

$\hookrightarrow S_{in} = 2 \quad S_{out} = 1$

ORIENTATO

(directed)

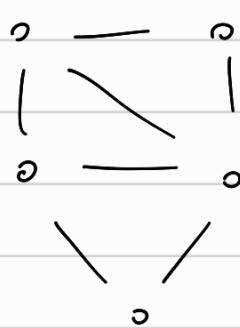
$$\text{arco} = (u, v)$$



il grado si divide

in uscente (S_{out})

ed entrante (S_{in})



NON ORIENTATO

$$(u, v) \equiv (v, u)$$

$$G = (V, E)$$

coppie nodi ordini
 $n \quad m$

In generale:

- i coppi sono accoppiati
- no multigradi (più ordini uguali fra 2 nodi:)
- grado = n° ordini di un nodo
- 2 nodi adiacenti = collegati da un arco

$$\sum S_{in}(n) = m = \sum_{u \in V} S_{out}(n)$$

• grado:

$$- \text{grafo non orientato} \quad \sum_{u \in V} S(n) = zm \quad (\begin{matrix} \text{ogni arco} \\ \text{contato 2 volte} \end{matrix})$$

$$- \text{grafo orientato} \quad \sum_{u \in V} S_{in}(n) = m = \sum_{u \in V} S_{out}(n)$$

• n° ordini min = 0

$$• n° ordini max (\text{non orientati}) = n + (n-1) + \dots + 1 = \frac{n(n+1)}{2}$$

collegati a tutti "meno uno"

$$• n° ordini max (\text{orientati}) = n^2 \quad (\text{dappi: ordini})$$

• $m \sim n^2$ il grafo è detto denso

La **componibilità** è espressa in termini di cammino

- **cammino**: sequenza di nodi v_0, v_1, \dots, v_n t.c. $v_i \in \Omega, n \geq 0$
(catena) $(v_i, v_{i+1}) \in E$
con $n = \text{lunghezza cammino}$

→ **degenero**: ho solo v_0 ($n=0$, nullo)

→ **semplice**: nodi dltt diversi tranne al più v_0 e v_n

→ **(orientati)** ho un ciclo se $v_0 = v_m$ (lunghezza min=1, il coppiaio)

→ **(non orientati)** ho un ciclo se $v_0 = v_m$ ed è lungo almeno 3 (**ciclo**)
(altrimenti coincerei uno stesso arco avendo e indietro \longleftrightarrow)

→ **v raggiungibile da v** se \exists cammino che congiunge $v \dots v$

- **connesso**: ogni nodo è raggiungibile da un altro
(non orientato)

(or) → **dolcemente connesso**: se considerando un non orientato, sarebbe
fortemente connesso $O \rightarrow O = O - O$ (arco singolo)
(avendo li sono nodi non raggiungibili con un cammino)

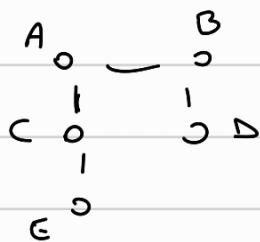
(or) → **fortemente connesso**: se \exists un cammino da ogni vertice v
a ogni vertice v'

- **sottografo** di G

è un grafo $G' = (V', E')$ con $V' \subseteq V, E' \subseteq E$

sottografo indett. da $V' \subseteq V$

- **albero libero**: particolare grafo non orientato connesso aciclico
(no radice prefissa, dunque puo' essendo)



→ albero con n nodi ha n-1 archi
(orientamento non connesso)

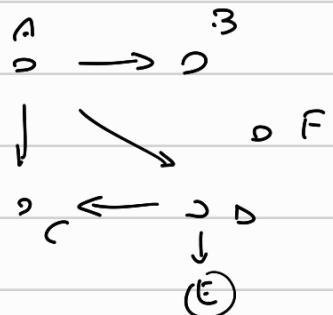
- **albero ricoprente** di G : è un sottografo di G che contiene molti i nodi ed è un albero libero.
Venne denominato **spanning tree**.

Rappresentazione di grafi: $G = (V, E)$

- 1) (nodi +) lista di archi

$F \ (A,C) \ (A,B) \ (D,C) \ (A,D), \ (D,E)$

(poco usato)



- 2) lista di adiacenze

A B, C, D

B /

C /

D E

E /

F /

3) notizie di educazione

A	B	C	D	E	F
1	2	1	1	0	0

(1 se c'e' vero)

B C D E F

Complessità

lista archi	liste adiacenze	matrici adiacenza
(quanto menzionato)		
Spazio $n+m$	$n+m$ $n+2m$	n^2
Suolo u m	$S(u)$ <small>scava gli archi del nodo u</small>	n
rimuovi lista adiacenti u m	$S(u)$	n
Elenco u, v m	$\min(S(u), S(v))$	1
Aggiungi nodo	1	n^2 <small>se dim fissata della velocità</small>
Aggiungi arco	1	1
Elimina nodo	m	n^2
Elimina arco	$S(u) + S(v)$ <small>(fase orientata $S(u)$)</small>	1

/ 15-03

- La matrice di adiacenza conviene con un grafo denso
- Un grafo **REGOLO** presenta un veloce sugli archi.

Visite di grafi

Ve ne sono di due tipi : per **AMPIEZZA** e per **PROFOUNDITA'**
(breadth first) (depth first)

E possono essere implementati
in modo **RICORSIVO** o **ITERATIVO**

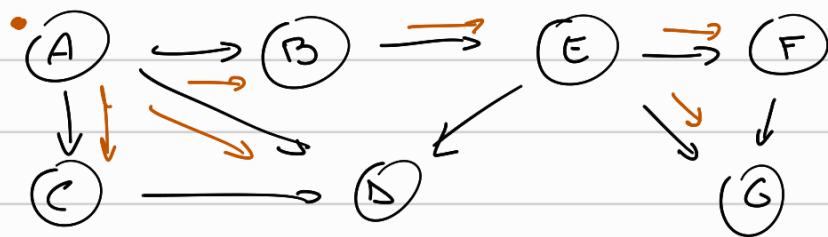
Nel pero' esistere di visitare due volte lo stesso nodo (entro ciclo),
si utilizza una **MARCATURA** dei nodi, solitamente a 3 valori:

bianco	nodo non toccato dalla visita
grigio	visita iniziata
nero	visita terminata

Nel caso di visite **iterative** l'algoritmo usa una struttura ausiliaria
per estrarre i nodi della **FONDA** (pila o coda)

Note gli algoritmi visti funzionano sia su grafi orientati che non

Esempio:



Visita compresa: A, B, C, D, E, F, G (natura iterativa)
(visito primo, gli adiacenti)



Visita profondità: $A, B, E, D, \bar{F}, \bar{G}, C$ (natura ricorsiva)
(visito primo, il primo adiacente) $\xrightarrow{\text{torno su } E}$ \downarrow $\xrightarrow{\text{torno su } A \text{ e poi } C}$

- Per la visita in compresa uss una coda (olgo in testa, add in coda)
 $\rightarrow A \rightarrow B/C/D \rightarrow C/D/E \rightarrow E/F/G$ (natura iterativa)
tolgo dalla coda e metto gli adiacenti

Algoritmo - FRONTEIRA = CODA

BFS (G, S)

for each (u nodo in G) move u bianco

$Q = \text{coda vuota}$

$Q = \text{add}(S);$ move S grigio

while (Q non vuota)

$u = Q.$ deprece()

"visita $u"$ \rightarrow fai le operazioni relative alla visita

for each (u, v) arco in G

| if (v bianco)

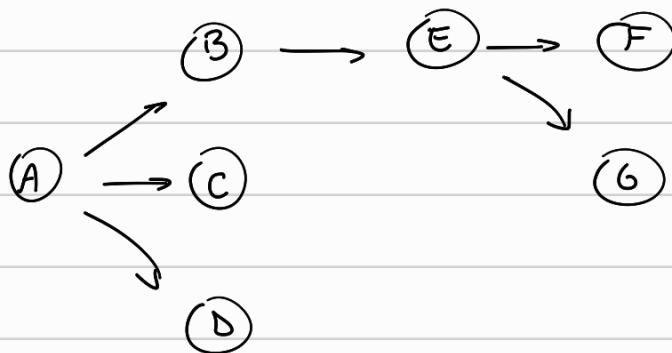
| $Q.$ add (v)

| marco v grigio

marco u nero

Ogni nodo entra ed esce dalla coda una sola volta

Una visita di un grafo costruisce un **ALBERO DI VISITA**



albero di ricoperto rispetto
al grafo (assumendo la parte
connessa e raggiungibile a partire da s)

L'albero può essere costruito utilizzando un array dei padri (specificiamo il genitore di ogni nodo)

- parent [s] = null

- parent [v] = u

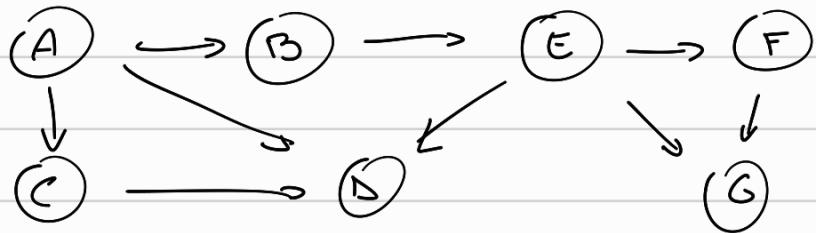
Il parent è assegnato una sola volta

nodi bianchi \rightarrow non ancora in coda

nodi grigi \rightarrow in coda

nodi neri \rightarrow usciti dalla coda

Per la visita in profondità uso un algoritmo iterativo e una pila
(add e delete da una pila)



(solo per le
nodi e delete da una pila)

Pila: ~~A~~ \leftarrow ~~B~~ \leftarrow ~~C~~ \leftarrow ~~D~~ \leftarrow ~~E~~ \leftarrow ~~F~~ \leftarrow ~~G~~ (inizio)
già nero

ottenendo A, B, E, D, F, G, C, D

Algoritmo - FRANCIA = PILA

DFS (G, s)

foreach (u nodo in G) u bianco (nodo: bianchi)

$S =$ pila vuota

\triangleright push (s); s grigio • parent [s] = null
while (S non vuota)

$u = S.pop();$ "visita u "

if u non nero

foreach (u, v) arco in G

| if (v bianco) marca v grigio

| if (v grigio) \triangleright push (v) • parent [v] = u ;

marca v nero

Il nodo puo' entrare in S piu' volte

Il parent puo' cambiare piu' volte

Nella pila trova anche nodi neri

Complessità (DFS)

- Operazioni su grafo in tempo costante (assunzione)
- Moltiplicazione nodi: $3n$ ($1B, -6, 1n$ ogni nodo)
- push (v); parent [v] = u
 - $\underbrace{}_m$
 - $\forall v$ capita d volte volte quanti sono gli archi entranti
- $\forall u$ $\forall e(u, v)$ v è in G = calcolo adiacenti
 - $\rightarrow \times$ liste adiacenze: $\delta(u)$ ($\text{fino a } m$)
 - TOTALE complessità: $\Theta(n+m) \Rightarrow (3n+2m)$
 - $\rightarrow \times$ matrici di adiacenza: n ($\text{fino a } n^2$) \Rightarrow calcolo adiacenti di tutti i nodi $\Rightarrow TOT (n \cdot n^2) = \Theta(n^3)$
 - $\rightarrow \times$ liste archi: m ($\text{fino a } m \cdot n$) \Rightarrow $TOT (n + n \cdot m) = \Theta(m \cdot n)$

La visita in ampiezza calcola la distanza di ogni nodo (di u da s)
ovvero cammino minimo.

Viene costruito un albero a livelli: il livello può essere esplicitato dell'algoritmo: (u, v) level [v] = level [u] + 1

Algoritmo ricorsivo DFS

DFS ricorsiva con visita completa (raggiunge tutti i nodi, faccio una DFS(G) anche se grafo non连通)

(visita completa) ↴

$\text{DFS}(G)$:

for each u node in G mark u bianco; $\text{parent}[u] = \text{null}$
for each u node in G
if (u bianco) $\text{DFS}(G, u)$

(visita a partire da u)

$\text{DFS}(G, u)$

mark u grigio;

for each (u, v) arco in G

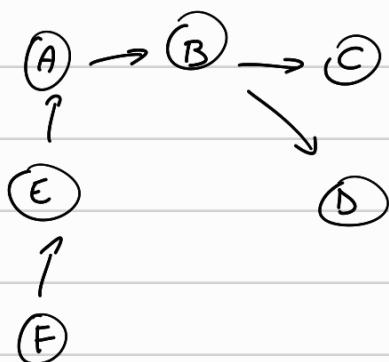
if (v bianco)

$\text{DFS}(G, v)$

$\text{parent}[v] = u$

mark u nero

Esempio



prendo i nodi casualmente:

- pescò A

$\rightarrow A, B, C, D$

- pescò F

$\rightarrow F, G$

- pescò C

\rightarrow nulla

- pescò B

$\rightarrow B, C, D$

... -

Cio' che ottengo ora e' un singolo solido ma con FORCE STA:

