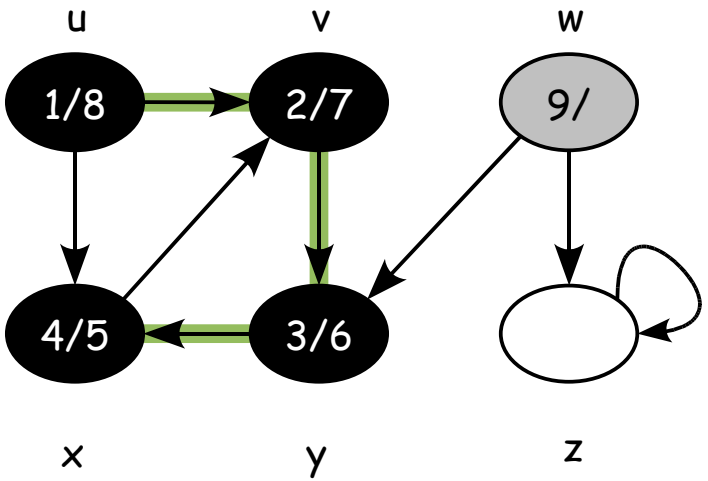
Visita in Profondità – un esempio

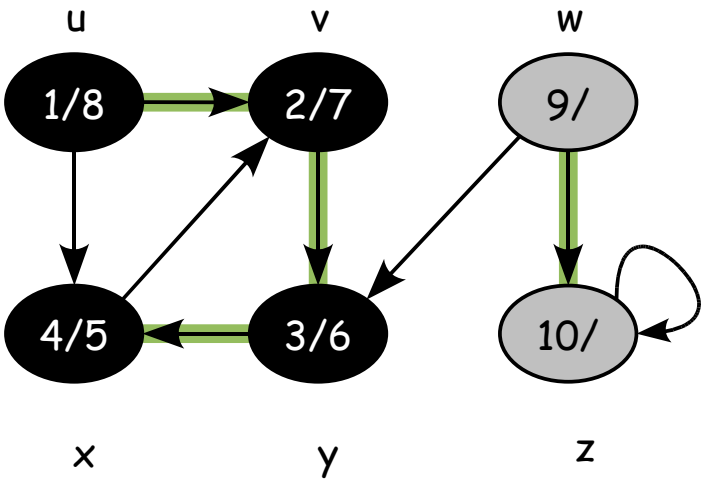


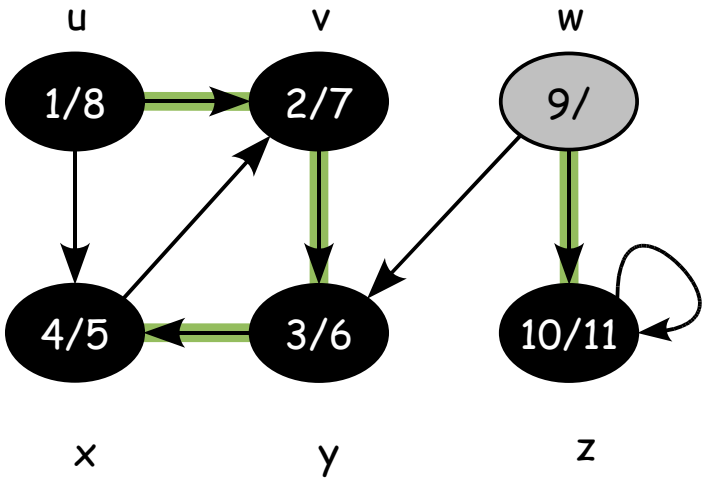
Visita in Profondità – un esempio

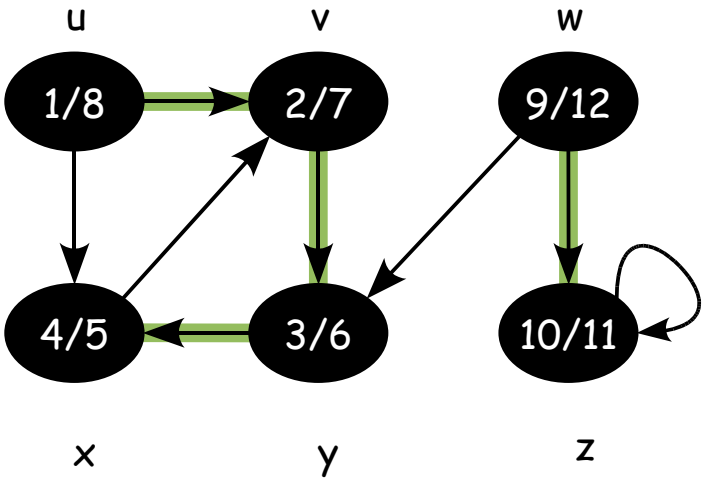


Visita in Profondità – un esempio









Visita in Profondità – complessità

Quale è il tempo di esecuzione di **DFS**? Il ciclo di righe 1-3 impiegano un tempo $O(n)$. Quanto ci costa eseguire ciascun chiamata di **DFS-Visit**?

La procedura **DFS-Visit** viene eseguita esattamente una volta per ogni vertice $v \in V$ (viene invoca solo se il colore è WHITE e la prima cosa che fa è cambiare il colore degli archi in GRAY)

Durante un'esecuzione di **DFS-Visit**, il ciclo delle righe 4-7 viene eseguito $|Adj[u]|$ volte. Poichè

$$\sum_{v \in V} |Adj[u]| = O(m)$$

dove $m = |E|$, il costo totale per le chiamate di **DFS-Visit** è $O(m)$.
Quindi il costo totale dell'algoritmo è $O(n + m)$

Sottografo dei predecessori

Il sottografo dei predecessori è definito in modo leggermente diverso da BFS

$$\begin{aligned} G_\pi &= (V, E_\pi) \\ E_\pi &= \{(\pi[v], v) \ : \ v \in V \text{ and } \pi[v] \neq NIL\} \end{aligned}$$

G_π forma una **foresta** depth-first perchè, in generale, è composta da vari alberi depth-first