Grafi: ordinamento topologico

Corso di **Algoritmi e strutture dati**
Corso di Laurea in **Informatica**
Docenti: Ugo de'Liguoro, András Horváth

Indice

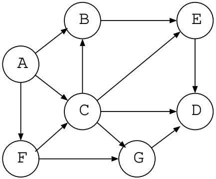
Sommario

Obiettivo:

- capire il concetto dell'ordinamento topologico
- sviluppare un algoritmo per trovare l'ordinamento topologico sulla base di una visita DFS

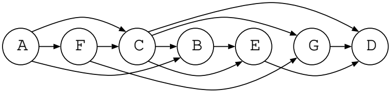
1. Definizione di ordinamento topologico I

- una funzione $\sigma : V \rightarrow \{1, ..., |V|\}$ tale che $\sigma(u) < \sigma(v)$ se esiste un cammino da u a v in G
- un esempio:



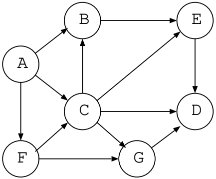
$\sigma(A) = 1$, $\sigma(F) = 2$,
$\sigma(C) = 3$, $\sigma(B) = 4$,
$\sigma(E) = 5$, $\sigma(G) = 6$,
$\sigma(D) = 7$

- ridisegnando lo stesso grafo secondo l'ordine σ:



1. Definizione equivalente di ordinamento topologico

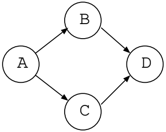
- un ordinamento lineare dei vertici di un grafo tale che $\forall(u, v) \in E$, u precede v nell'ordinamento
- un esempio:



ordinamento: A, F, C, B, E, G, D

1. Proprietà di ordinamento topologico

- l'ordinamento topologico può esistere solo se il grafo è aciclico (DAG)
  - se esiste un cammino da u a v allora σ deve essere tale che $\sigma(u) < \sigma(v)$
  - se esiste un cammino da v a u allora σ deve essere tale che $\sigma(v) < \sigma(u)$
  - $\lightning$ abbiamo una contraddizione!
- possono esistere diversi ordinamenti topologici dello stesso grafo:



$$\sigma_1(A) = 1 \quad \sigma_1(C) = 2 \quad \sigma_1(B) = 3 \quad \sigma_1(D) = 4$$

$$\sigma_2(A) = 1 \quad \sigma_2(B) = 2 \quad \sigma_2(C) = 3 \quad \sigma_2(D) = 4$$

2. Algoritmo “naive”

- il primo nodo deve essere un nodo senza archi entranti
- denotiamo questo nodo con o_1
- il secondo nodo può avere un arco entrante solo da o_1
- denotiamo questo nodo con o_2
- il terzo nodo può avere archi entranti solo da o_1 e o_2
- denotiamo questo nodo con o_3
- il quarto nodo può avere archi entranti solo da o_1, o_2 e o_3
- denotiamo questo nodo con o_4
- ...

2. Algoritmo “naive”

ORDINAMENTO-TOPOLOGICO(G)

$H \leftarrow G$ $\triangleright$ una copia di G in H

$o \leftarrow$ lista vuota di vertici

while $\exists u : \neg \exists v : (v, u) \in E(H)$ **do**

$\triangleright$ esiste un nodo u senza archi entranti

appendi u come ultimo elemento di o

rimuovi u da H (con tutti suoi archi uscenti)

if H non è vuoto **then**

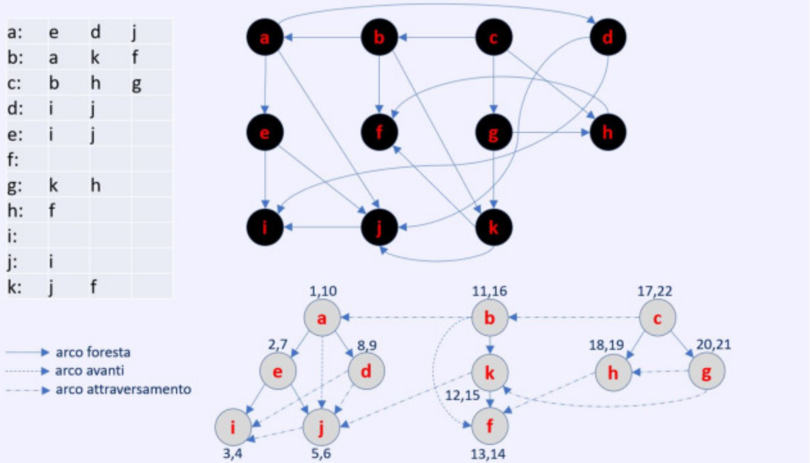
stampa “il grafo non è aciclico”

restituisci o

Complessità?

3. Algoritmo basato su DFS

a:	e	d	j
b:	a	k	f
c:	b	h	g
d:	i	j	
e:	i	j	
f:			
g:	k	h	
h:	f		
i:			
j:	i		
k:	j	f	



3. Algoritmo basato su DFS

- il nodo con attributo fine visita più grande sicuramente non ha archi entranti
- può essere primo nell'ordine topologico, denotiamo questo nodo con o_1
- il nodo con attributo fine visita secondo più grande può avere un arco entrante solo da o_1
- può essere secondo nell'ordine topologico, denotiamo questo nodo con o_2
- il nodo con attributo fine visita terzo più grande può avere archi entranti solo da o_1 e o_2
- può essere terzo nell'ordine topologico, denotiamo questo nodo con o_3
- ...

3. Algoritmo basato su DFS

- possiamo adattare l'algoritmo di visita in profondità al problema di ordinamento topologico
- basta creare una lista dei vertici in ordine decrescente dei tempi di fine visita
- (anche un controllo su aciclicità sarebbe facile da fare)

3. Algoritmo basato su DFS

TOPOLOGICAL-SORT(G)
  $L \leftarrow$ lista vuota di vertici
  INIZIALIZZA(G)
  for $\forall u \in V$ **do**
    if $u.color = bianco$ **then**
      DFS-TOPOLOGICAL(G, u, L)
  restituisci L

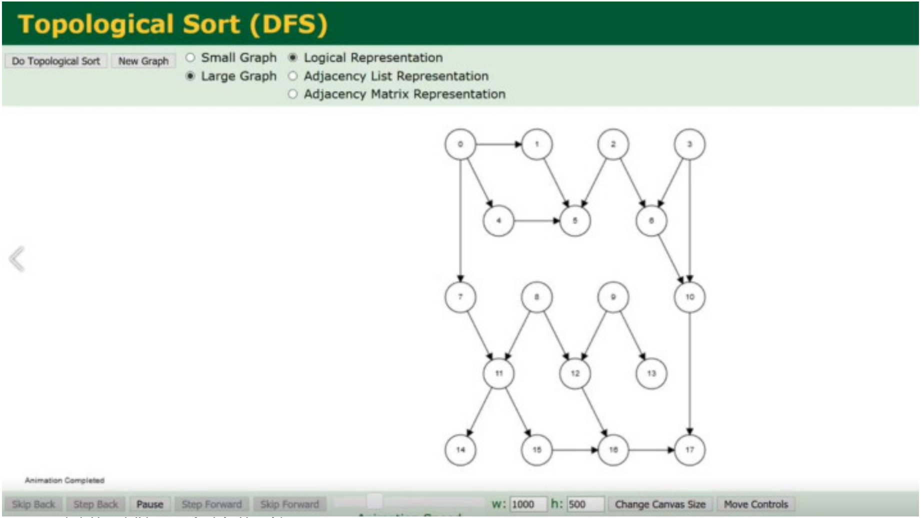
3. Algoritmo basato su DFS

- DFS-TOPOLOGICAL(G, s, L)
  $s.color \leftarrow grigio$
  $s.d \leftarrow time$
  $time \leftarrow time + 1$
  for $\forall v : v$ è *bianco* ed è $\in$ adj[s] **do**
    $v.\pi = s$
    DFS-TOPOLOGICAL(G, v, L)
  $s.color \leftarrow nero$
  $s.f \leftarrow time$
  $time \leftarrow time + 1$
  in testa di L inserisci s
- complessità è uguale alla complessità della visita in profondità

3. Correttezza dell'algoritmo basato su DFS

- basta dimostrare che una (qualunque) DFS di un grafo orientato e aciclico associa ai nodi tempi di fine tali che $v.f < u.f$ per ogni arco $(u, v) \in E$
- supponiamo per assurdo che per un arco (u, v) si abbia $v.f > u.f$; questo può succedere in due modi:
  - $u.d < u.f < v.d < v.f$ (l'intervallo di u precede l'intervallo di v): impossibile perchè u non può diventare nero prima che tutti i suoi adiacenti siano scoperti
  - $v.d < u.d < u.f < v.f$ (l'intervallo di u è contenuto nell'intervallo di v): impossibile perchè u sarebbe un discendente di v in un albero di scoperta e l'arco (u, v) sarebbe un arco all'indietro (avere un arco all'indietro vuole dire che il grafo non è aciclico)

3. Algoritmo basato su DFS



3. Algoritmo basato su DFS