

Ordine topologico

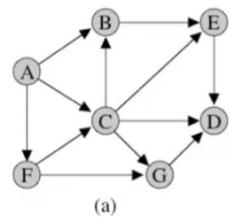
/ 28-03

- per grafi orientati aciclici (DAG)

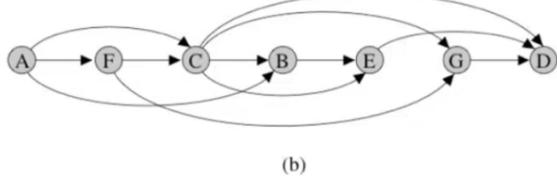
- si ottiene relazione "v raggiungibile da u" (comune da u a v)

↳ mi stabilisce un ordine parziale = relazione riflessiva
(u raggiungibile da se stesso) e transitiva (se $u \rightarrow w$ e
 $w \rightarrow v$ allora $u \rightarrow v$); non è detto che fra due elementi ha relazione

Intuitivamente si può pensare che vi sia un ordine di "precedenze"
fra i nodi



(a) un grafo aciclico



(b)

(b) ordine topologico dei nodi del grafo (a)

e) pensando i nodi come precedenze, per arrivare a B devo passare per A prima. Per arrivare a D devo passare per E ma ↳ perché ho un orlo fra i due non ho relazione fra B e F (nel senso non ho un ordine specifico di chi deve venire prima fra i due) e fra E e G lo "diretto".

Si osserva che si deve partire da A perché non ha nessuno che lo precede nella relazione (non dipende da nessuno)

Trovare un ordine topologico significa, dal grafo DAG i nodi senza uscite e gli archi sono relazioni di precedenza, stabilire una sequenza, un ordine totale, dei nodi.

Lo si dice uno prima e uno dopo

b) Ad esempio quello in figura: le precedenze sono rispettate
Ma ve n'è altri possibili?

$A \rightarrow B$ no, perché B dipende anche da C , viola $C < B$

$A \rightarrow C$ no, perché C dipende da F , viola il Requisito $F < C$

A, F, C, B, G, E, D
 \downarrow ↳ già passato per B ✓
posso farlo perché sono già passato
per C (e per B) ✓

Quindi dato $G = (V, E)$ è un ordine totale su V t.c.

$\forall u, v \in V \quad (u, v) \in E \rightarrow u < v$

N.B. Meno un DAG è connesso, più ordini ci sono (ha più
arbitrarietà). Se non ho archi non ho vincoli, ho tanti ordini
quante sono le permutazioni dei nodi

E' sempre un ordine topologico per un DAG, perché:
è un grafo acchico (non ha inversioni di ordini)



Dimostriamo x assurdo

- detto source nodo con nessun arco entrante } \Rightarrow sempre perché
- detto sink nodo con nessun arco uscente } ciclico

per assurdo supponiamo \exists source, prescelto un modo e cosa u_0 ,
c'è per forza un arco entrante (u_1, u_0) , ma allora
c'è un arco uscente in u_1 , detto $(u_2, u_1) \dots$

Ogni volta un modo diverso dei precedenti, altrimenti avrei un ciclo,
ma i nodi sono di numero finito, prima o poi non c'è arco
entrante altrimenti ho ciclo.

Ne segue un algoritmo la cui idea è estuare ad ogni ciclo il modo
sorgente. Per farci ciò ed evitare modifiche al grafo, si calcola
x ogni nodo il suo indegree (ordini entranti) e ad ogni passo
mezzo di rimuovere (u, v) decrementa indegree. Quando questo è zero
allora passo inserire il modo nel mio insieme di nodi sorgenti

topological sort (G)

$S =$ insieme nodo; $Ord =$ sequenza vuota

for each (u nodo in G)

$\text{indegree}[u] =$ indegree di u

for each (u nodo in G)

 if ($\text{indegree}[u] = 0$) \triangleright .add(u)

while (S non vuoto)

$u = S.\text{remove}()$ \nearrow // estraggo un qualsiasi elemento di S $\parallel \Theta(n)$

$Ord.\text{add}()$ \nearrow // aggiungo in fondo

 for each $((u, v)$ arco in G) $\quad //$ arco entrante in v

$\text{indegree}[v] --$

$\parallel \Theta(m)$ visita tutti gli archi

\parallel assunzione add costante

$\Theta(n)$

$\parallel \Theta(n)$

$\parallel \Theta(m)$

if ($\text{indegree}[v] = 0$) S.add(v) $\rightarrow 1$
return Ord

| $\Theta(n)$

Complessità: $\Theta(n+m)$ complessità di una visita

Un altro algoritmo x topological sort: visita in profondità ricorsiva con timestamp; uso un contatore e ogni volta segno il tempo di inizio visita e fine visita di un nodo

DFS(G) // visita completa anche x grafi non连通;

for each (u nodo in G) $(\text{time var globale})$

mark u bianco; $\rightarrow \text{time} = 0$

for each (u nodo in G)

if (u bianco) DFS(G, u)

// pescò un nodo bianco e lo so

DFS(G, u) // visita della parte raggiungibile da u

visita u ; mark u come grigio // inizio visita

for each $((u, v)$ arco in G)

if (v bianco)

DFS(G, v)

$\text{time}++$ (primo nodo ha start 1)

$\text{start}[u] = \text{time}$

mark u nero

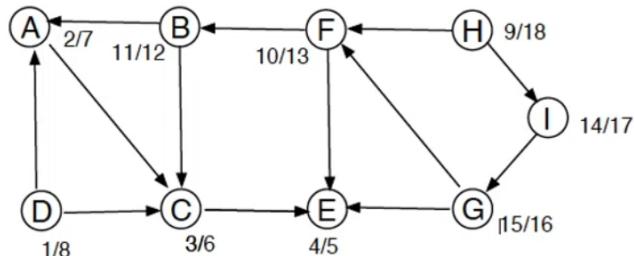
// fine visita

$\text{time}++$

$\text{end}[u] = \text{time}$

Cio' che succede e' che prendendo i tempi di fine visita in ordine inverso otteniamo un ordine topologico. Posso iniziare l'algoritmo da qualsiasi nodo (essendo DFS)

iniziamo per esempio la visita da D



H, I, G, F, B, D, A, C, E → ord. top.

per ogni $(u, v) \in E$

Questo funziona perché essendo DAG aciclico, "in qualunque visita in G si ha: $\text{end}[v] < \text{end}[u]$ (quindi nell'ord top $u < v$, inverso)

Prove di correttezza:

- se inizio visita da u

trovo (u, v) , v bianco e lo visito, per concludere la visita di u, deve concludersi la visita di v ma allora $\text{end}[v] < \text{end}[u]$

- se inizio visita da v

durante la visita non posso trovare un cammino da v a u (avendo (u, v)) perché G aciclico, quindi finisce prima necessariamente la visita su v prima di toccare u, quindi $\text{end}[v] < \text{end}[u]$
(e anche $\text{end}[v] < \text{start}[u]$)

Complessità = visita DFS ($\mathcal{O}(n+m)$)