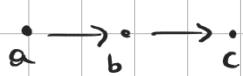
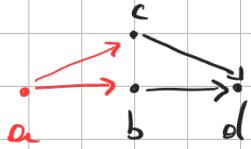


E,



graph che ammette solo un ordinamento topologico abc



ammette sia abcd che acbd

ALGORITMO DI ORDINAMENTO TOPOLOGICO

La dimostrazione sull'ordinamento topologico se venisse implementata come algoritmo verrebbe eseguito in tempo NV^2 , il che non è economico.
Vediamo come migliorare l'algoritmo.

L'algoritmo cerca i nodi con 0 archi entranti. Se avessimo quest'informazione si ridurrebbe tutto ad una ricerca lineare su un array che tiene traccia dei gradi entranti.

Non è in realtà necessario rimuovere i nodi con grado entrante 0 per costruire G' , bensì basta aggiornare il grado entrante dei nodi adiacenti a v, che vogliamo rimuovere.

Questi nodi tuttavia non è detto abbiano g.e. 0 dopo l'aggiornamento, il che è una complicazione perché mai stiamo cercando quelli.

Per ovviare a questo problema cerchiamo tutti i nodi con

grado entrante 0 e li salviamo in una struttura dati (una
queue ad esempio). Tutto ciò che dobbiamo fare ora è inserire
nella sequenza i nodi salvati nella queue e avremo l'ordinamento

ORD-TOPOLOGICO (G, GE)

GE è l'array dei gradi entranti

$S = \emptyset$

S è la queue

GRADO ENTRANTE (G, GE)

FOR $v \in V$ DO

IF $GE[v] = 0$ THEN

$S = ACCODA(S, v)$

WHILE $S \neq \emptyset$ DO

$x = TESTA(S)$

PRINT x

FOR EACH $y \in ADJ(x)$ DO

Aggiorno il grado entrante degli
adiacenti

$GE(y) = GE(y) - 1$

IF $GE(y) = 0$

$S = ACCODA(S, y)$

$S = DECODA(S)$

Supponendo che GRADO ENTRANTE (G, GE) costi $\Theta(|V| + |E|)$.

Il primo for costa $\Theta(|V|)$ perché accede a costante e viene ripetuto per $|V|$ volte. Il WHILE costa lineare sul for.

Il for piu' interno costa al piu' $\sum_{v \in V} (\text{Adj}(v) + 1) = |E| + |V|$

Quindi in totale il costo e' $\Theta(|V| + |E|)$

Vediamo come e' fatta la funzione che calcola il grado entrante di un nodo

GRADO ENTRANTE (G, GE)

FOR EACH $v \in V$ DO

$$GE(v) = 0$$

} $\Theta(|V|)$

FOR EACH $v \in V$ DO

FOR EACH $u \in \text{Adj}(v)$ DO

$$GE(u) = GE(u) + 1$$

} $\Theta(|V| + |E|)$

L'algoritmo ha effettivamente costo $\Theta(|V| + |E|)$

ORDINAMENTO TOPOLOGICO CON DFS

Grazie alla DFS possiamo costruire l'ordinamento al contrario.

Si noti che i vertici pozzi non hanno archi uscenti quindi possono sicuramente essere messi alla fine.

La DFS colora questi nodi di nero alla fine del percorso corrente degli adiacenti del nodo attuale v , quando tutti gli adiacenti sono neri

ORDINAMENTO TOPOLOGICO DFS (G)

INIT (G)

$OT = \emptyset$

OT e' uno stack

FOR EACH $v \in V$ DO

IF COLOR(v) = b

$OT = VISITA(G, v, OT)$

RETURN OT

VISITA (G, v, OT)

COLOR(v) = g

FOR EACH $u \in \text{Adj}(v)$ DO

IF COLOR(u) = b

OT = VISITA(G, u, OT)

COLOR(v) = m

OT = PUSH(OT, v)

RETURN OT

TEOREMA

Dato un grafo aciclico G , Al termine di ORDINAMENTO DFS (G)

OT è un ordinamento topologico di G .

Quindi $x \leq_{OT} y$ se x sta sopra y in OT

Potremmo dire che OT è un ordinamento topologico se

$\forall (x, y) \in E \quad x \leq_{OT} y$

DMOSTRAZIONE

Se x sta sopra y allora $f(y) < f(x)$ nella DFS.

Dobbiamo verificare questa proprietà al termine di DFS per ogni nodo

Vediamo cosa succede all' attraversamento di (x,y) :

- Se y e' bianco, x e' stato scoperto prima di y . Per il teorema della struttura a parentesi se vale $d(x) < d(y)$ possiamo completare solo con $f(y) < f(x)$ ✓
- Se y e' nero, e' avvenuto $d(x)$. Abbiamo due possibili casi:
 - y già era nero prima della scoperta di x , quindi $f(y) < d(x)$
 x a un certo punto sarà per forza nero quindi $f(y) < d(x) < f(x)$.
 - y era bianco quando e' stato scoperto x e quindi $d(x) < d(y)$.
Ha adesso e' nero quindi $d(x) < d(y) < f(y)$ e possiamo completare solo in un modo, ovvero $d(x) < d(y) < f(y) < f(x)$
- y non può essere grigio perché significa che (x,y) e' di ritorno, quindi G e' ciclico (va contro la nostra ipotesi)

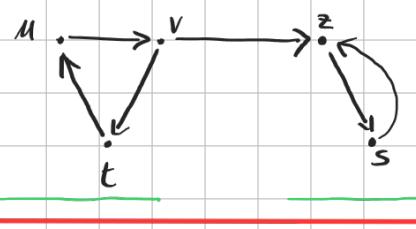
COMPONENTI (FORTEMENTE) CONNESSE

Una **COMPONENTE CONNESSA** è un sottografo massimale e connesso

Un grafo orientato è **CONNESSO** se $\forall v, u \in V$ u è raggiungibile da v .

Si dice **FORTEMENTE** connesso se il grafo è anche orientato

Per **MASSIMALE** intendiamo che non possono essere aggiunti altri nodi al sottografo senza perdere la forte connessione



La relazione di reciproca raggiungibilità è una relazione d'equivalenza.

Quindi per ogni oggetto possiamo definire una classe d'equivalenza.

$$[x]_{\text{RecReach}} = \{y \in G \mid (x, y) \in \text{RecReach}\}$$

Tutti gli elementi di una classe d'equivalenza possono essere rappresentanti della classe. Ciò significa che tutti i nodi di una **SCC** possono essere rappresentati da un nodo e ridurre il problema a uno banale. Nel caso del grafo precedente avremo

$$[v] — [z]$$

SCC IN GRAFI NON ORIENTATI

Nei grafici non orientati due sottografi sono SCC diverse se non ci sono archi tra le due classi, se ci fosse sarebbe un'unica componente perché essendo un grafo non orientato possiamo navigare gli archi in ogni verso.

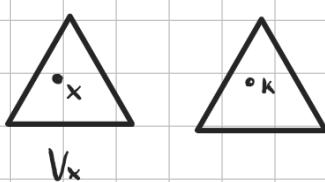
L'algoritmo di DFS ci dà tutte le informazioni necessarie per trovare le classi d'equivalenza:

Immaginiamo di applicare $\text{DFS}(G)$ su G grafo non orientato.

La classe di x vertice casuale è un sottografo di G , $[x] = \langle V_x, E_x \rangle$ con:

- V_x insieme dei vertici dell'albero che contiene x .

Perché ogni vertice può stare solo in un albero della foresta e tutti i vertici sono in un albero.



L'albero di sinistra è "la classe di x " perché è non orientato quindi ogni modo può comunque raggiungere gli altri essendoci almeno un vertice in comune (almeno la radice).

L'albero di destra **NON** può essere parte della classe di x perché non ci sono archi che collegano i due alberi, quindi per il

discorso precedente sono classi diverse.

Quindi i vertici di una componente sono tutti e soli i nodi
di un albero.

• E_x

Sono tutti gli archi presenti in E che connettono i vertici di V_x

Quindi nel caso dei grafici non orientati il problema si riduce
a trovare gli alberi del grafo poiché la componente è il
sottografo di G inselto su V_x .