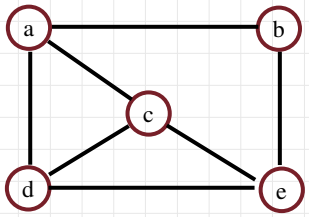
Visita di grafi

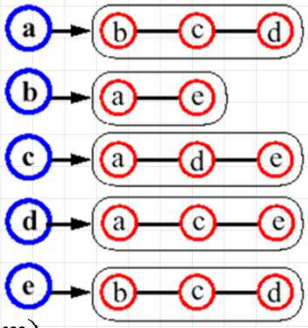
Vittorio Maniezzo - Universita di Bologna

Liste di adiacenza

Lista di adiacenza di un vertice v: lista che concatena tutti i vertici adiacenti a v

Il grafo può essere rappresentato dalle liste di adiacenza di tutti i suoi vertici

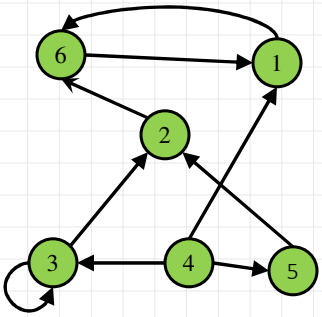




Spazio $= \Theta(n + \sum_v \deg(v)) = \Theta(n + m)$

Vittorio Maniezzo - Universita di Bologna

Liste di adiacenza



1	6
2	6
3	2 → 3
4	1 → 3 → 5
5	2
6	1

3

Matrice di adiacenza

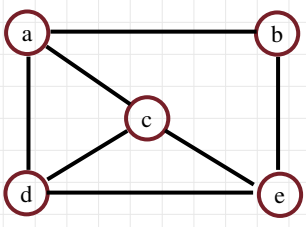
Una matrice di adiacenza è una matrice M di variabili booleane con una cella per ogni coppia di vertici

$\mathrm{M}[i,j]$ = vero (oppure 1) – c'è l'arco (i,j) nel grafo

$\mathrm{M}[i,j]$ = falso (oppure 0) – non c'è l'arco (i,j) nel grafo

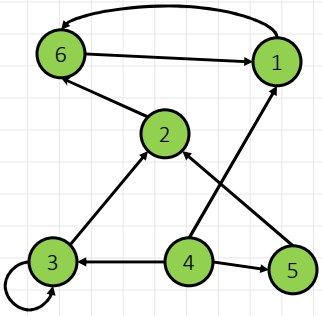
Spazio = $\Theta(n^2)$

	a	b	c	d	e
a	F	T	T	T	F
b	T	F	F	F	T
c	T	F	F	T	T
d	T	F	T	F	T
e	F	T	T	T	F



4

Matrice di adiacenza



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

5

Algoritmi di ricerca su grafo

Ricerca sistematica di archi o vertici di un grafo

Il grafo G = (V,E) può essere diretto o non diretto

Negli algoritmi nel seguito si assume una rappresentazione a lista di adiacenza

Applicazioni

- Compilatori
- Grafica
- Mappe digitali
- Reti: routing, clustering, localizzazione, ecc.
- …

6

Attraversamento di grafo

Graph traversal, definizione del problema:

Dato un grafo G=(V, E) ed un vertice r ∈ V (detto sorgente o radice), si richiede:

- visitare ogni vertice raggiungibile nel grafo dal vertice r
- ogni nodo deve essere visitato una volta sola

Visita in ampiezza (*breadth-first search BFS*)

- Visita i nodi espandendo una frontiera fra nodi scoperti / da scoprire cercando di restare il più possibile vicini alla radice

Visita in profondità (*depth-first search DFS*)

- Visita i nodi andando subito il più lontano possibile nel grafo

7

Ricerca per ampiezza: Breadth First

Una ricerca per ampiezza (Breadth-First Search, BFS) percorre una *componente connessa* di un grafo e facendolo definisce un *albero di copertura* con molte proprietà utili

Dato un vertice sorgente s, BFS calcola la *distanza* (*minimo numero di archi*) da s a ogni vertice raggiungibile.

Al vertice iniziale *s* viene associata una distanza di 0.

Nella prima mossa vengono visitati (scoperti) tutti i nodi raggiungibili dall'origine percorrendo un solo arco, e viene assegnata loro una distanza di 1

Ogni vertice ha un colore associato (bianco = non scoperto, grigio = scoperto ma non espanso, nero = espanso).

8

Ricerca per ampiezza: Breadth First

Al passo successivo vengono visitati tutti i nodi raggiungibili dall'origine percorrendo due archi, e viene assegnata loro la distanza 2

Si continua così finché non è associata un'etichetta (un livello) ad ogni vertice

Ogni vertice v viene etichettato con la lunghezza d del cammino più breve (nel numero di archi) da s a v e con l'id π(v) del vertice predecessore, da cui si proveniva quando si è scoperto v.

I vertici scoperti ma non espansi sono mantenuti in una *coda* Q.

9

Algoritmo BFS

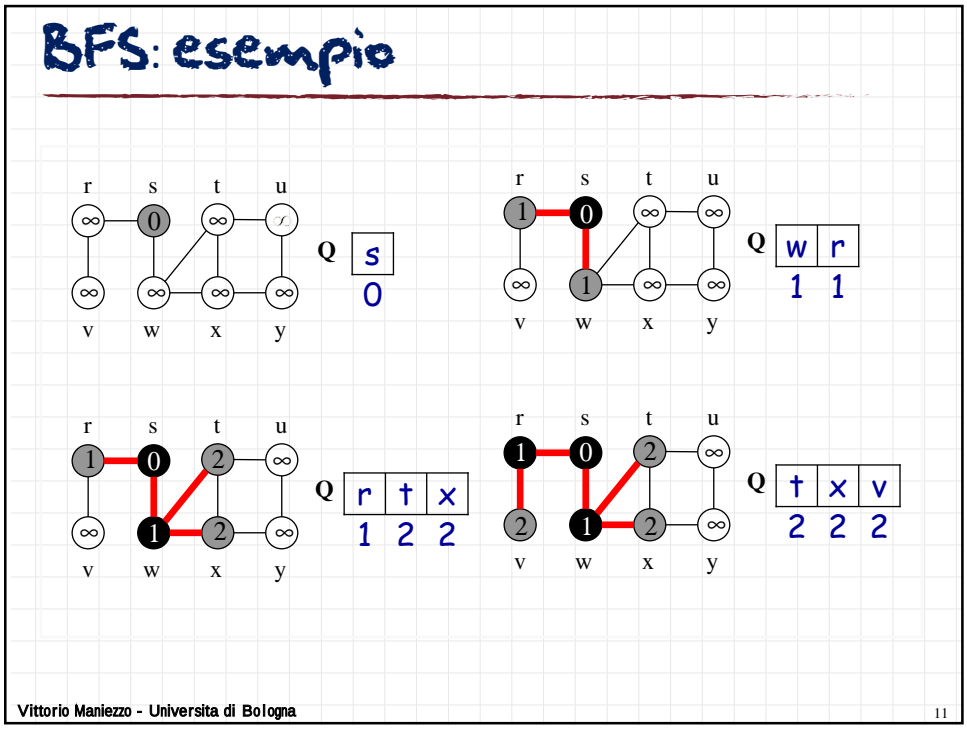
```
foreach vertice u ∈ V[G]−{s} do
   color[u] = white
   d[u] = ∞
   π[u] = NIL
color[s] = gray
d[s] = 0
π[s] = NIL
Q = {s}
while Q ≠∅ do
   u = head[Q]
   foreach v ∈ Adj[u] do
      if color[v] == white
      then color[v] = gray
           d[v] = d[u] + 1
           π[v] = u
           Enqueue(Q,v)
   Dequeue(Q)
   color[u] = black
```

Inizializza tutti i vertici

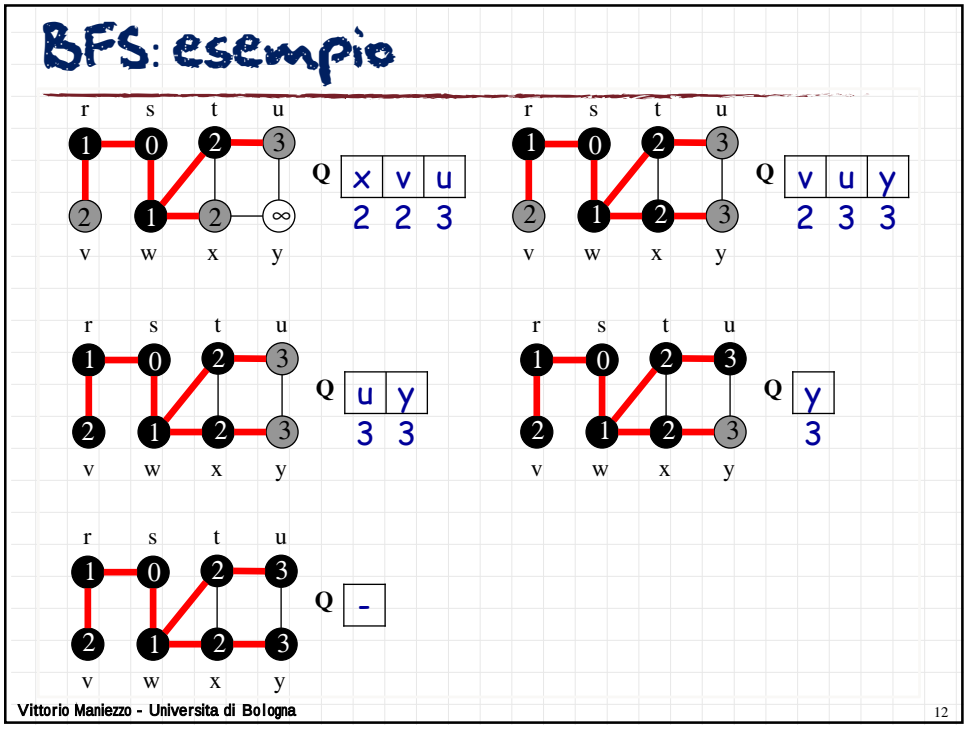
Inizializza BFS con *s*

Gestisci tutti i figli di *u* prima di passare ai figli dei figli

10


BFS: esempio
r s t u
v w x y
Q s
0
Q w r
1 1
Q r t x
1 2 2
Q t x v
2 2 2
Vittorio Maniezzo - Universita di Bologna
11

11


BFS: esempio
r s t u
v w x y
Q x v u
2 2 3
Q v u y
2 3 3
Q u y
3 3
Q y
3
Q -
Vittorio Maniezzo - Universita di Bologna
12

12

Analisi

Dato un grafo G = (V,E)

- I vertici vengono accodati se il loro colore è bianco
- Assumendo che un accodamento / rimozione richieda tempo O(1) il costo totale di questa operazione è O(V)
- La lista di adiacenza di un vertice viene percorsa quando il vertice viene rimosso dalla lista
- La somma delle lunghezze di tutte le liste è Θ(E). Quindi è richiesto un tempo O(E) per percorrerle tutte
- L'inizializzazione dell'algoritmo richiede O(V)

Tempo totale di CPU O(V+E) (lineare nella dimensione della rappresentazione del grafo con liste di adiacenza)

BFS: Proprietá

- Dato un grafo G = (V,E), la BFS scopre tutti i vertici raggiungibili da un vertice origine *s*
- Calcola la distanza minima (in numero di archi) per ogni vertice raggiungibile
- Calcola un albero breadth-first che contiene tutti i vertici raggiungibili
- Per ogni vertice *v* raggiungibile da *s*, il cammino nell'albero breadth first da s a v corrisponde ad un cammino minimo in G

Alberi BFS

Sottografo dei predecessori di G

$$G_\pi = (V_\pi, E_\pi)$$
$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$$
$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

Se applicata ad un grafo G, la procedure BFS costruisce π in modo tale che il sottografo dei predecessori G_π è un albero breadth-first

- V_π consiste di tutti i vertici raggiungibili da s (componente connessa),
- per ogni $v \in V_\pi$, c'è un unico cammino elementare in G_π da *s* a *v* ed è anche il cammino minimo da *s* a *v* in G

Gli archi in G_π sono detti "archi dell'albero" (tree edges)

Ricerca in profondità: Depth-First

La ricerca in profondità (depth-first search, DFS) in un grafo non diretto G si basa sullo schema seguente:

- Si inizia da un vertice *u* qualsiasi, etichettandolo "visitato (scoperto)". Si etichetta *u* come vertice corrente
- Si percorre uno qualsiasi degli archi (*u*,*v*), $v \in Adj(u)$.
- Se l'arco (*u*,*v*) porta ad un vertice *v* già visitato, si ritorna in *u*
- Se il vertice *v* non è ancora stato visitato, diventa il vertice corrente, lo si etichetta "visitato" e si ripete il passo precedente
- In questo modo si visita la componente connessa raggiungibile da *u*.

Ricerca in profondità: Depth-First

Prima o poi, si arriva ad un punto in cui **tutti gli archi incidenti su *u* portano a vertici visitati.**

Allora si attua un backtrack ritornando al vertice *v* visitato prima di *u*. Il vertice *v* diventa il vertice corrente e si ripetono i passi precedenti.

Quando anche tutti i vertici incidenti su *v* portano a vertici visitati, si effettua un altro backtrack come prima. Si **continua a effettuare backtrack lungo il cammino percorso**, esplorando archi che portano a vertici inesplorati e ripetendo la procedura.

17

Algoritmo DFS

```
DFS(G)
 foreach vertice u ∈ V[G] do
    color[u] = white
    π[u] = NIL
 time = 0
 foreach vertice u ∈ V[G] do
    if color[u] == white
    then DFS-Visit(u)
```

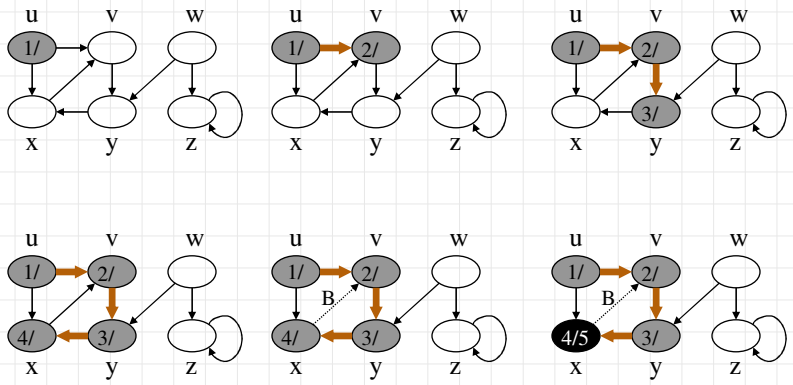
Inizializza tutti i vertici

```
DFS-Visit(u)
 color[u] = gray
 time++; d[u] = time
 foreach v ∈ Adj[u] do
    if color[v] == white
    then π[v] = u
         DFS-Visit(v)
 color[u] = black
 time++; f[u] = time
```

Visita ricorsivamente tutti i figli

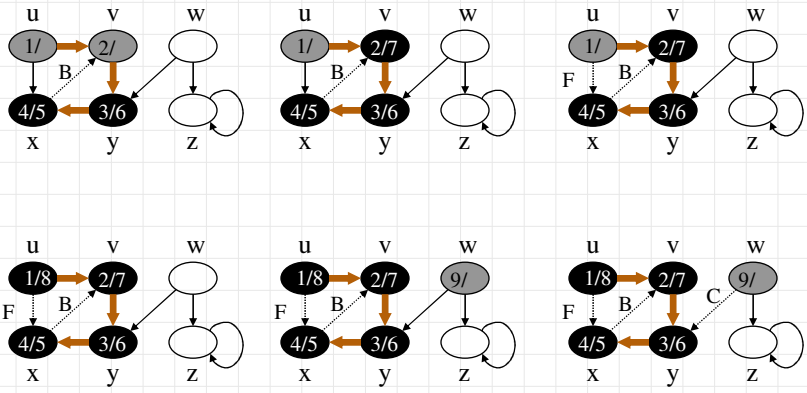
18

DFS: esempio



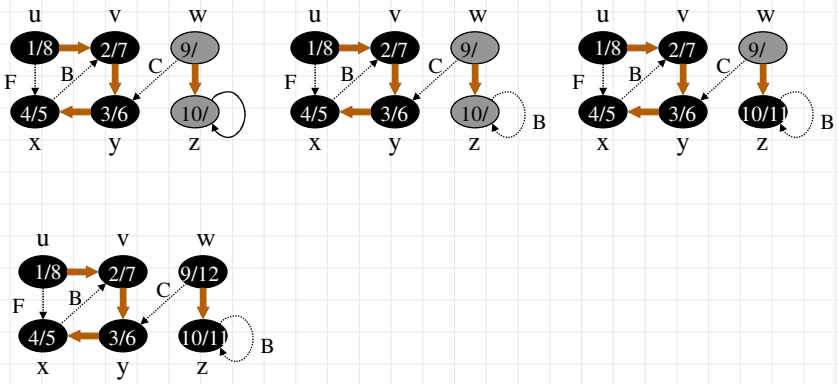
19

DFS: esempio



20

DFS: esempio



21

Algoritmo DFS

- Inizializza – colora di bianco tutti i vertici.
- Visita ogni vertice bianco usando DFS-Visit
- Ogni chiamata a DFS-Visit(u) inizializza un nuovo albero con radice nel vertice u
- Quando DFS finisce, ogni vertice *u* ha associato un tempo di inzio visita $d[u]$ e un tempo di fine visita $f[u]$
- Quando un vertice v viene scoperto, gli si associa in $\pi(v)$ l'id del predecessore.

22

Algoritmo DFS

Tempo di CPU

- il ciclo in DFS richiede un tempo $\Theta(V)$, escludendo il tempo per eseguire DFS-Visit
- DFS-Visit è chiamata una volta per ogni vertice
  - è chiamata solo per i vertici bianchi
  - etichetta immediatamente il vertice col grigio
- per ogni DFS-visit un ciclo percorre tutta Adj[*v*]
- il costo totale per DFS-Visit è $\Theta(E)$

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

- il tempo di CPU di DFS è $\Theta(V+E)$

Sottografo dei predecessori

Definito in modo leggermente diverso da BFS (forse):

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

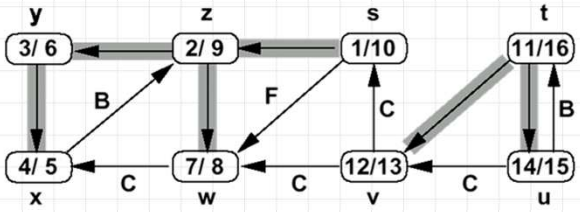
Il sottografo in questo caso forma una foresta depth-first composta di vari alberi depth-first

Gli archi in G_π sono chiamati archi dell'albero (tree edges)

DFS: tempi di visita

L'algoritmo DFS mantiene un (*metaforico*) orologio globale su tempo di inizio visita $d[u]$ e tempo di fine $f[u]$

Per ogni vertice u è verificata la diseguaglianza $d[u] < f[u]$



25

DFS: tempi di visita

Un vertice u è:

- bianco prima del tempo $d[u]$
- grigio fra il tempo $d[u]$ e il tempo $f[u]$
- nero dopo

Strutture:

- i vertici grigi formano una catena lineare
- implementabile come uno stack di vertici che non sono stati esplorati esaustivamente (DFS-Visit iniziata ma non completata)

26

DFS: teorema delle parentesi

I tempi di visita e di fine hanno una struttura a parentesi

- si può rappresentare il tempo di inizio visita di *u* con una parentesi aperta "(u"
- si può rappresentare il tempo di fine di *u* con una parentesi chiusa "u)"
- la storia delle visite e delle fini definisce una espressione ben formata (le parentesi sono annidate correttamente)

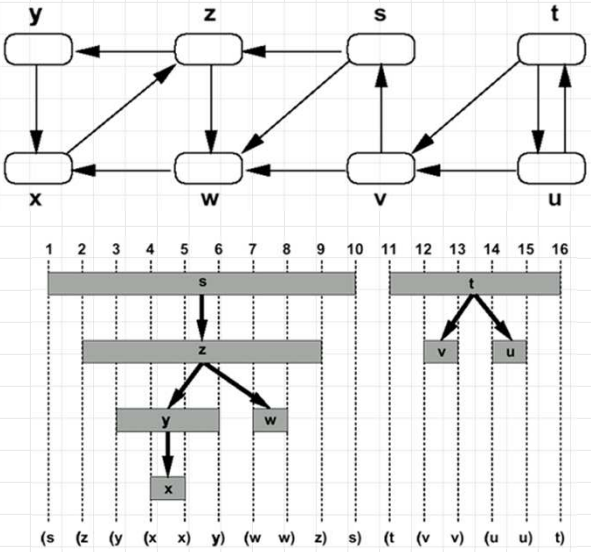
Teorema delle parentesi

In ogni DFS di un grafo G=(V,E), per ogni coppia di vertici u e v, una e una sola delle seguenti condizioni è soddisfatta:

- gli intervalli [d[u],f[u]] e [d[v],f[v]] sono disgiunti;
- l'intervallo [d[u],f[u]] è contenuto in [d[v],f[v]] e u è un discendente di v nell'albero DFS;
- l'intervallo [d[v],f[v]] è contenuto in [d[u],f[u]] e v è un discendente di u nell'albero DFS;

27

DFS: teorema delle parentesi



28

Teorema del cammino bianco

Teorema del cammino bianco

In una foresta DFS di un grafo G=(V,E), un vertice v è un discendente di un vertice u sse al tempo d[u] il vertice v è raggiungibile da u con un cammino contenente esclusivamente nodi bianchi.

DFS: classificazione degli archi

E' possibile utilizzare la visita per classificare gli archi (u,v) del grafo di input. 4 tipi di archi:

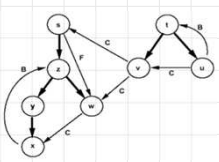
- Archi dell'albero (da grigio a bianco)

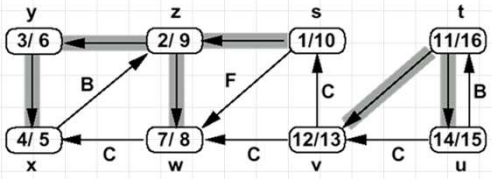
  v è stato scoperto esplorando l'arco (u,v)
- Archi all'indietro (da grigio a grigio)

  connettono un vertice u ad un antenato v in un albero DFS
- Archi in avanti (da grigio a nero)

  non sono archi dell'albero, connettono u ad un discendente v
- Archi di attraversamento

  tutti gli altri, possono connettere vertici in alberi DFS distinti

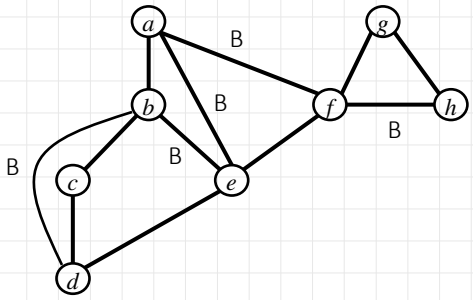




DFS: classificazione degli archi

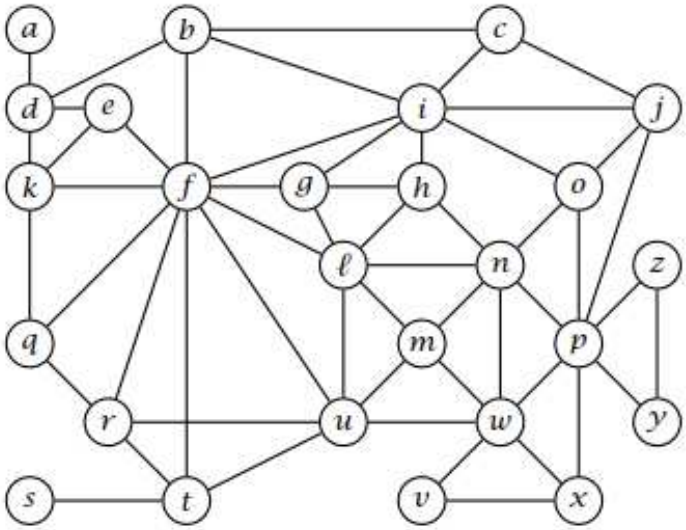
Teorema

In una DFS di un grafo ***non orientato*** G, ogni arco di G è un arco dell'albero oppure un arco all'indietro.



31

A graph



32

A graph

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*ℓ*
a						1						
b								1	1	1		
c									1		1	
d	1		1		1							1
e							1			1	1	
f					1					1	1	1
g		1										
h		1		1					1	1		1
i								1				
j		1					1	1				
k	1							1		1		
ℓ									1		1	